EXPLOITING WIRELESS BROADCAST PROPERTY TO IMPROVE PERFORMANCE OF DISTRIBUTED ALGORITHMS AND MAC PROTOCOLS IN WIRELESS NETWORKS

BY

GHAZALE HOSSEINABADI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Doctoral Committee:

Professor Nitin Vaidya, Chair
Assistant Professor Matthew Caesar
Assistant Professor Sayan Mitra
Professor David M. Nicol

ABSTRACT

Because a wireless channel is a shared medium, messages sent on the wireless links might be overheard by the neighboring stations. The information obtained from the overheard messages can be used in order to design more efficient distributed algorithms as well as MAC protocols for wireless networks. We exploit the wireless broadcast property in three different aspects. First, we design mutual exclusion algorithms for wireless networks in which opportunistic packet overhearing is exploited to decrease the number of transmitted messages as well as the delay of the algorithm. Second, we design a distributed and dynamically adaptive MAC protocol for wireless networks, called Token-DCF. In Token-DCF an implicit token passing algorithm is proposed to reduce idle and collision times of the random access mechanism of IEEE 802.11 DCF protocol. In Token-DCF, packet overhearing is employed to exchange scheduling information across the network. Third, we consider a dense deployment of wireless LANs and we propose Concurrent-MAC, a MAC protocol for increasing concurrent transmissions in dense wireless LANs. In Concurrent-MAC, based on SINR values between stations and access points (APs), sets of concurrent transmitters are identified by the backhaul of APs. A station gaining access to the channel schedules a set of its neighbors for concurrent transmissions. Neighbors chosen for concurrent transmission can start transmitting on the channel immediately after they overhear the privilege given to them for concurrent transmission.

To my parents, for their love and support

ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor, Prof. Nitin Vaidya, for his endless guidance and support of my Ph.D. study and research. I appreciate all his contributions of time, ideas, immense knowledge and funding which made this thesis possible.

Besides my advisor, I thank Prof. Caesar, Prof. Mitra and Prof. Nicol for serving on my doctoral committee, for their valuable perspectives on improving the dissertation and for their insightful comments.

The members of the Distributed Algorithms and Wireless Networking (DAWN) have greatly contributed to my professional experience at University of Illinois. I am very grateful to past and present members, Adrian Djokic, Tae Hyun Kim, Guanfeng Liang, Vijay Raman and Shehla Saleem Rana, for their kind assistance and encouragement.

I acknowledge the National Science Foundation, U.S. Army Research Office and Boeing for financially supporting this research. I also thank Carol Wisniewski for her kind help with various administrative matters.

This thesis would not have been possible without the support and help of my dearest friends. Special thanks go to Figen Oktem, Leila Hajibabai, Maryam Karimzadegan, Marjan Baghai and Sanaz Barghi for helping me get through the difficult times and for all the emotional support in the past five years.

I owe my deepest gratitude to my mom, dad, brother and sister; without them, writing this thesis would have been impossible. My family stood by me throughout my graduate studies, supporting me, believing in me, and loving me. I would like to thank them for being the best family one could ever imagine and for unconditionally supporting me every step of the way.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The past few years have seen a tremendous growth in the deployment of wireless networks such as wireless local area networks (WLANs) and wireless ad hoc networks. WLANs are widely used in homes, offices and public areas, offering flexibility and mobility to users, fast communication speed, and inexpensive deployment. Wireless ad hoc networks are one of the most innovative networking technologies which have broad applications in several domains, such as battlefield communication, search and rescue, and range expansion for rural networks.

Since the deployment of wireless networks, several MAC protocols and distributed algorithms for wireless networks have been proposed in the literature. This research focuses on introducing new techniques to improve the performance of MAC protocols as well as distributed algorithms in wireless networks.

Because the wireless channel is a shared medium, messages sent on the wireless links might be overheard by the neighboring stations. Wireless broadcast property is one of the main characteristics of the wireless channel that enables opportunistic message overhearing. Via opportunistic message overhearing, network nodes might overhear unicast messages not intended for them. Message overhearing can be exploited to transfer information among network nodes, without transmitting extra control messages. In this research, we show that by exploiting wireless broadcast property and opportunistic overhearing feature of the wireless medium, more efficient mutual exclusion algorithms and MAC protocols can be designed for wireless networks. We have identified the following three research problems in which message overhearing can be exploited to improve the performance:

1. How to improve the performance of mutual exclusion algorithms in wireless networks, where opportunistic overhearing can help to transfer information about the current status of the algorithm?

2. How to exploit wireless opportunistic overhearing to decrease collision time and idle time of IEEE 802.11 DCF protocol in wireless networks?

3. How to design more efficient MAC protocols for dense wireless LANs, where client stations are covered by several access points and opportunistic overhearing can be used to enable reliable concurrent transmissions in order to achieve higher throughput?

We briefly summarize each of these three problems below.

1.1 Research Summary

1.1.1 Improving Performance of Mutual Exclusion in Wireless Networks

Mutual exclusion (MUTEX) is the problem of ensuring that no two processes or threads can be in their critical section (CS) at the same time, where a critical section refers to a period of time when the process accesses a shared resource, such as shared memory. Most of the existing MUTEX algorithms are designed for typical wired networks [1],[2],[3],[4],[5]. Design of mutual exclusion algorithms for mobile ad hoc networks had received some interest in the past few years [6], [7]. Although the underlying network in these algorithms is wireless, the proposed algorithms are only mobility aware solutions, where the goal is to deal with the problems caused by node mobility, such as link failures and link formations. In this work, we show that the broadcast property of the wireless medium can be exploited to improve the performance of the MUTEX algorithms in wireless networks. We design two token based mutual exclusion algorithms for wireless networks. Our mutual exclusion algorithms are designed such that the neighboring nodes that overhear messages can learn more recent information about the current status of the algorithm. The designed mutual exclusion algorithms are called *Token Overhearing Algorithm (TOA)* and *Token and Request Overhearing Algorithm (TROA). TOA* is based on the MUTEX algorithm designed by Raymond [1]. In Raymond's algorithm, messages are transmitted over a static spanning tree of the network. *TOA* is based on overhearing of the token transmission and the spanning tree maintained by the algorithm changes

when token transmission is overheard by the neighboring nodes. *TROA* is based on Trehel-Naimi's algorithm [2]. In Trehel-Naimi's algorithm, when a node requires entry to the CS, the node sends a request message to the last known owner of the token. In *TROA*, overhearing of both request and token messages is exploited in order to obtain recent information about the latest token holder in the network. The performance metrics that we improve in this work are the number of transmitted messages and delay per CS entry. We measure the performance of our protocols via both ns-2 simulations and testbed implementation. We identify the scenarios in which opportunistic overhearing results in significant improvement.

1.1.2 Token-DCF: An Opportunistic MAC Protocol For Wireless Networks

Next, we propose a distributed MAC protocol, called Token-DCF, in which we show that by using wireless broadcast property and opportunistic message overhearing feature of the wireless channel, the idle time and collision time of IEEE 802.11 DCF protocol can be decreased. IEEE 802.11 DCF is the MAC protocol currently used in wireless LANs. IEEE 802.11 DCF employs a binary exponential backoff algorithm to resolve channel contention. 802.11 DCF specifies random backoff, which forces a station to defer its access to the channel for a random period of time. Two types of overhead are associated with random access protocols. One is channel idle time (i.e., backoff time) which is the time when contending stations are waiting to transmit. Another is collision, which happens when multiple stations transmit simultaneously. Due to idle and collision times, 802.11 DCF performs poorly when it comes to channel utilization, system throughput, and channel access time. To overcome these sources of inefficiency in 802.11 DCF, we propose a distributed and dynamically adaptive MAC protocol for wireless networks, called Token-DCF. The main focus of our approach is on reducing idle and collision times by introducing an implicit token passing algorithm. In Token-DCF, a transmitting station schedules one of its neighboring stations for the next transmission on the channel using a distributed opportunistic algorithm. Token-DCF is a token passing MAC protocol which uses opportunistic overhearing to pass the token and to exchange scheduling information among

network stations.

In Token-DCF, when a station transmits on the channel, it might give a token (i.e., a privilege) to one of its neighbors. When a transmission ends, the privileged station, if there is any, starts transmitting after a short period of time, namely SIFS (short inter frame space). Non-privileged stations follow the backoff procedure of 802.11 DCF to access the channel. In this way, the privileged station does not go through the contention resolution phase and grabs the channel immediately. A distributed scheduling algorithm is used for choosing the privileged stations. Token-DCF is an opportunistic MAC protocol which behaves similar to 802.11 DCF when packets are not overheard by the neighboring stations. However, when the opportunistic overhearing is feasible, the backoff procedure of 802.11 DCF is eliminated to improve efficiency.

1.1.3 Concurrent-MAC: Increasing Concurrent Transmissions in Dense Wireless LANs

IEEE 802.11 is designed for WLANs in a "point-to-multipoint" scenario, in which each access point (AP) covers a number of client stations and each client station is associated with only one AP. Recently, large scale WLANs with high density of APs have emerged in many hotspots [8]. In dense wireless LANs, several APs are present in the transmission range of each client station and a client's packets are received by multiple APs within station's transmission range, due to the broadcast property of the wireless channel. As we explained before, wireless is a broadcast medium which creates the opportunity for multiple APs to receive packets of a station.

IEEE 802.11 defines a distributed and contention based Carrier Sense Multiple Access/Collision Avoidance (CSMA/CA) access control scheme. In the CSMA/CA mechanism of IEEE 802.11 DCF, a station willing to transmit senses the wireless channel to determine if the channel is busy or idle. If the channel is sensed busy, the station has to defer its transmission until the medium becomes idle. The main drawback of the carrier sensing mechanism is that, by simply comparing received power level with a carrier sense threshold, the information regarding which station is transmitting on the channel is lost. Some stations might be eligible for concurrent transmissions while

some might not, which is directly related to the SINR values at the receiver. Concurrent transmissions are transmissions that overlap in time. Maximizing the number of successful concurrent transmissions results in maximizing the aggregate throughput of the network.

In this part of the research, we investigate how wireless broadcast property can be used in dense wireless LANs in order to increase network throughput. The purpose of this work is to explore the dense architecture to improve the performance in dense WLANs. In this work, we design a MAC protocol, called Concurrent-MAC, which increases reliable concurrent transmissions in dense WLANs and achieves higher network throughput compared to IEEE 802.11 DCF.

The network model that we consider in this part of the research is a dense deployment of wireless stations and APs where the APs are connected by a fast backhaul to a central component, called the *controller*. Based on SINR values between stations and APs, sets of concurrent transmitters are identified by the backhaul of APs. A station gaining access to the channel schedules one of its neighbors for concurrent transmission. A neighbor chosen for concurrent transmission can start transmitting on the channel immediately after it overhears the privilege given to it for concurrent transmission. As we have found out in this research, in dense WLANs, there are many instances in which two nearby stations can transmit concurrently. Although the concurrent transmissions of the stations collide in some of the APs, it is captured successfully by other APs. Our simulation results show that, in some network topologies, Concurrent-MAC can improve aggregate throughput significantly, compared to 802.11 DCF.

1.2 Dissertation Outline

The dissertation is organized as follows:

- In Chapter 2, we present two distributed mutual exclusion algorithms designed for wireless networks. Simulation and implementation comparison with two existing mutual exclusion algorithms are presented.
- In Chapter 3, we present the design and evaluation of Token-DCF, a MAC protocol that decreases the idle time and collision time of IEEE

802.11 backoff mechanism.

- In Chapter 4, we design a MAC protocol for dense WLANs, which exploits the wireless broadcast property to increase concurrent transmissions and to improve network throughput.
- Finally, in Chapter 5 we conclude the dissertation.

CHAPTER 2

IMPROVING PERFORMANCE OF MUTUAL EXCLUSION IN WIRELESS NETWORKS

In this chapter, we design two mutual exclusion algorithms for wireless networks. Our mutual exclusion algorithms are distributed token based algorithms which exploit the opportunistic message overhearing in wireless networks. One of the algorithms is based on overhearing of token transmissions. In the other algorithm, overhearing of both token and request messages is exploited. The design goal is to decrease the number of transmitted messages and delay per critical section entry using the information obtained from overheard messages.

2.1 Introduction

A wireless ad hoc network is a network in which a pair of nodes communicates by sending messages over wireless links. The wireless channel is a shared medium and messages sent on the wireless links might be overheard by the neighboring nodes. The information obtained from the overheard messages can be used in order to design distributed algorithms, for wireless networks, with better performance metrics. Although existing distributed algorithms will run correctly on top of wireless ad hoc networks, our contention is that efficiency can be obtained by developing distributed algorithms, which are aware of the shared nature of the wireless channel. In this chapter, we present distributed mutual exclusion algorithms for wireless ad hoc networks.

Distributed processes often need to coordinate their activities. If a collection of processes share a resource or collection of resources, then often *mutual exclusion (MUTEX)* is required to prevent interference and ensure consistency when accessing the resources. In a *distributed* system, we require a solution to *distributed* mutual exclusion. Consider users who update a text file. A simple means of ensuring that their updates are consistent is to allow

them to access the file only one at a time, by requiring the editor to lock the file before updates can be made. A particularly interesting example is where there is no server, and a collection of peer processes must coordinate their access to shared resources amongst themselves.

Mutual exclusion (often called MUTEX) is a well known problem in distributed systems in which a group of processes require entry into the *critical section (CS)* exclusively, in order to perform some critical operations, such as accessing shared variables in a common store or accessing shared hardware. Mutual exclusion in distributed systems is a fundamental property required to synchronize access to shared resources in order to maintain consistency and integrity. To achieve mutual exclusion, concurrent access to the CS must be synchronized such that at any time only one process can access the CS. The proposed solutions for distributed mutual exclusion are categorized into two classes: token based [1], [2], [6] and permission based [3], [4], [5]. In token based MUTEX algorithms, a unique token is shared among the processors. A processor is allowed to enter the CS only if it holds the token. In a permission based MUTEX algorithm, the processor that requires entry into the CS must first obtain the permissions from a set of processors.

In this chapter, we design mutual exclusion algorithms for wireless networks. Wireless networks have fundamentally different characteristics mainly because the wireless channel is a shared medium and messages sent on the wireless links might be overheard by the neighboring nodes. The information obtained from the overheard messages can be used in order to design distributed algorithms, for wireless networks, with better performance metrics.

Most of the existing MUTEX algorithms are designed for typical wired networks [1],[2],[3],[4],[5]. Design of mutual exclusion algorithms for mobile ad hoc networks had received some interest in the past few years [6], [7]. Although the underlying network in these algorithms is wireless, the proposed algorithms are only mobility aware solutions, where the goal is to deal with the problems caused by node mobility, such as link failures and link formations. In this work, we show that the broadcast property of the wireless medium can be exploited in order to improve the performance of the MUTEX algorithms in wireless networks. To the best of our knowledge, this work is the first in which opportunistic overhearing is exploited to improve the performance of MUTEX in wireless networks.

In this chapter, we present two token based mutual exclusion algorithms that are designed for wireless networks. Network nodes communicate by transmitting unicast messages. Since the channel is wireless, a unicast message transmitted from node i to node j might be overheard by neighbors of node i, for example node k. In this case, node k is not the intended receiver of the message, but it has overheard the message due to the shared nature of the wireless medium. We design our algorithms such that the neighboring nodes that overhear messages can learn more recent information about the current status of the algorithm.

We call our algorithms *Token Overhearing Algorithm (TOA)* and *Token and Request Overhearing Algorithm (TROA)*. *TOA* is based on the MUTEX algorithm designed by Raymond [1]. In Raymond's algorithm, messages are transmitted over a static spanning tree of the network. *TOA* is based on overhearing of the token transmission and the spanning tree maintained by the algorithm changes when token transmission is overheard by the neighboring nodes. *TROA* is based on Trehel-Naimi's algorithm [2]. In Trehel-Naimi's algorithm, when a node requires entry to the CS, the node sends a request message to the last known owner of the token. In *TROA*, overhearing of both request and token messages are exploited in order to obtain recent information about the latest token holder in the network.

Performance of a distributed mutual exclusion algorithm depends on whether the system is lightly or heavily loaded. If no other processor is in the CS when a processor makes a request to enter the CS, the system is lightly loaded. Otherwise, when there is a high demand for the critical section which results in queueing up of the requests, the system is said to be heavily loaded. The performance metrics that we aim to improve in this work are the number of transmitted messages and delay per CS entry. We define the delay as the interval between the request of a node to enter critical section and the time it finishes executing the critical section.

Our mutual exclusion algorithms satisfy three correctness properties:

1. *Mutual Exclusion*: at most one processor is in the CS at any time.

2. *Deadlock free*: if any processor is waiting for the CS, then in a finite time some processor enters the CS.

3. *Starvation free*: if a processor is waiting for the CS, then in a finite

time the processor enters the CS.

The remainder of this chapter is organized as follows: We first review the existing mutual exclusion algorithms in Section 2.2. We then describe the network model in Section 2.3. In Section 2.4, we present *Token Overhearing Algorithm (TOA). Token and Request Overhearing Algorithm (TROA)* is described in Section 2.5. Simulation results are presented in Section 2.7.

2.2 Related Work

The distributed mutual exclusion algorithms are mainly classified in two categories: *Permission based* and *Token based*. Permission based mutual exclusion algorithms require that a requesting node should receive permissions from other nodes, either a set of nodes or all other nodes. In the token based algorithms, a unique token is shared among the set of the nodes. The node holding the token is eligible for entering the critical section. Several solutions have been proposed in the literature for fixed networks. Recently, some mutual exclusion algorithms have been proposed for mobile ad hoc networks.

In the token-based mutual exclusion algorithms, two techniques are usually used, namely circulating token and requesting token. In the requesting token method, a node requesting the CS has to obtain the token to be able to enter the CS. In some mutual exclusion algorithms, the request is sent to all the nodes, because the token holder is unknown [9]. In other MUTEX algorithms, a logical structure, for example a tree [1], is defined to point the token holder. The request is sent over the edges of the tree which leads to the token holder. In the MUTEX algorithm proposed by Lann [10], the logical structure is an unidirectional ring and the token circulates on this. When the token is received by a node, it enters the critical section, if it is requesting it. After executing the critical section, the node sends the token to its successor in the ring.

In [9], Ricart and Agrawala proposed an algorithm which requires transmitting at most N messages to achieve mutual exclusion. The requesting node sends the request message to all the other $N-1$ nodes and waits for a response. The token holder sends the token to the requesting node, whenever it exits the token. Based on the Ricart-Agrawala's algorithm [9], Suzuki and Kazami [11] designed a mutual exclusion algorithm in which the queue of

requesting nodes is piggybacked within the token. The queue is updated by the local queue of each visited node in an ascending node number in order to ensure the deadlock free property. Singhal [5] has improved the performance of the Suzuki and Kazami algorithm [11], to at most N messages in heavy loads. Their proposed algorithm uses a heuristic algorithm to guess what nodes of the system are probably the current or future token holder, and sends a request message only to these nodes rather than to all the nodes.

In [1], Raymond proposed an algorithm, in which a logical tree on the network rooted by the token holder is maintained. Raymond's algorithm requires $O(\log N)$ messages to enter the CS. The tree is maintained by the logical pointers of network nodes. The root of the tree is always the node holding the token. When a node wants to access its CS, it enqueues its identity and sends a request message to its parent. This message is then routed successively by the other nodes on the path between the token holder and the root of the tree. The token is transmitted on the reverse path to the requesting node. In [12], Chang, Singhal and Liu designed an algorithm which improves Raymond's algorithm which tolerates link failures and node failures by maintaining multiple paths to find the token. Their algorithm also tries to avoid cycles when the token returns to the requesting node along the revers links. In [13], Dhamdhene and Kulkarni designed an algorithm which aims to resolve the problem of still remaining cycle in the algorithm of [12]. Their algorithm is k-resilient, meaning that it can tolerate k node/link failures.

In permission-based mutual exclusion algorithms, a node requesting entry to the critical section, must receive permission from all the other nodes in the network, or from a set of nodes. In the algorithm proposed by Lamport [14], when a node wants to enter the critical section, it sends a request to all other nodes and waits for reply messages. When the node exits the CS, it sends a release message to all other nodes. This algorithm requires $3(N-1)$ messages per critical section entry. In [9], Ricalrt and Agrawala proposed a MUTEX algorithm which requires $2(N-1)$ messages per critical section entry. When a node wants to enter the critical section, it sends a request to all the nodes of the network and waits for responses. If it receives a response from all these nodes, it enters the critical section. Requests are ordered by Lamport clocks [14]. Maekawa [15] has designed an algorithm which uses a logical structure defined by a set of nodes associated with each node. This

set has a nonempty intersection with sets associated to other nodes. This structure allows each node requesting CS entry to acquire permission only from members of its associated set.

A more detailed overview of mutual exclusion algorithms can be found in [16].

2.3 Network Model

We consider a network of n nodes, communicating by message passing in a wireless ad hoc network. Each node has a unique identifier, i, $0 \leq i \leq n-1$. Messages transmitted in the network are unicast messages. We assume that lower layers of the network, such as MAC layer and transport layer, ensure reliable delivery of unicast messages. To ensure reliability, a retransmission mechanism is used in lower layers in case packets are lost due to noise or interference. Since the network is wireless, a unicast message from node i to node j might be overheard by neighbors of node i, such as node k. We assume that if such an opportunistic overhearing happens, node k does not discard the overheard message; instead it uses the information included in the message. We do not assume that the unicast message of node i to node j is delivered reliably to the neighbors of node i. Instead, the overhearing is opportunistic, meaning that if the neighboring nodes overhear messages not intended for them, they exploit the information included in the messages. We assume that network nodes do not fail and each node is aware of the set of nodes with which it can directly communicate.

2.4 Token Overhearing Algorithm (TOA)

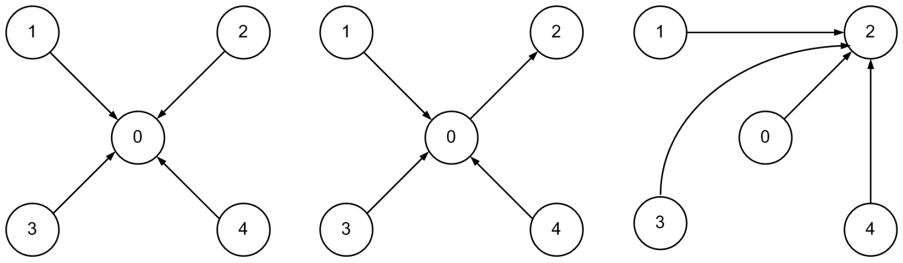
Token Overhearing Algorithm (TOA) is based on Raymond's algorithm [1]. Raymond designed a distributed token based mutual exclusion algorithm in which requests are sent over a static spanning tree of the network, towards the token holder. The tree is maintained by logical pointers distributed over the nodes and directed to the node holding the token. At each time instance, there is a single directed path from each node to the token holder. When a node has a request for the token, a sequence of request messages

are sent on the path between the requesting node and the token holder. The token is sent back over the reverse path to the requesting node. The direction of the links over which the token is transmitted is reversed. In this way, at each time instance, all edges of the tree point towards the token holder. Since Raymond's algorithm uses a spanning tree of the network, the number of messages exchanged per CS depends on the topology of this tree. However, in a randomly constructed tree of n nodes, the number of messages exchanged is $O(\log n)$ under light demand and approximately three messages under saturated demand [1].

Similar to Raymond's algorithm, *Token Overhearing Algorithm* uses a spanning tree of the network over which messages are passed. But unlike Raymond's algorithm, the spanning tree in *TOA* is dynamic and changes if token transmission is overheard by the neighboring nodes. Sender and receiver of the token are specified in the token message. When a token is sent from node i to node j, any other node k that overhears transmission of the token, changes its parent in the tree. We assume that each node maintains a list of its neighboring nodes. If the destination of the token, j, is a neighbor of k, node k chooses node j as its parent in tree. Otherwise, since the token message is overheard by k, this means that i is an immediate neighbor of k, and as a result, k chooses i as its parent in the tree. Unlike Raymond's algorithm, the tree in *TOA* is not necessarily fixed and might change when the token transmission is overheard by the neighboring nodes.

2.4.1 Example of Algorithm Operation

An example execution of Raymond's algorithm and *TOA* is illustrated in Figure 2.1. The network is a wireless ad hoc network composed of five nodes, node 0 through node 4. We assume that the network is single-hop, in which all nodes are in the communication range of each other. Figure 2.1(a) shows the initial spanning tree of the network. In this figure, the token holder is node 0 and all edged point towards this node. We consider a case where node 2 requires entry to the CS and sends a request message to node 0. We assume that there is no other pending request in the network. When node 0 receives the request of node 2, it sends the token to node 2. Figure 2.1(b) shows the spanning tree in Raymond's algorithm after the token is sent from



(a) Initial tree (b) Raymond's algorithm (c) TOA

Figure 2.1: Example execution of Raymond's algorithm and *TOA*

node 0 to node 2. At this point, the direction of the edge between node 0 and node 2 is reversed. In Figure 2.1(b), nodes 1, 3 and 4 are two hops away from node 2. If any of these nodes, for example node 1, requests to enter the CS, two request messages are sent, one request message from node 1 to node 0 and one from node 0 to node 2. It then takes two messages to send the token from node 2 to node 1. So, total of four messages are sent so that node 1 can enter the CS.

We now describe how *TOA* performs when the nodes are initially configured as depicted in Figure 2.1(a) and node 2 requires CS entry. Like Raymond's algorithm, when node 0 receives the request of node 2, it sends the token to node 2. Since all nodes are in the communication range of each other, token transmission from node 0 to node 2 might be overheard by nodes 1, 3 and 4. We consider the best scenario for our algorithm, in which all nodes 1, 3 and 4 overhear the token transmission from 0 to 2. As a result, nodes 1, 3 and 4 point to node 2. Figure 2.1(c) shows the spanning tree in our algorithm when the token is sent to node 2 and overheard by nodes 1, 3 and 4. In this figure, nodes 1, 3 and 4 are only one hop away from the token holder, node 2. If any of these nodes requests entry to the CS, only two messages are transmitted, one request message and one token message. This example shows that in single-hop wireless networks and when requests for the token are initiated separate enough in time, *TOA* might perform better than Raymond's algorithm. The reason is that as a result of messages being overheard, network nodes might be aware of the current token holder in which case they send their requests directly to the token holder. In single hop networks, if every node overhears token transmission, *TOA* is optimal and only two messages, one request message and one token message,

are transmitted per CS entry. On the other hand, in Raymond's algorithm four messages might be sent for one CS entry, in single-hop networks. The reason is that, although the token holder and the requesting node are in the communication range of each other, they might not communicate directly; rather, they exchange messages through the initial root (e.g. node 0 in this example) simply because Raymond's algorithm uses a static spanning tree.

2.4.2 Data Structures and Messages

In this section, we present the data structures and message types of *Token Overhearing Algorithm*. Each node of the network maintains the following data structures.

- ***parent*** denotes the parent of node i in the tree. If node i is the root of the tree, ***parent*** $= i$.
- ***using*** is equal to `true` if the node is in the CS and `false` otherwise.
- ***requestQ*** is a FIFO queue that contains the id of the nodes that have sent a pending request to node i.
- ***asked*** is `true` if node i has sent a request to ***parent*** and has not received the token yet. Otherwise, ***asked*** is `false`.

The following information is included in each message.

- ***sender***: id of the node that sends the message.
- ***receiver***: id of the receiver node.
- ***type***: three types of messages are transmitted in the network: `REQUEST`, `TOKEN` and `TOKENandREQUEST`.

2.4.3 Algorithm Procedures

There are five procedures in *Token Overhearing Algorithm*. **Enqueue**, **Dequeue**, **SendToken**, **MakeRequest** and **OverhearToken**. In the following, we describe these procedures.

Enqueue

When node i calls **Enqueue(*requestQ*,j)**, j is enqueued on *requestQ* of node i.

Dequeue

Dequeue(*requestQ*) removes the id at the head of *requestQ* and returns the removed id.

SendToken

In this procedure, if the token holder does not use the token and its *requestQ* is not empty, it sends the token to the node at the head of the *requestQ*. If token holder itself is at the head of the *requestQ*, it enters the CS. If the token needs to be sent to another node and the token holder does not have any more pending requests in its *requestQ*, message with *type* = `TOKEN` is sent to the only one node in *requestQ*. On the other hand, if the token holder has other requests in *requestQ*, then a message with *type* = `TOKENandREQUEST` is sent to the node at the head of *requestQ*. This means that the current token holder requires the token to be sent back, because there are other pending requests in its *requestQ*. Sending `TOKENandREQUEST` means that the token is sent and the sender requests the token as well. The receiver of `TOKENandREQUEST` will enqueue the sender of the message at the end of its *requestQ*. This mechanism is called the piggyback strategy [1]. Proc. 1 describes **SendToken** in which `myId` denotes the id of the node calling the procedure.

MakeRequest

In this procedure, if a node has a non-empty *requestQ*, and has not asked for the token, and does not hold the token, it sends a request to its parent in the tree.

1 SendToken

```
1: if (parent == myId) and (using == false) and (requestQ is not empty) then
2:     parent = Dequeue(requestQ)
3:     asked = false
4:     if (parent == myId) then
5:         using = true
6:         enter CS
7:     else
8:         if requestQ is not empty then
9:             send TOKENandREQUEST to parent
10:            asked = true
11:    else
12:        send TOKEN to parent
```

2 MakeRequest

```
1: if (parent ≠ myId) and (requestQ is not empty) and (asked == false) then
2:     send REQUEST to parent
3:     asked = true
```

OverhearToken

When a node k overhears the transmission of `TOKEN` or `TOKENandREQUEST` from node i to node j, **OverhearToken** is executed. In this case, k is not the intended receiver of the message, but it has overheard the message. $parent_k$ becomes j if k and j are immediate neighbors; otherwise k chooses i as its parent.

3 OverhearToken

```
1: if type is TOKEN or TOKENandREQUEST then
2:     if receiver is my neighbor then
3:         parent = receiver
4:     else
5:         parent = sender
```

2.4.4 Events

The designed algorithm is event-driven in which six events might happen.

I) The node i requires access to the CS. In this case, the following procedures are executed:

1. **Enqueue** (*requestQ*, i)
2. **SendToken**
3. **MakeRequest**

Node i itself might hold the token, in which case i enters the CS when **SendToken** is executed. Otherwise, **MakeRequest** sends a request to *parent*.

II) The node j receives a `REQUEST` from node i. The node executes these procedures:

1. **Enqueue** (*requestQ*, i)
2. **SendToken**
3. **MakeRequest**

If j is the token holder, **SendToken** might send the token to i. Otherwise, **MakeRequest** sends a request to *parent*.

III) The node k overhears the transmission of `TOKEN` or `TOKENandREQUEST` from node i to node j. The procedure **OverhearToken** is executed, after which k will point to either i or j.

IV) The node j receives the `TOKEN`:

1. *parent* $= j$
2. **SendToken**

SendToken might allow j itself to enter the CS, or it might send the `TOKEN` to another node. In case there are more pending requests in *requestQ*, `TOKENandREQUEST` is sent instead of `TOKEN`.

V) The node j receives `TOKENandREQUEST` from node i:

1. *parent* $= j$
2. **Enqueue** (*requestQ*, i)
3. **SendToken**

First, the request of i is enqueued in `requestQ`. Then, **SendToken** is executed in which node j might enter the CS or it might send `TOKEN` or `TOKENandREQUEST` to another node.

VI) The node i exits the CS:

1. *using* = `false`
2. **SendToken**.

When node i exits the CS, in case there is any pending request in its *requestQ*, the `TOKEN` is sent to the requesting node. If return of the token is needed, `TOKENandREQUEST` is sent instead of `TOKEN`.

2.5 Token and Request Overhearing Algorithm (TROA)

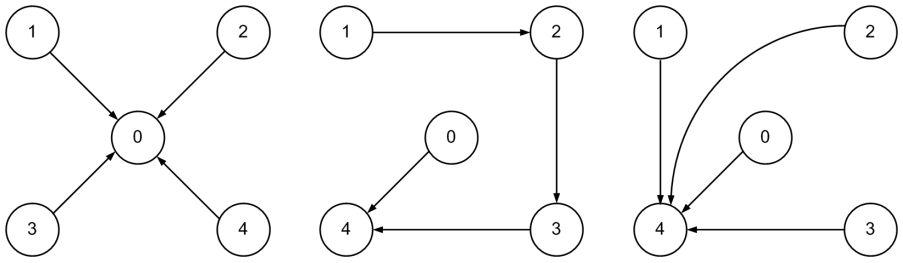
We design another mutual exclusion algorithm, called *Token and Request Overhearing Algorithm (TROA)* for wireless networks. *TROA* is based on Trehel-Naimi's algorithm [2]. The objective in designing *TROA* is to find a MUTEX algorithm in which overhearing of both token and request messages is exploited in order to improve the performance. We note that *TOA* is only based on overhearing of token transmission.

Trehel-Naimi's algorithm [2] is a token-based algorithm which maintains two data structures:

1. A dynamic tree structure in which the root of the tree is the last node that will hold the token among the current requesting nodes. This tree is called the *last* tree. Each node i has a local variable *last* which points to the last probable token holder that node i is aware of.

2. A distributed queue which maintains requests for the token that have not been answered yet. This queue is called the *next* queue. Each node i keeps the variable *next* which points to the next node to whom the token will be sent after i releases the critical section.

In Trehel-Naimi's algorithm, when a node i requires entry to the critical section, it sends a request to its *last* and then changes its *last* to `null`.



(a) Initial tree (b) Trehel-Naimi's algorithm (c) Our algorithm

Figure 2.2: Example execution of Trehel-Naimi's algorithm and *TROA*

As a result, i becomes the new root of the last tree. When node j receives the request of node i, one of the following cases happens:

1. j is not the root of the tree. It forwards the request to its last and changes its last to i.

2. j is the root of the tree. If j holds the token, but does not use it, it sends the token to i. If j is in the critical section or is waiting for the token, j sets its next to i. Whenever j exits the critical section, it sends the token to $\mathit{next} = i$.

Trehel-Naimi's algorithm is designed for wired networks in which transmitted messages are not overheard by the neighboring nodes. We modify the algorithm to perform better in wireless networks by exploiting the broadcast property of wireless networks. In *TROA*, nodes can learn more recent information about the last token holder in the network by overhearing of messages not intended for them.

2.5.1 Example of Algorithm Operation

An example execution of Trehel-Naimi's algorithm and *TROA* is illustrated in Figure 2.2. We assume all nodes are in the communication range of each other and every message is overheard by all the nodes. In the initial configuration, Figure 2.2(a), $\mathit{last}_i = 0$ for all $0 \leq i \leq 4$. We consider a case where nodes 1, 2, 3 and 4 initiate a request for the token, sequentially in this order and separate enough in time. Figure 2.2(b) shows the tree constructed by Trehel-Naimi's algorithm after all requests are received at their destinations. When node 1 initiates a request for the token, it sends its request to last_1 which

is node 0. When node 0 receives request of node 1, it sends the token to node 1 and changes $last_0$ to 1. Later, when node 2 sends a request to node 0 (since $last_2$ is 0), node 0 forwards this request to $last_0$, node 1. When request of node 2 is received by node 1, $last_1$ is changed to 2. Whenever node 1 exits the CS, it sends the token to node 2. The requests of nodes 3 and 4 are forwarded to nodes 2 and 3, respectively, by node 0. At this point, $last_0 = 4$ and $last_2 = 3$. Token will be sent from 2 to 3 and 3 to 4. The final configuration is shown in Figure 2.2(b), in which each node i points to $last_i$.

Considering Figure 2.2(b), we observe that nodes 0, 1, 2 and 3 are one, three, two and one hop away from the node holding the token, 4, respectively. In this configuration, if for example node 1 wants to enter the CS, three request messages are sent, from 1 to 2, from 2 to 3 and from 3 to 4. Token is directly sent from 4 to 1. So, the total of four messages are transmitted so that node 1 can enter the CS.

Figure 2.2(c) shows the constructed tree in our algorithm, *TROA*, after requests of nodes 1, 2, 3 and 4, are received at their destinations. We note that, in our algorithm, $last$ of a node changes if the node overhears a more recent request or token message. Since all nodes are in the communication range of each other, the last request message, which is the request initiated by node 4, is overheard by nodes 1, 2 and 3 and as a result all of them change their $last$ to 4. The reason is that $last_i$ is equal to the initiator of the latest request or token message that indicates the highest number of CS entry in the network. Nodes 0, 1 and 2 overhear the last event in this example, transmission of token from node 3 to node 4 and change their $last$ to 4. $last_3$ changes to 4 as a result of the regular reception of the request from 4 by 0. We observe that, in Figure 2.2(c), all nodes are one hop away from the token holder, node 4. In this scenario, only two messages are transmitted (one request message and one token message), if any of the nodes 0-3 wants to enter the CS.

2.5.2 Data Structures

In *TROA*, each node maintains the following data structures:

- $privilege$ is `true` if the node holds the token, and `false` otherwise.

- *requestingCS*: When a node initiates request for the token, it sets its *requestingCS* to `true`. *requestingCS* becomes `false` when the node releases the CS.

- *last*: When a node wants to enter the critical section, it sends a request to its *last*. *last* of a node might change when the node receives or overhears messages.

- *next*: When a node that is waiting for the token receives a request message from another node, it saves the *initiator* of the request message in its *next*. Later, when the node releases the critical section, it sends the token to *next*.

- *numCSEntry* of node i denotes how many times critical section entry has happened in the network such that node i is aware of.

- *numReceivedRequests* denotes how many request messages are received by a node while the node was waiting for the token.

numCSEntry and *numReceivedRequests* are used as counters to determine if a node should change its *last* when it overhears messages. We will present more details later, when we describe algorithm procedures.

2.5.3 Messages

There are two types of messages in the algorithm, `REQUEST` and `TOKEN`. A `REQUEST` message includes the following information.

- *initiator* is the id of the node that has initiated the request for the token.

- *destination* denotes the final destination of the message. Since we consider the general case of multi-hop networks, *destination* is not necessarily a neighbor of *initiator*. In this case, the message is routed on the shortest path between *initiator* and *destination*.

- *numberCSEntry*: When a node transmits a message, it writes its *numCSEntry* in *numberCSEntry* part of the message. We note that *numCSEntry* of node i denotes how many times critical section entry

has happened in the network such that node i is aware of. *numberCSEntry* is used by nodes that overhear the message to determine if their *last* should be changed or not.

A message of type TOKEN includes the following information:

- *destination* denotes the final destination of the token.
- *numberCSEntry*: As we explained before, when a node transmits a message, it writes its *numCSEntry* in *numberCSEntry* part of the message, to indicate that how many CS entries, that node i is aware of, have happened in the network.

2.5.4 Algorithm Procedures

In this section, we present the procedures of *Token and Request Overhearing Algorithm (TROA)*.

Initialization

Procedure 4 is executed at the beginning of the algorithm by every node i to set the initial value of i's data structures.

4 Initialization

```
1: last = INITIAL-TOKEN-HOLDER
2: next = null
3: requestingCS = false
4: numReceivedRequests = 0
5: numCSEntry = 0
6: if last == myId then
7:     privilege = true
8:     last = null
9:     numCSEntry = 1
10: else
11:    privilege = false
```

RequestCS

Procedure 5 is called when a node wants to enter the critical section. If the node holds the token, it enters the critical section. Otherwise, it sends a

REQUEST to its *last*.

5 RequestCS

1: *requestingCS* = true
2: **if** (*privilege* == false) **then**
3: send REQUEST to *last*
4: *last* = null
5: **else**
6: enter CS

OverhearRequest

When a REQUEST message from node i to node j is overheard by node k, Procedure 6 is executed, in which if some conditions hold, node k changes its *last* to *initiator* of the message.

6 OverhearRequest

1: **if** *numberCSEntry* > *numCSEntry*+*numReceivedRequests*+1 and *last* != null and *requestingCS* == false **then**
2: *last* = *initiator*
3: *numCSEntry* = *numberCSEntry*−1
4: *numReceivedRequests* = 0

OverhearToken

When node k overhears the transmission of TOKEN from node i to node j, Procedure 7 is executed.

7 OverhearToken

1: **if** *numberCSEntry* > *numCSEntry* + *numReceivedRequests* and *last* != null and *requestingCS* == false **then**
2: *last* = *destination*
3: *numCSEntry* = *numberCSEntry*
4: *numReceivedRequests* = 0

ReceiveToken

Procedure 8 is executed when `TOKEN` is received at its final destination, *destination*. Intermediate nodes on the path that forward the message do not run this procedure.

8 ReceiveToken

```
1: privilege = true
2: numCSEntry = numberCSEntry +1
3: numReceivedRequests = 0
4: enter CS
```

ReleaseCS

Procedure 9 is executed when a node exits the critical section.

9 ReleaseCS

```
1: requestingCS = false
2: if next != null then
3:     privilege = false
4:     send TOKEN to next
5:     next = null
```

ReceiveRequest

Procedure 10 is executed when a `REQUEST` message is received at its final destination, *destination*.

10 ReceiveRequest

```
1: if last==null then
2:     if requestingCS==true then
3:         next=initiator
4:     else
5:         privilege=false
6:         send TOKEN to initiator
7: else
8:     send request to last
9:     numReceivedRequests++
10: last=initiator
```

2.6 Handling Message Loss and Link Failure

Wireless channel is an unreliable environment and messages sent on wireless links might be lost. Furthermore, wireless networks may suffer link failures. Link failure happens either when wireless nodes are mobile or when the link quality decreases to a level that makes reliable communication impossible. If a link fails, the communication between the two end points is interrupted until the link failure is detected and an alternate route is chosen. The characteristics of the wireless medium, such as message loss and connectivity loss, raise additional issues in token based mutual exclusion algorithms of wireless networks. In order to increase the robustness of our algorithms, *TOA* and *TROA*, we add mechanisms to handle message loss and link failure.

2.6.1 Application-level Acknowledgements and Packet Retransmissions

Due to unreliable wireless channel, request or token messages can be lost in wireless networks. We add a mechanism to *TOA* and *TROA*, to detect message loss and retransmit lost messages. Since message transmission at the wireless MAC layer is not reliable, an acknowledgment method with retransmission is used to guarantee reliability of message transmissions. This method requires the receiver to respond with an acknowledgment message to each `TOKEN` or `REQUEST` message it receives. The sender keeps track of each message it sends. The sender also keeps a timer from when the message was sent, and retransmits the message when the timer expires before the message has been acknowledged. The timer is needed in case a message gets lost. We use a sequence number to identify each message. For every message transmitted, the sequence number is incremented by one. We use selective acknowledgments to provide information about which packets are received successfully. When retransmission timer expires before the message has been acknowledged, the sender infers that the message has been lost, and it retransmits the lost message.

2.6.2 Handling Link Failure

The link failure on link (i, j) is detected when node i sends a message to node j `numRetry` times, but it does not receive the application-level acknowledgement before the retransmission timer expires, for none of the transmitted messages. When some link failure is detected during transmission of `REQUEST` or `TOKEN` messages, we send the message through another route. In this scenario, we let the routing protocol to find a route between nodes i and j. From this moment onward, messages between nodes i and j are transmitted via the alternative route, instead of the failed direct link between nodes i and j.

2.7 Simulations

We run simulations to measure the performance of *TOA* and *TROA*. We also simulate Raymond's algorithm and Trehel-Naimi's algorithm to find the improvements obtained by message overhearing. In our simulations, network nodes are placed uniformly at random in a square area. The node closest to the center of the area is chosen as the initial root of the tree. Breadth-first search (BFS) algorithm is run to construct the initial tree in *TOA* and Raymond's algorithm. Dijkstra's algorithm is run to find the shortest path between nodes in *TROA* and Trehel-Naimi's algorithm. Messages sent in the network are unicast messages. In order to implement message overhearing, we change the 802.11 MAC layer of ns-2. In the current implementation of ns-2, packets that are received in the MAC layer of node i with MAC destination address different from i's MAC address are dropped. We change ns-2 so that such packets are not dropped, and they are delivered to the application layer of node i. For the simulations in which message overhearing is not exploited, messages with destination address different from the receiving node address are dropped at the MAC layer and are not delivered to the upper layers.

In this work we measure two performance metrics, which we call the cost of the algorithms:

1. Number of messages per CS entry: it is equal to the number of messages transmitted in the network per entry to the CS.

2. Delay per CS entry: the delay is measured as the interval between the

time at which a node initiates a request to enter the CS and the time at which the node enters the CS.

Requests for CS entry are assumed to arrive at a node according to a Poisson distribution with rate λ requests/second. When λ is small, no other processor is in the CS when a processor makes a request to enter the CS. In this case, the network is said to be lightly loaded. When λ is large, there is a high demand for entering the CS which results in queueing up of the requests, and the network is said to be heavily loaded. The time to execute the CS is 10^{-5} second.

Figures 2.3-2.5 plot number of messages and delay per entry to the CS against λ in three example networks. λ increases from 10^{-3} to 10^{2} requests / second. Each point in Figures 2.3-2.5 is obtained by taking the average of 10 runs of the algorithms. In each run, the total number of entries to the CS is $5 * n$, where n is number of nodes in the network. In other words, each point in Figures 2.3-2.5 corresponds to the average cost of $50 * n$ entry to the CS.

2.7.1 Single Hop Network

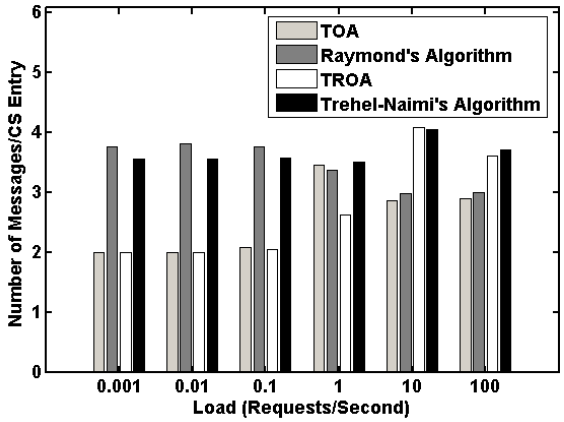
Figure 2.3 plots the cost of the algorithms against λ, in a single-hop network. The network is composed of $n = 20$ nodes placed uniformly at random in an area of $100m \times 100m$. The transmission range is $250m$. In such a scenario, each node is an immediate neighbor of every other node. We observe that in Figure 2.3(a), *TOA* outperforms Raymond's algorithm, when λ is small (i.e., under light demand for the token). In single-hop networks and for small λ, approximately 4 messages are transmitted per CS entry in Raymond's algorithm while 2 messages per CS entry are transmitted in *TOA* (as explained in Section 2.4.1). Figure 2.3(b) shows that the delay per CS entry is smaller in *TOA* than in Raymond's algorithm, under light demand for the token. Under light demand for the token, when node i makes a request to enter the CS, no other message is transmitted in the network except the messages correspond to the request of node i. In this case, the delay per CS entry is equal to the time required to transmit request and token messages between the requesting node and the token holder, and the wireless channel is available whenever a node wants to transmit a message; i.e. there is no contention in the network.

Raymond's algorithm is designed such that, under heavy demand for the token, a constant number of messages (approximately 3) are transmitted per CS entry. Detailed explanation can be found in [1]. In Figure 2.3(a) we observe what we expected, meaning that under heavy demand approximately 3 messages are transmitted in both Raymond's algorithm and *TOA*. As Figure 2.3(b) shows, both *TOA* and Raymond's algorithm have the same delay when λ is large, simply because both algorithms transmit the same number of messages per CS entry. Delay increases as λ increases, because at each time instant, there is more than one node requiring access to the channel and so packet of a node might be delayed by other nodes that are using the channel.

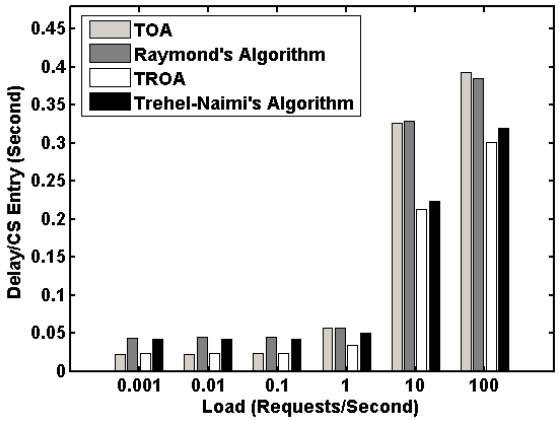
As Figure 2.3 shows, the cost of *TOA* is half of the cost of Raymond's algorithm under light demand. Under heavy demand both algorithms perform approximately the same. We conclude that the cost is decreased by opportunistic overhearing when demand for the token is light.

Figure 2.3 also plots the cost of *TROA* and Trehel-Naimi's algorithm. As Figure 2.3(a) shows, in *TROA* two messages are transmitted per CS entry under light demand. The reason is that in *TROA*, nodes send their request to the last token holder, which is known to them because of message overhearing. So, only two messages, one request message and one token message, are transmitted per every CS entry. On the other hand, in Trehel-Naimi's algorithm, a requesting node does not necessarily know which node is the last token holder, since a node does not receive messages exchanged between other nodes. In such a case, a sequence of request messages are transmitted until the request of the requesting node is received by the token holder. We conclude that under light demand, cost of *TROA* is less than the cost of Trehel-Naimi's Algorithm.

Figure 2.3(a) shows that when demand for the token increases, the number of messages transmitted in *TROA* increases. The reason is that requests for the token from different nodes are initiated close to each other, and so a node might not know the latest status of the algorithm when it initiates a request. For example, we consider a case where node i sends a request to the token holder, node j. If another node k initiates a request before it overhears the request of node i, node k sends its request to node j which is not the last requesting node any more. Node j will forward the request of node k to node i and so one extra request message is transmitted. Figure 2.3(b)



(a) Number of Messages

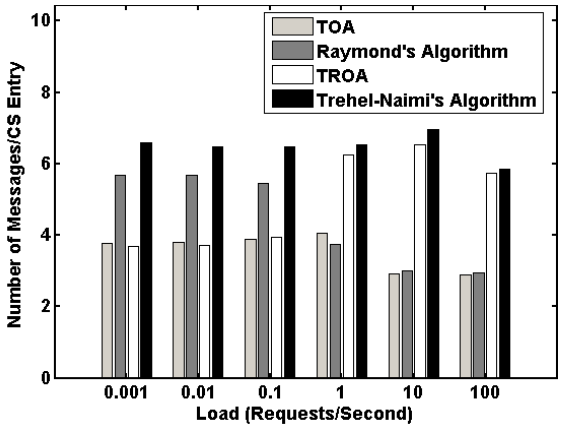


(b) Delay

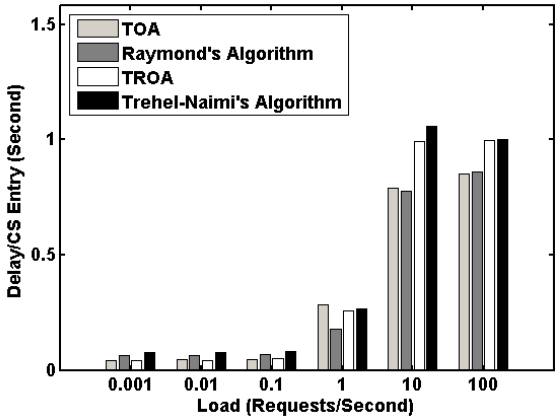
Figure 2.3: 20 nodes placed in 100mx100m

shows that delay per CS entry in *TROA* is always less than Trehel-Naimi's algorithm, when λ is small, simply because fewer messages are transmitted in *TROA*. When λ increases, delay of both *TROA* and Trehel-Naimi's algorithm increases, as a result of contention between nodes on accessing the wireless channel. Considering Figure 2.3, we conclude that in single hop networks, opportunistic overhearing improves the performance the most when demand for the token is light, and the improvement is about 100%.

Figures 2.4 and 2.5 plot the cost of the algorithms in multi-hop networks. Figure 2.4 plots the cost in a network of 40 nodes placed randomly in an area of $500m \times 500m$. The underlying network in Figure 2.5 is 60 nodes placed randomly in an area of $800m \times 800m$. Figure 2.4 shows that in this network topology, under light demand for the token (small λ), the cost of *TOA* is less than the cost of Raymond's algorithm. Comparing *TOA* and Raymond's



(a) Number of Messages

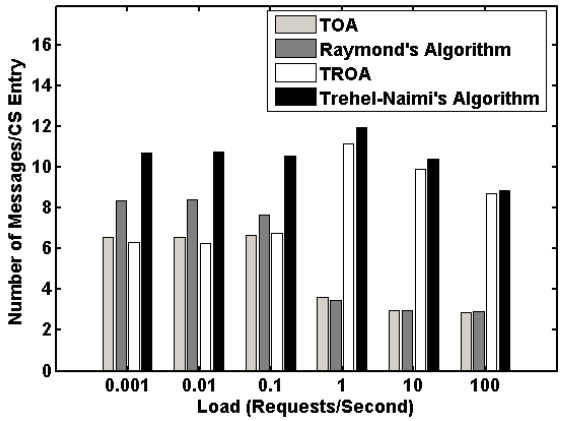


(b) Delay

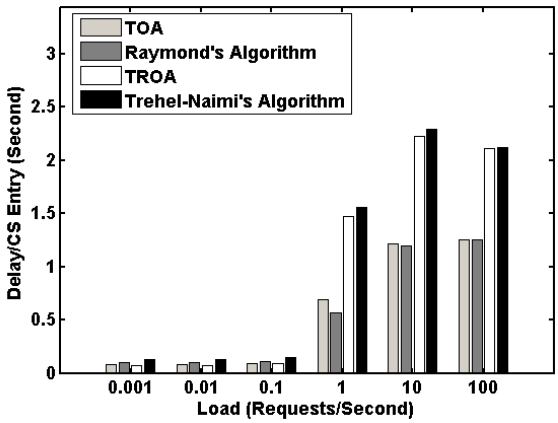
Figure 2.4: 40 nodes placed in 500mx500m

algorithm in Figure 2.3 and Figure 2.4, we observe that the improvement obtained by message overhearing has decreased in Figure 2.4. The reason is that in a multi-hop network, nodes do not overhear all transmitted messages in the network and so they are not able to learn the latest status of the algorithm; i.e., they might not know which node currently holds the token. As Figure 2.4 shows and as we explained before, under heavy demand (large λ), Raymond's algorithm and *TOA* have almost the same cost. Figure 2.4(b) shows that the delay of Raymond's algorithm and *TOA* increases when λ increases. This is because of the contention between nodes in accessing the wireless channel.

As we observe in Figure 2.5, the cost of *TOA* is still less than the cost of Raymond's algorithm, although improvement percentage has decreased. This shows that in Raymond's algorithm, as the size of the network increases,



(a) Number of Messages



(b) Delay

Figure 2.5: 60 nodes placed in 800mx800m

the improvement percentage obtained by exploiting message overhearing decreases. As Figures 2.4 and 2.5 show, in these networks when λ is small, *TROA* outperforms Trehel-Naimi's algorithm and the improvement is still significant. We conclude that the effect of exploiting message overhearing in different MUTEX algorithms is not always the same; instead it depends strongly on the design of the algorithm.

2.8 Implementation

We have implemented Raymond's algorithm and *Token Overhearing Algorithm (TOA)* in a testbed to evaluate the performance of these two mutual exclusion algorithms in an indoor office environment. The implementation

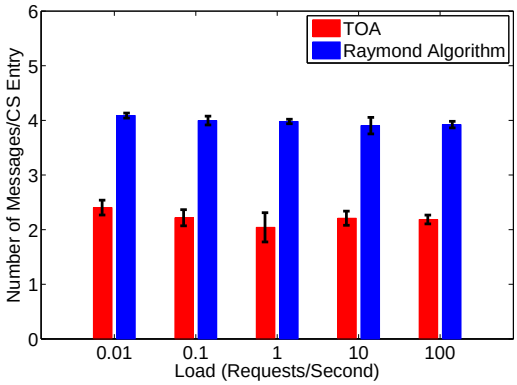


Figure 2.6: 6 laptops placed in room 459 CSL

consists of 6 laptops placed in Coordinated Science Lab (CSL) at UIUC. Each laptop is equipped with a TP-LINK TL-WN721N Wireless USB Adapter. These wireless cards are based on Atheros chipsets and are driven by ath9k drivers. In the implementation used for our experiments, transmitted messages are broadcast messages, where no MAC layer acknowledgement is transmitted. Since 802.11 MAC layer broadcasting does not guarantee reliable message delivery, we incorporate transmission of application layer acknowledgements and packet retransmissions in the implemented mutual exclusion algorithms, to ensure reliable message delivery. The retransmission timer is set to 20 seconds. We run the measurements for both single hop and multihop settings. For each setting, we repeat the measurements 5 times and the results are averaged over the 5 different runs. In each run, and for each node, the total number of entries to the critical section is 10. Similar to our ns-2 simulations, requests for critical section entry are assumed to arrive at a node according to a Poisson distribution with rate λ requests/second. We vary λ from 10^{-2} to 10^{2} requests/second.

The first set of experiments are conducted to demonstrate the performance of *TOA* and Raymond's algorithm in a single hop setting. In this experiment, we place 6 laptops next to each other on a table in room 459 CSL. All laptops are placed in the transmission range of each other and can send packets to each other and receive packets from each other. One of the laptops is chosen as the root node and does not require entry to the critical section. In each run, we measure the total number of transmitted messages in the algorithms. The results for this setting are plotted in Figure 2.6.

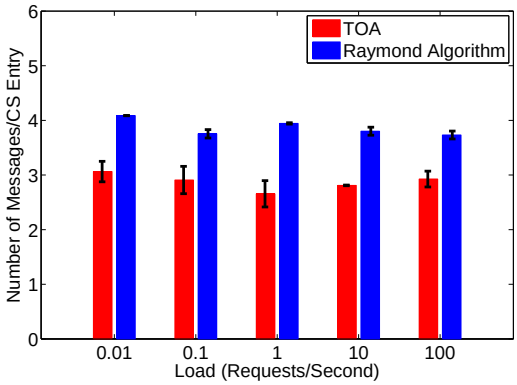


Figure 2.7: 3 laptops placed in room 459 CSL and 3 laptops placed in the stairway

From Figure 2.6, we observe that the average number of transmitted messages per CS entry in Raymond's algorithm is between 3.63 and 4.16, while in *TOA*, average number of transmitted messages per CS entry is between 1.61 and 2.58. This shows that number of transmitted messages per CS entry in *TOA* is approximately half of the number of transmitted messages per CS entry in Raymond's algorithm in an indoor office environment. As explained in Section 2.4.1, in *Token Overhearing Algorithm (TOA)*, and in a single-hop network, due to message overhearing, network nodes are aware of the current token holder and they transmit their request for token to the node holding the token, which results in one request message and one token message being transmitted to enter critical section once. This means that 2 messages per CS entry are transmitted in *TOA* algorithm. We note that more than 2 messages might be transmitted in the network, due to message loss and message retransmissions. On the other hand, in Raymond's algorithm, 4 messages per CS entry are transmitted in a single-hop network, since nodes do not communicate directly; instead they communicate via the root node, which results in transmitting 2 request messages and 2 token messages for 1 CS entry, for most cases (except when the root has the token).

In our second experiment, we place 3 laptops in room 459 CSL and 3 laptops in the stairway in the same floor. In this way, the two sets of laptops are separated by two walls. The communication between a laptop in the room 459 CSL and a laptop in the stairway might be lossy and some of the messages exchanged between these two laptops might be lost. The number

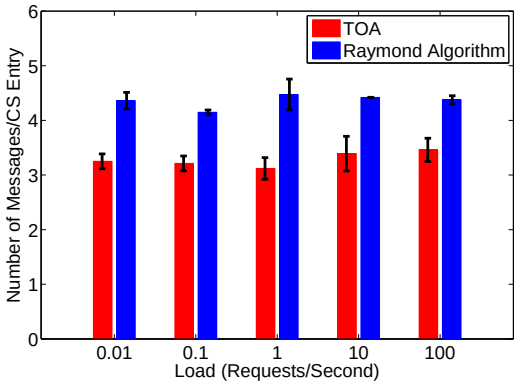


Figure 2.8: 3 laptops placed in the first floor and 3 laptops placed in the second floor

of transmitted messages per CS entry for *TOA* and Raymond's algorithm are shown in Figure 2.7. As shown in this figure, the number of transmitted messages per CS entry for both *TOA* and Raymond's algorithm has increased, compared to Figure 2.6. This is caused by the lossy channel and message retransmissions. We note that *TOA* still outperforms Raymond's algorithm significantly, due to message overhearing. Although some messages might not be overheard by some laptops, still a significant percentage of packets transmitted in the network are overheard by the laptops.

In our third experiment, we place 3 laptops in room 108 CSL and 3 laptops in room 206 CSL. In this setting laptops are separated by one floor, and communication between laptops in the first floor and the second floor is too lossy, meaning that a lot of packet retransmissions happen to deliver one packet from one floor to the other. The communication between laptops in the same room does not experience high loss, due to high channel gain between laptops in the same room. The numbers of transmitted messages per CS entry for *TOA* and Raymond's algorithm are shown in Figure 2.8. As we observe in this figure, the improvement obtained from overhearing is decreased, significantly, compared to Figure 2.6. The reason is that, since laptops are separated by a very lossy channel, i.e. one floor, network nodes do not overhear all messages transmitted in the network. Network nodes only hear parts of the packets transmitted in the network, and therefore their information about the latest token holder might be stale. For example, we consider a scenario in which node i hears the token being transmitted from

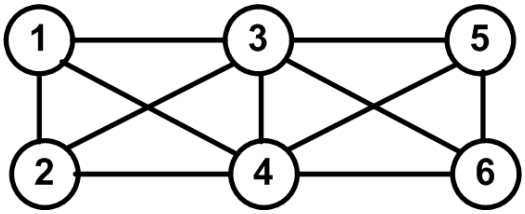


Figure 2.9: Topology of our two hop experiment

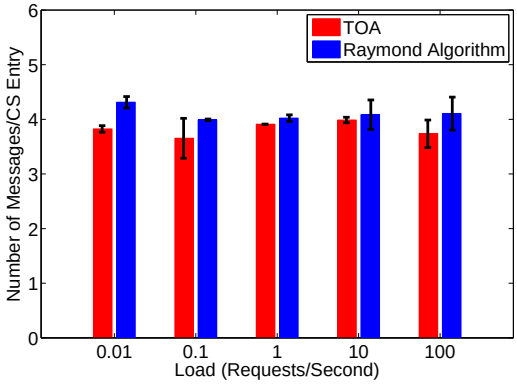


Figure 2.10: 6 laptops placed in a 2 hop network in the fourth floor of CSL

node j to node k. At this point, node i chooses node k as its parent. We then assume that node k sends the token to another node, node l, but node i does not overhear this token transmission. At this point, if node i requires CS entry, it sends a request message to node k. Since k does not hold the token anymore, it has to send a request to node l. This causes transmission of extra messages, simply because node i is not aware of the latest token holder in the network.

In the last experiment, we place the 6 laptops in the fourth floor of CSL to build a two hop network. The considered topology is shown in Figure 2.9, where an edge between two laptops denotes that the two laptops are in the transmission range of each other, and they are able to receive each other's messages. In this experiment, laptops 3 and 4 are placed in the middle and can communicate with all other laptops. On the other hand, laptops 1 and 2 can only communicate with laptops 3 and 4. The same is true for laptops 5 and 6. Laptops 1 and 2 are not able to receive messages of laptops 5 and 6 and vice versa. Since the considered network is a two-hop network, all transmitted messages cannot be overheard by all network nodes.

This degrades the performance of *TOA*, since nodes are not aware of the latest token holder in the network. Furthermore, the lossy channel causes message loss, in which nodes do not overhear all messages transmitted by the neighboring nodes. The number of transmitted messages per CS entry for *TROA* and Raymond's algorithm are shown in Figure 2.10. As we observe, in this setting, the imperfection in overhearing all messages transmitted in the network degrades the performance of *TOA* and the improvement obtained by message overhearing in *TOA* is marginal.

2.9 Conclusion

We design two distributed token based mutual exclusion algorithms for wireless networks, called *TOA* and *TROA*. Our algorithms exploit the shared nature of the wireless channel in which nodes can overhear the messages not intended for them. We measured the performance of our algorithms as well as Raymond's algorithm and Trehel-Naimi's algorithm through ns-2 simulations as well as implementations, in networks of different sizes and under various rates of the demand for the token. We discussed under what conditions the performance of the considered MUTEX algorithms is improved by exploiting message overhearing.

CHAPTER 3

TOKEN-DCF: AN OPPORTUNISTIC MAC PROTOCOL FOR WIRELESS NETWORKS

IEEE 802.11 DCF is the MAC protocol currently used in wireless LANs. However, due to idle and collision times, 802.11 DCF performs poorly when it comes to channel utilization, system throughput, and channel access time. To overcome these sources of inefficiency in 802.11 DCF, in this chapter, we propose a distributed and dynamically adaptive MAC protocol for wireless networks, called Token-DCF. Main focus of our approach is on reducing idle and collision times by introducing an implicit token passing algorithm. In Token-DCF, a transmitting station schedules one of its neighboring stations for the next transmission epoch using a distributed opportunistic algorithm. Furthermore, packet overhearing is employed to exchange scheduling information across the network. Our simulation results show that Token-DCF can achieve more than 2X improvement in system throughput and channel access delay compared to 802.11 DCF for most network configurations.

3.1 Introduction

IEEE 802.11 defines the distributed coordination function (DCF) to share the wireless medium among multiple stations. DCF employs CSMA/CA with a binary exponential backoff algorithm to resolve channel contention. DCF specifies random backoff, which forces a station to defer its access to the channel for a random period of time. This backoff period corresponds to the number of idle slots a station has to wait before its transmission attempt. If multiple stations choose the same backoff, they will attempt to transmit at the same time and collisions will occur. Two types of overhead are associated with random access protocols. One is channel idle time (i.e., backoff time) which is the time when contending stations are waiting to transmit. Another is collision which happens when multiple stations transmit simultaneously. If

there are few contending stations, idle time is the dominant overhead. If there are many contending stations, collision probability increases and becomes the main source of low channel utilization.

In this chapter, we design a distributed MAC protocol, called *Token-DCF*, in which both idle time and collision time are reduced and network throughput is improved significantly. In Token-DCF, when a station transmits on the channel, it might give a privilege (i.e., a token) to one of its neighbors. When a transmission ends, the privileged station, if there is any, starts transmitting after a short period of time, namely SIFS (Short Inter Frame Space). Non-privileged stations follow the backoff procedure of 802.11 to access the channel. In this way, the privileged station does not go through the contention resolution phase and grabs the channel immediately. A distributed scheduling algorithm is used for choosing the privileged stations.

Token-DCF is fully distributed and does not require any centralized point of coordination. In Token-DCF, a station might schedule one of its neighbors for transmission on the channel. In this way, each network station acts as a scheduler. Token-DCF uses an opportunistic approach based on packet overhearing for exchanging scheduling information as well as token passing. In Token-DCF, queue length of a station is included in the MAC header of the transmitted packets and is overheard by the neighboring stations. Each station keeps track of queue length of its neighbors. Queue length information is used in the scheduling component of the protocol, where a neighbor of the transmitting station is selected as the privileged station. No extra control packet is needed for giving a privilege to a station. Instead, the next privileged station (i.e., the scheduled station) is specified in the MAC header of data packets being transmitted on the channel. The probability of giving a privilege is always less than 1 to cope with newly arrived traffic as well as imperfections in traffic estimation. This probability is adjusted based on the accuracy of the neighbors' traffic estimation. Token-DCF is an opportunistic MAC protocol which behaves similar to 802.11 DCF when packets are not overheard by the neighboring stations. However, when the opportunistic overhearing is feasible, we can then eliminate the backoff procedure of 802.11 DCF to improve efficiency.

The rest of this chapter is organized as follows. We first review some related work in Section 4.1. We then present our protocol, Token-DCF, in Section 4.3. We compare our protocol with IEEE 802.11 in Section 4.4 and

finally present concluding remarks in Section 4.5.

3.2 Related Work

We summarize the prior work into:

1. Distributed MAC protocols to improve the efficiency of 802.11 DCF [17], [18], [19], [20], [21], [22], [23].
2. Token passing MAC protocols [24], [25], [26].
3. Scheduling algorithms of wireless networks [27], [28], [29], [30], [31].

3.2.1 Enhancing 802.11 DCF

Various MAC protocols have been proposed to improve the efficiency of DCF. Cali et al. modify the backoff algorithm of the IEEE 802.11 MAC protocol and derive a contention window size that maximizes network throughput [17]. The backoff window size is tuned at run-time to increase the overall throughput. In this protocol, for light and medium load conditions, where the window size defined in 802.11 DCF is sufficient for guaranteeing low collision probabilities, the standard backoff algorithm is adopted. On the other hand, when the network congestion increases, based on the existing load condition, a contention window with the right size is used.

Tay et al. consider a network in which all stations become simultaneously backlogged at some point in time. They proposed *CSMA/p** which finds the optimal backoff distribution for all stations [18]. In *Idle Sense* [19], each host observes the average number of idle slots across its transmission attempts to dynamically control its contention window. Idle Sense enables each host to estimate its frame error rate, which is used for switching to the right bit rate. In *Implicit pipelining* [20], the task of contention resolution and packet transmission is partially paralleled. This technique reduces the channel idle time and collision time.

Our protocol, Token-DCF, reduces idle time and collision time by implementing an opportunistic token passing algorithm. When a transmission ends, the station holding a token, if there is any, may immediately transmit

after waiting for an idle duration of SIFS. A distributed scheduling algorithm that considers network status (e.g., links' queue length) is used to choose the next station receiving a token. This results in higher channel utilization and system throughput.

Zeng et al. present *CHAIN* [21], in which clients maintain a precedence relation among one another, and a client can immediately transmit a new packet after overhearing a successful transmission of its predecessor. When the network load is low, CHAIN behaves similar to DCF; However, when the network becomes congested, clients automatically start transmission chains to improve efficiency. CHAIN requires transmission of control packets between an access point and its stations periodically, which adds overhead to the protocol. Furthermore, during each scheduling period, the specified precedence relation is fixed and does not adapt to traffic changes during that period.

IEEE 802.11 itself has been enhanced in a number of ways recently. For instance, IEEE 802.11e [22] has introduced the concept of transmission opportunities (TXOPs). A station that gains access to the channel can transmit multiple of frames separated by a SIFS. Thus, a backoff stage does not occur before each packet transmission; Instead, backoff happens before each TXOP. Also, an exchange mechanism called the *Reverse Direction (RD) protocol* has been introduced by IEEE 802.11n [23]. In this mechanism, once the transmitting station has obtained a TXOP, it may grant permission to another station to send information back during its TXOP. The transmitting station sends its permission to the RD responder using a Reverse Direction Grant (RDG) frame. The responder starts the response burst a SIFS after RDG. Our protocol, Token-DCF, is more general than these mechanisms, because in Token-DCF during each transmission, any station might be chosen as the privileged station, where a privileged station transmits on the channel without going to the backoff procedure.

3.2.2 Token passing MAC protocols

Token passing is a medium access method where a short packet called *token* is passed between stations to authorize a station for transmission. In token passing protocols, stations take turns in transmitting by passing the token

from one station to another. Stations that have data frames to transmit must first acquire the token before they can transmit them. A station can only send data if it possesses the token; Thus, avoiding collisions. Token passing schemes provide round-robin scheduling method. Their advantage over contention-based medium access is that collisions are eliminated, and the available bandwidth can be fully utilized when there is a high demand. On the flip side, when the demand is light, a station wishing to transmit must wait for the token, increasing latency.

The IEEE 802.4 Token Bus protocol [24] is a well-known example of token passing protocols. This protocol is based on a broadcast medium (e.g., broadband coaxial cable), which connects all nodes to each other. The token is passed among a logical ring of stations attached to the medium. The order in which stations receive the token is determined based on their MAC addresses.

The Wireless Token Ring Protocol (WTRP) [25] is a token bus protocol, derived from IEEE 802.4. WTRP presents a token passing MAC protocol for wireless networks. When token passing is to be used in a WLAN, the characteristics of the wireless medium, such as connectivity loss, network partitioning and token loss, raise additional token management issues. WTRP is capable of recovering from token loss and duplication, and also dealing with changes in network connectivity and membership. The main modifications to 802.4, introduced by WTRP, address the partial connectivity issues in wireless networks.

Johnson et al. design another token passing MAC protocol for wireless networks, called High Frequency Token Protocol (HFTP) [26]. HFTP is based on WTRP, but adds two new mechanisms: token relaying and ring merging. Token relaying deals with a situation where a station attempts to pass a token to its successor, but fails to receive an acknowledgement due to link failure. In such a scenario, HFTP attempts to find an indirect path to its successor rather than reorganizing the ring to exclude that link. This requires new mechanisms to find and use token relay nodes. HFTP also differs from WTRP in how it merges rings that come into each other's range. This can occur after a network that was partitioned regains connectivity.

Our protocol, Token-DCF does not use round-robin scheduling for passing the token among network stations. In round-robin scheduling, when demand is low, the token might be given to stations with no traffic, which

results in under-utilization of the medium. Instead, in Token-DCF, every station estimates queue length of its neighbors and the token is always given to a neighboring station with a non-zero queue length. Furthermore, token passing mechanism of Token-DCF is implicit, in which, no additional token message is transmitted to pass the token from one station to another. Instead, token passing is done via embedding the scheduling information in the header of data packets by the source station and overhearing the packets by the neighboring stations to retrieve such information. When opportunistic overhearing is not feasible, i.e. if token is not received by the privileged station, Token-DCF operates similar to 802.11 DCF. This method eliminates the need for dealing with complicated token management issues in wireless networks such as recovering from connectivity loss, network partitioning and token loss.

3.2.3 Scheduling algorithms of wireless networks

Prior work on scheduling algorithms of wireless networks can be largely classified into two main categories:

1. Throughput-optimal scheduling: Here it is assumed that the mean arrival rates of the packets into each queue is within the capacity region, where capacity region is defined as the set of sustainable arrival rates of the channel. The centralized scheduler knows the current queue lengths and the current channel conditions. The first throughput optimal scheduling algorithm was introduced in the seminal work of Tassiulas and Ephremides [27]. The proposed algorithm is a centralized algorithm known as Backpressure. In Backpressure algorithm, the schedule at each time slot t is determined by

$$\vec{r}(t) = \text{argmax}_{\vec{r}\in\mathcal{R}}\left[\sum_{(i,j)}(q_i - q_j)r_{ij}\right] \tag{3.1}$$

   For each link (i, j) from station i to station j, $(q_i - q_j)$ denotes its queue differential and r_{ij} denotes its rate. $\mathcal{R}$ is the convex hull of the capacity region. In Backpressure, at each time slot, the set of non-conflicting links that maximizes the above sum is activated.

   Longest-Queue-First scheduling (a.k.a., greedy maximal scheduling)

[28] is another centralized scheduling algorithm, which has been observed to achieve throughput optimality in most practical wireless networks. LQF makes scheduling decisions based on the queue length information as follows. It starts with an empty schedule and first adds the link with the largest queue length to the schedule. It then looks for the link with the largest queue length among the remaining links. This selected link will be added to the schedule only if this addition creates a feasible schedule (i.e., the set of added links satisfies the SINR constraints). This process continues until no more link can be added to the schedule.

Throughput optimal scheduling algorithms are generalized in many different directions [29], [30], [31]. In throughput optimal scheduling algorithms, queues are stable if the arrival rates lie within the capacity region. Throughput-optimal scheduling is suitable for inelastic traffic where the sources do not adapt their transmission rate based on congestion in the network. In this case, admission control is required to ensure that the arrival rates lie within the capacity region of the network.

2. Fair Scheduling: An obvious drawback of throughput optimal policies is that no traffic policing is enforced. For instance, if one or more sources misbehave and increase their arrival rates so that the set of arrival rates lies outside the capacity region, then the system becomes unstable. In other words, all flows will be penalized due to the behavior of a few misbehaving flows. Thus, an alternative is to allocate resources in a fair manner to the various queues. Two examples of fair allocation are weighted proportional fair allocation and max-min fair allocation [32]. Fair scheduling is more suited for elastic traffic sources which can adjust their traffic rates in response to feedback from the network regarding the network conditions.

Various scheduling algorithms can be incorporated in our protocol, Token-DCF, depending on the objective of the scheduling algorithm and type of the arrival traffic.

3.3 Token-DCF Design

In this section, we first provide a high-level overview of Token-DCF and then detail the scheduler signaling and algorithm.

3.3.1 Overview

At a high level, the operation of Token-DCF is described as follows. Token-DCF runs an opportunistic token passing protocol, where a token (or a privilege) might be assigned by a transmitting station to one of its neighbors. In this chapter, we use the terms privilege and token interchangeably. While transmitting, the transmitting station might select one of its neighbors and give it a higher priority for the next transmission. Various selection mechanisms can be used. When a transmission ends, the station with a higher priority, called ***privileged***, starts transmitting after a short period of time (i.e., SIFS), if the channel is sensed idle. Since all other stations should wait for at least a longer DIFS, the transmission of the ***privileged*** station will not collide with other transmissions.

Token-DCF is implemented in the MAC layer of the protocol stack. Scheduling information is embedded in the MAC header of data packets and is transferred to the neighboring stations via overhearing. Each station maintains queue length of the neighboring stations. These queue lengths are then used in the scheduling phase to select the privileged station for the next transmission. Transmitting station announces the privileged station in the ***privileged*** field of the MAC header of the data packets it transmits. By overhearing these packets, the privileged station is informed that it has a higher priority for the next transmission. When a transmission ends, the privileged station can start transmitting after SIFS if the channel is sensed idle. If opportunistic overhearing does not work, i.e., token is not received by the next privileged station, Token-DCF operates similar to 802.11 DCF. But when the next privileged station overhears the token, it can transmit on the channel without going to the backoff procedure.

Signaling mechanism in the scheduling component of Token-DCF is done via embedding the scheduling information in the header of data packets by the source station and overhearing the packets to retrieve such information by the neighboring stations. When a packet is transmitted, the ***privileged***

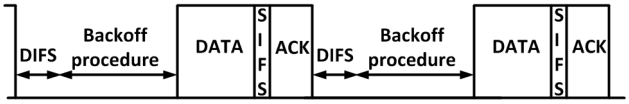


Figure 3.1: Access method of IEEE 802.11 DCF

station and the queue length of the transmitter are embedded in the MAC header of the packet. Once a packet is received or overheard, the queue length of the source of the packet is retrieved. Furthermore, a neighboring station checks the *privileged* field to find out if it is privileged for the next transmission. In Token-DCF, no extra control messages are transmitted to obtain network status and to pass the privileges. Collecting the information needed for scheduling, assigning a privilege to one of the neighbors and obtaining the privilege by the *privileged* station are all done through overhearing.

Token-DCF has two major components: (1) A method to reduce the idle time of the backoff procedure. (2) A scheduling algorithm to determine which neighbor should be chosen as the privileged station.

3.3.2 Reducing idle time

Token-DCF reduces the idle time of the backoff mechanism by assigning privileges to network stations. When a station transmits data packets, it might give a higher priority to one of its neighbors for the next transmission with probability p and with probability $1-p$, no station is privileged. As we will explain in Section 3.3.3, the scheduling algorithm of Token-DCF determines which neighbor is chosen as the privileged station. When a transmission ends, the privileged station starts transmitting after SIFS, if the channel is sensed idle. Non-privileged stations follow the backoff procedure of IEEE 802.11 to access the wireless medium. Backoff mechanism of 802.11 DCF is shown in Figure 3.1. In this mechanism, after a transmission ends, the station senses the channel after DIFS interval and if the channel is sensed idle, it waits for a random backoff time. It chooses backoff b, an integer distributed uniformly in the window $[0, CW]$, and waits for b time slots before trying to transmit.

Channel access method of our protocol is shown in Figure 3.2. In Token-DCF, when the channel becomes idle, the privileged station, if there is any, starts transmitting on the channel immediately, and non-privileged stations

have to defer backoff count down till when transmission of the privileged station finishes. This process of giving a privilege to one of the neighbors of the transmitting station repeats in each transmission. Whenever a privileged station transmits on the channel, the idle time of the channel is limited to SIFS. On the other hand, in IEEE 802.11 protocol, the channel idle time between two consecutive transmissions is equal to DIFS plus a random backoff duration. Furthermore, since the privileged station immediately transmits after waiting an idle duration of SIFS, while all other stations should wait for at least a longer DIFS, the transmission of the privileged station will not collide with other transmissions.

3.3.3 Scheduling algorithm

The scheduling algorithm of Token-DCF provides a mechanism for choosing the privileged stations. In Token-DCF, when a station transmits, it acts as a scheduler as well and with probability p gives a higher priority for the next transmission to one of its neighbors. This removes the need for a separate scheduler as well as transmission of control messages between the scheduler and network stations. If a privilege is assigned to a station with an empty queue, the privileged station would not take its chance to transmit on the channel without going through the contention phase. This simply results in under-utilization of the underlying wireless channel. As long as the privilege is assigned to a station with backlogged traffic, the privileged station can immediately transmit a new packet without dealing with contention.

Different scheduling algorithms can be used for choosing the next privileged station, depending on the objective of the scheduling algorithm and type of the traffic. Here, we present two example scheduling policies. In the first algorithm, a transmitting station picks the neighbor with the largest q_i as the next privileged station, where q_i is the queue length of station i. In single hop networks, if every station overhears every transmission, this policy implements Longest-Queue-First [28] as the scheduling component of Token-DCF, which guarantees throughput optimality. In the second algorithm, a transmitting station uniformly at random chooses one of its neighbors with backlogged traffic (i.e., with non-zero queue length). This policy achieves fairness among the network stations, because in this policy all stations have

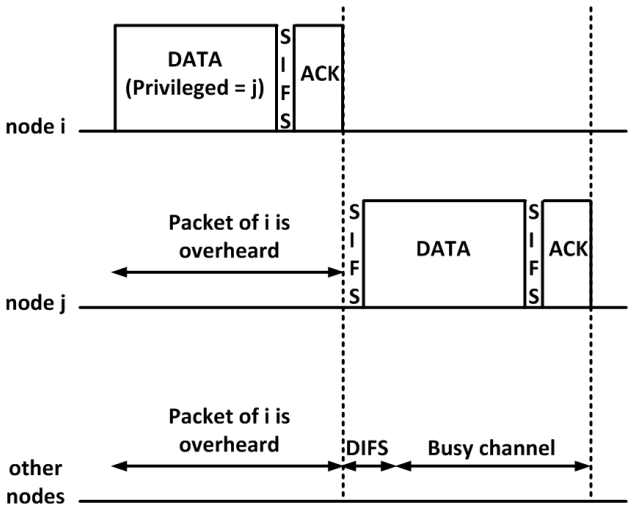


Figure 3.2: Access method of Token-DCF protocol

an equal chance to be chosen for transmitting on the channel.

3.3.4 Protocol details

The detailed Token-DCF is presented in this section. Procedure 11 sets the initial values of Token-DCF parameters. p, the probability of giving a privilege, is initially set to zero and changes during the execution. *active* denotes the set of neighbors of a station that has transmitted on the channel during the current scheduling period and the transmission is overheard by the station. The station itself, *myId*, is also included in the set *active*. When a station transmits, it might give a privilege to one of the stations in the set *active*. By including *myId* in the set *active*, a station might choose itself as the *privileged*. Each station keeps track of the transmissions on the channel by overhearing of the packets. *nFail* denotes the number of transmissions in which the sender of the packet is not in the set *active*. *nSuccess* denotes the number of transmissions from the set *active*. *flag* is a Boolean variable denoting if the station has a privilege for accessing the channel or not. *flag* equals to `true` means that the station has a privilege for transmission on the channel. *flag* is initially set to `false`, meaning that no station is privileged initially. Initially, a station with data for transmission has to go through the backoff process to access the medium. Protocol parameters are reset to initial values after each *period* seconds. Protocol parameters are reset

periodically in order to prevent the stale information from degrading the protocol performance.

11 Initialization at station *myId*

1: $p = 0$
2: $active = \{myId\}$
3: $nFail = 0$
4: $nSuccess = 0$
5: $flag =$ `false`
6: call Initialization after *period*

Procedure 19 is executed right before a packet is transmitted on the channel. If the packet is a MAC data packet, the station might give a privilege to one of its neighbors. The mechanism of assigning a privilege or transmitting as the privileged station is not used when control packets are transmitted. In this way, the transmission of non-data packets such as ARP packets or routing packets are not affected by our protocol. The station that is chosen to be the privileged station is called *privileged*.

As explained before, different criteria can be used for choosing the privileged station, *privileged*. One example scheduling algorithm is presented in 19. In 19, *privileged* is the station in the set *active* with the longest queue. Another example of scheduling algorithms is the one in which *privileged* is chosen uniformly at random from the set of stations in *active* with non-zero queue length. This policy achieves fairness among the network stations. Many other queue based scheduling algorithms are presented in the literature that can be incorporated in Token-DCF protocol [27], [29], [30].

If a station chooses itself as the *privileged*, it sets its *flag* to `true`. Otherwise, *flag* is set to `false`. Procedure 16, called Adapt, is then called to update *nSuccess*, *nFail* and *p*.

Procedure 20 is called when a packet is received or overheard. Since the wireless channel is a shared medium, station i might overhear packets that are not intended for it, i.e., packets with destination address different from i. If the station is chosen to be the *privileged* in the received or overheard packet, it sets its *flag* to `true`. Otherwise, *flag* is set to `false`. The station then calls Adapt (Procedure 16) in which, *nSuccess*, *nFail* and *p* are updated. The station also saves the queue length of *src* in a local variable *qLen*.

12 Transmit a packet

```
1: if transmitting a MAC data packet then
2:     generate a random number r uniformly distributed on [0, 1]
3:     if r < p then
4:         privileged = station with the longest queue in active
5:     else
6:         privileged = null
7:     if privileged == myId then
8:         flag = true
9:     else
10:        flag = false
11:    Adapt
12: else
13:    privileged = null
```

13 Receiving or Overhearing a packet from station *src*

```
1: if privileged == myId then
2:     flag = true
3: else
4:     flag = false
5: Adapt
6: qLen[src] = queue length of src
```

Procedure 14 is executed when a station starts or resumes its backoff timer. If the station has higher priority (i.e., *flag* == `true`) and the packet is a MAC data packet, the backoff duration is set to SIFS. Otherwise, the backoff duration is chosen to be DIFS plus random number of time slots, similar to 802.11 DCF.

14 Starting or resuming backoff timer

```
1: if flag == true && packet is a MAC data packet then
2:     schedule backoff timer for SIFS
3: else
4:     schedule backoff timer for DIFS + random number of time slots
```

As explained in Procedure 15, when the backoff timer expires, *flag* is reset to `false`. In this way, a privileged station has the privilege to transmit only one packet immediately after the last transmission ends. In case the packet is lost, the station does not have the privilege for retransmission of the packet and will follow the backoff procedure to access the channel. When a host detects a failed transmission (it does not receive the ACK of a frame),

it executes the exponential backoff algorithm, doubling Contention Window CW (CW may vary between CW_{min} and CW_{max}).

15 Backoff timer expiration

1: $flag$ = `false`

When a packet is transmitted, received or overheard, Adapt (Procedure 16) might be called, in order to update the value of $nSuccess$, $nFail$ and p. Station i calls Adapt when it transmits, receives or overhears a packet. If transmitter of the packet, src, does not belong to the set $active$, $nFail$ is increased by one and src is added to the set $active$. In this case, the station that receives or overhears the packet, has not received any transmission from src during the current transmission period. Otherwise, if src belongs to the set $active$, $nSuccess$ is increased by 1. Recall that the set $active$ is reset every $period$ seconds.

Enough transmissions should happen before adapting p. If so, (i.e., if $nSuccess+nFail >= maxNum$), ratio of $nSuccess$ to $nSuccess+nFail$ is then recalculated to adapt p. If $ratio$ is larger than a threshold, $maxRatio$, p is increased by δ and $nSuccess$ and $nFail$ are reset to 0. We note that p is increased up to a threshold, $maxP$. It is reasonable to choose $maxP$ less than 1 in order to always give a chance to stations not in the set $active$ to be able to transmit on the channel. If $ratio$ is less than a threshold, $minRatio$, p is decreased by δ and $nSuccess$ and $nFail$ are reset to 0.

What we have presented in Procedure 16 is an example of dynamically adapting the parameter p, the probability of giving a token to a neighbor for immediate transmission on the channel. There are other alternatives for adapting protocol parameters. For example, different moving average techniques (e.g., simple, cumulative, weighted, exponential, $\cdots$) can be used to adapt the parameters. Procedure 17, presents *simple moving average (SMA)* method for adapting p. A simple moving average (SMA) is the unweighted mean of the previous n data points. If we denote the ith access to the channel by x_i, where $x_i = 1$ is equivalent to a transmission from the set $active$, and $x_i = 0$ is a transmission from outside of the set $active$, then we calculate the probability p as the SMA of transmissions on the channel, where

$$p = \frac{x_i + x_{i-1} + \cdots + x_{i-(n-1)}}{n}$$

When calculating successive values, a new value comes into the sum and an old value drops out. The number of samples n, depends on the type of network users and network traffic, where the status of the network might change in short, intermediate, or long term.

16 Adapt

```
1: if src ∉ active then
2:     nFail ++
3:     add src to active
4: else
5:     nSuccess ++
6: if (nSuccess+nFail >= maxNum) then
7:     ratio = nSuccess / (nSuccess+nFail)
8:     if (ratio >= maxRatio) then
9:         if (p <= maxP) then
10:            p = p + δ
11:        nSuccess = 0
12:        nFail = 0
13:    if (ratio <= minRatio) then
14:        if (p >= δ) then
15:            p = p − δ
16:        nSuccess = 0
17:        nFail = 0
```

17 Simple Moving Average (SMA)

```
1: if src ∉ active then
2:     x_i = 0
3: else
4:     x_i = 1
5: p = (x_i + x_{i−1} + ··· + x_{i−(n−1)}) / n
```

3.4 Evaluation

We simulate Token-DCF and 802.11g in ns-2 to measure and compare performance of these two MAC protocols. Table 3.1 reports the configuration parameter values of the wireless network analyzed in this section. Table 3.2 summarizes the parameter values of Token-DCF chosen in the simulations. Transmit power and carrier sense threshold reported in Table 3.2 are default

Table 3.1: WLAN configuration

Transmit power	24.5 dBm
Carrier sense threshold	-78 dBm
SIFS	10 μsec
DIFS	28 μsec
slot time	9 μsec
phy preamble	16 μsec
bit rate	54 Mbps
CWmin	15
CWmax	1023

values of ns-2. The rest of the parameters reported in Table 3.2 are the same as values used in [21]. The network is a wireless ad hoc network in which transmitting stations are placed uniformly at random in a square area. Flows might be single hop or multi hop. In the case of single hop flows, the receiver of each flow is placed at a distance of $100m$ from the transmitter of the flow. In case of multi-hop flows, receiving stations are placed uniformly at random in the area. We run the simulations for different network sizes, including single-hop and multi-hop networks. The effective transmission range in the simulations is limited to 250 meters and carrier sense range is limited to 550 meters. IEEE 802.11 RTS/CTS mechanism is turned off. Two-ray ground radio propagation model is assumed. Packet payload size is 1500 bytes. Each simulation lasts for 30 seconds and the presented results are averaged over 20 runs. In each run, a different random network topology is considered. In our simulations, the scheduling algorithm presented in Procedure 19 is used as the scheduling component of Token-DCF. We measure the performance of Token-DCF and 802.11 DCF in terms of aggregate throughput, average access delay, channel idle time and collision frequency.

3.4.1 Performance evaluation in saturated single-hop networks

Figures 3.3(a) - 3.3(d) plot the performance parameters in a single-hop network. The size of the network is $150m$x$150m$ and all flows are single-hop. Traffic is full buffer CBR, meaning that there is always backlogged traffic in

Table 3.2: Token-DCF parameters

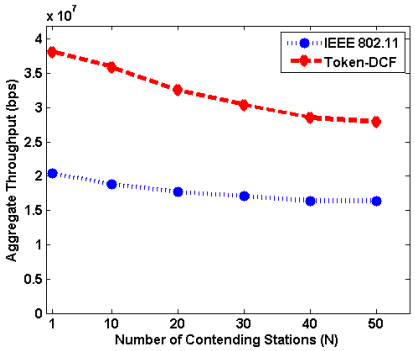
Parameter	Value
minRatio	0.2
maxRatio	0.8
maxNum	20
maxP	0.9
period	0.1 sec
δ	0.1

the transmission queue of each transmitter.

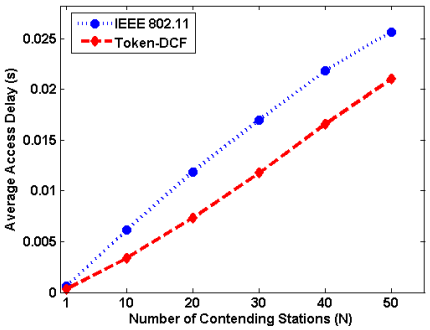
The aggregate throughput of 802.11 DCF and Token-DCF is presented in Figure 3.3(a). As can be seen, throughput gain obtained by Token-DCF compared to IEEE 802.11 in Figure 3.3(a) is a factor of $1.7 - 1.9$. Figure 3.3(b) shows the average access delay of the two protocols. Access delay is defined as the delay between the time a packet arrives at the MAC layer and the time the source of the packet receives acknowledgment from the destination. Access delay of a packet consists of the waiting time before transmitting on the channel and the time spent in packet retransmissions. As we can see in Figure 3.3(b), access delay is smaller in Token-DCF by a factor of $0.53 - 0.81$. As we will explain, the reason is that Token-DCF has a much shorter idle time compared to IEEE 802.11 DCF. Furthermore, many retransmissions are avoided because of reduced collision frequency.

Figure 3.3(c) presents the average number of idle slots before each media access. Token-DCF has shorter channel idle time, because in Token-DCF, a privileged station accesses the channel immediately after the latest transmission finishes. In this way, channel stays idle only for SIFS seconds, instead of DIFS plus random backoff duration. We note that the average number of idle slots in Token-DCF is not zero. The reason is that with a non-zero probability, no station is chosen as the privileged station for the next transmission. In such a case, stations follow the backoff mechanism of 802.11 DCF to get an access for transmission on the channel.

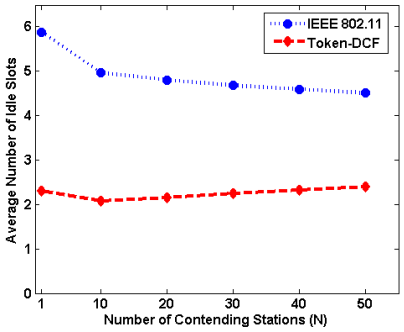
Collision frequencies of 802.11 DCF and Token-DCF are shown in Figure 3.3(d). Collision frequency is defined as the number of times a transmission fails due to collision normalized by the total number of transmissions (counting retransmissions as well). Figure 3.3(d) indicates that Token-DCF has a



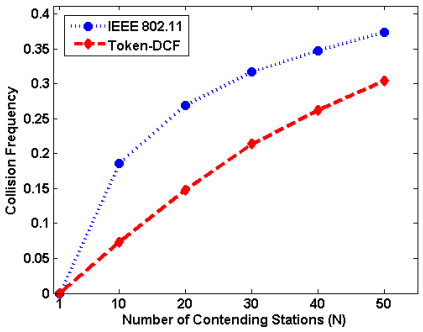
(a) Aggregate throughput



(b) Average access delay



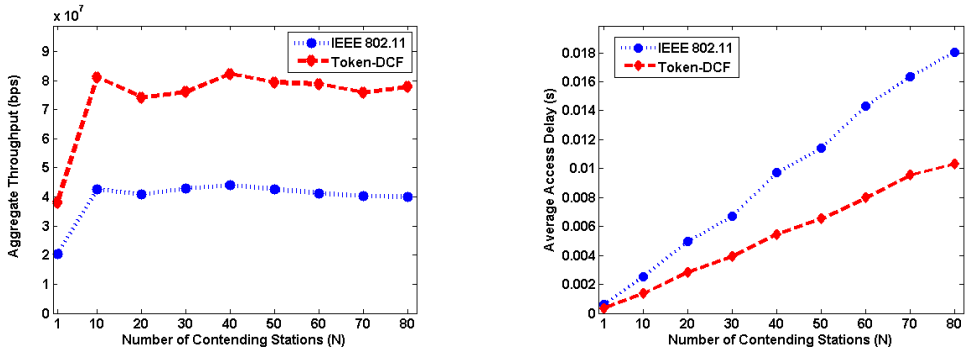
(c) The average number of idle slots before each media access



(d) Collision frequency

Figure 3.3: Full buffer CBR traffic, area = 150 m x 150 m

much lower collision frequency than 802.11 DCF. Recall that when a station transmits, it might choose one of its neighbors as the privileged station. In a single-hop network, at each time instant, at most one station successfully transmits on the media and as a result, there is at most one privileged station at each time instant. Since a privileged station does not follow the backoff mechanism of 802.11 DCF, the transmission by a privileged station does not collide with any other transmission in a single-hop network. This reduces the collision frequency of the protocol. Reducing the idle time and collision time of the channel increases throughput and decreases media access delay. As we can see in Figure 3.3(d), with greater number of contending stations, the collision frequency in both Token-DCF and 802.11 DCF increases. Token-DCF has non-zero collision frequency, because with probability $1-p$, stations implement backoff mechanism for contention resolution, which might cause collisions.



(a) Aggregate throughput (b) Average access delay

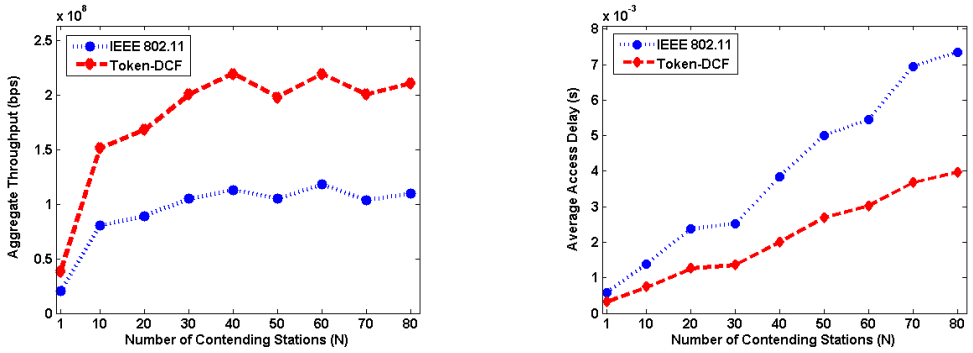
Figure 3.4: Full buffer CBR traffic, area = 800 m x 800 m

3.4.2 Performance evaluation in saturated multihop wireless networks

In this section, we study performance of Token-DCF in multihop wireless networks. We consider two network sizes; $800m$x$800m$ and $1500m$x$1500m$. Recall that the effective transmission range in the simulations is limited to 250 meters and carrier sense range is limited to 550 meters. Traffic is full buffer CBR and all flows are single hop. Aggregate throughput and average access delay of the networks with size $800m$x$800m$ versus number of contending stations are presented in Figures 3.4(a) and 3.4(b), respectively. Comparing Token-DCF and 802.11 DCF in these two figures, we can see that throughput gain is a factor of $1.8-2$ and access delay is reduced by a factor of $0.53-0.58$. For the networks of size $1500m$x$1500m$, aggregate throughput and average access delay are presented in Figures 3.5(a) and 3.5(b), respectively. In this case, throughput gain is a factor of 1.9 and access delay is reduced by a factor of $0.52-0.55$. Considering Figures 3.3(a)-3.5(b), we see that similar performance improvement is obtained by Token-DCF in single hop and multihop networks. The reason is that, in multi-hop networks, Token-DCF improves the channel utilization in each transmission range.

3.4.3 Stations with unsaturated traffic

Having shown the performance improvement of Token-DCF over 802.11 for saturated networks, we further identify its performance in networks that have less traffic load. This set of simulations focuses on comparing the performance



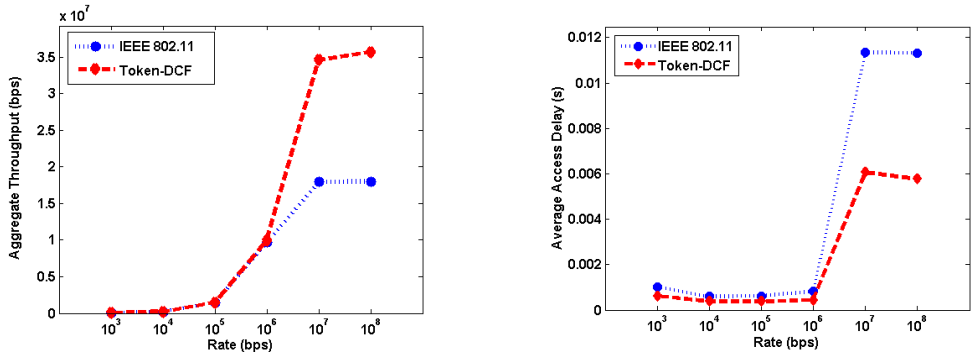
(a) Aggregate throughput (b) Average access delay

Figure 3.5: Full buffer CBR traffic, area = 1500 m x 1500 m

of Token-DCF with 802.11 when varying the traffic load from low to high. On/Off traffic with burst times and idle times taken from pareto distributions is used. Average on time of the traffic generator is $50ms$. Its average off time is also set to $50ms$.

We perform simulations for a randomly generated network of size $150m$x$150m$. There are a total of 20 one-hop flows. Each source station generates its packets independently. Sending rate during on time, called Rate, is varied between 10^3 *bps* and 10^8 *bps*. With Rate = 10^3 *bps* and 1500 bytes packet size, the traffic demand is far below the network capacity. When gradually varying Rate from 10^3 to 10^8 bps, offered load is increased from small to very large. The corresponding aggregate throughput and average access delay are presented in Figures 3.6(a) and 3.6(b), respectively. When the network load is very low, station queues are empty most of the time in which case, no station is chosen as the privileged station. Under low load, Token-DCF behaves very similar to 802.11 DCF. Their performance starts to diverge when the network is loaded more heavily. The saturation throughput of Token-DCF is approximately 2 times of 802.11 DCF.

As we explained before, access delay is defined as the delay between the time a packet arrives at the MAC layer and the time the source of the packet receives acknowledgment from the destination. Access delay of a packet consists of the waiting time before transmitting on the channel and the time spent in packet retransmissions. As we observe in Figure 3.6(b), access delay is approximately the same for Rate = 10^7 *bps* and Rate = 10^8 *bps*, i.e., access delay does not increase with increasing Rate after some point. The reason is that queueing delay is not considered in computing access delay.



(a) Aggregate throughput (b) Average access delay

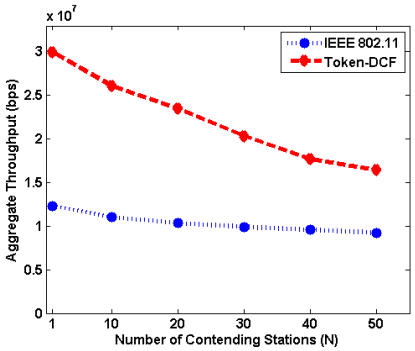
Figure 3.6: Pareto traffic

If queueing delay is considered in computing access delay, access delay is increased by increasing Rate, since packets encounter larger queueing delay at higher Rates. With very high Rates, some packets might get dropped, in which case queueing delay goes to infinity.

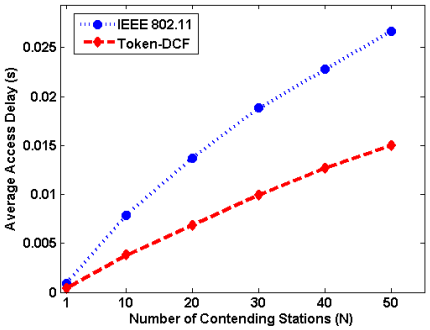
3.4.4 Networks with multi-hop TCP flows

In this section, we study Token-DCF's performance under single hop and multi-hop TCP traffic. We perform simulations for networks of different sizes, i.e., $150m$x$150m$, $800m$x$800m$ and $1500m$x$1500m$. Transmitting and receiving stations of each flow are placed uniformly at random in the area. As a result, for networks of size $800m$x$800m$ and $1500m$x$1500m$, where network is multi-hop, flows might also be multi-hop. Destination-Sequenced Distance-Vector Routing (DSDV) is used as the routing protocol. Figures 3.7(a) and 3.7(b) show the total throughput and average access delay for single-hop networks of size $150m$x$150m$. Comparing Figures 3.3(a) and 3.7(a), we can see that the performance improvement of Token-DCF over 802.11 is similar for both TCP and saturated CBR traffic. When traffic is TCP, although buffer of stations might not be fully backlogged, stations might have few packets backlogged in their transmission queue, in which case privilege can be given to one of the stations. This results in decreasing the idle time as well as collision time and increasing the throughput.

The results for network size of $800m$x$800m$ are shown in Figures 3.8(a) and 3.8(b). In these networks, throughput gain is a factor of $2 - 2.3$ and

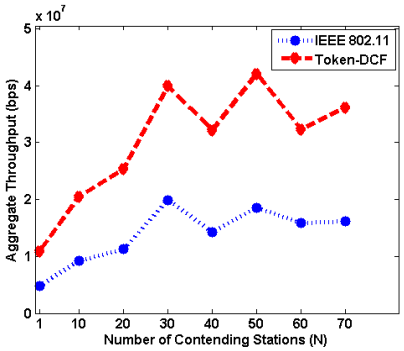


(a) Aggregate throughput

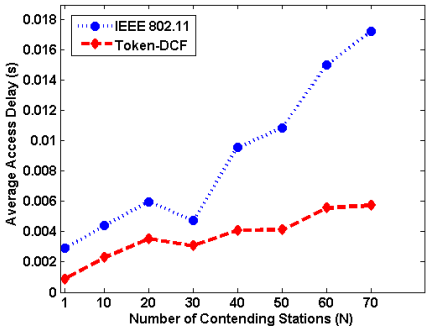


(b) Average access delay

Figure 3.7: TCP traffic, area = 150 m x 150 m



(a) Aggregate throughput



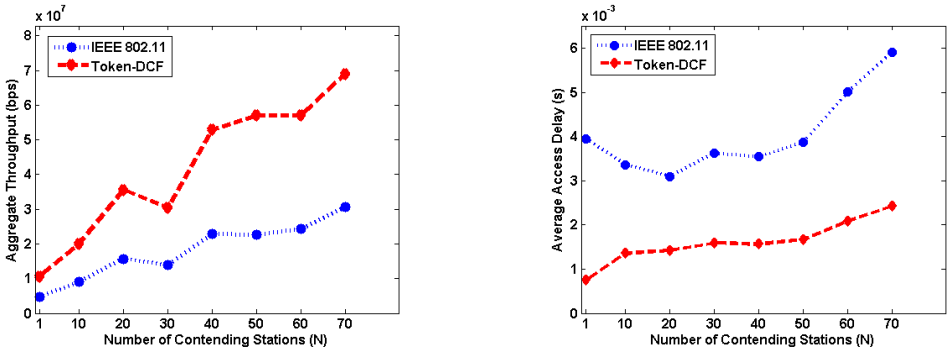
(b) Average access delay

Figure 3.8: TCP traffic, area = 800 m x 800 m

access delay is reduced by a factor of $0.3 - 0.65$. Figures 3.9(a) and 3.9(b) present aggregate throughput and average access delay for networks of size $1500m$x$1500m$. Throughput gain is a factor of $2.2 - 2.5$ and access delay is reduced by a factor of $0.18 - 0.45$. Although flows are multi-hop in these networks, since Token-DCF improves the channel utilization in each transmission range, aggregate throughput gain and delay reduction is similar to single-hop networks.

3.5 Simulations in 802.11Ext Module

We also simulate Token-DCF and 802.11g using 802.11Ext module of ns-2 [33]. 802.11Ext module of ns-2 provides a more accurate modelling of IEEE 802.11protocol, which introduces two new modules: mac-802_11Ext



(a) Aggregate throughput (b) Average access delay

Figure 3.9: TCP traffic, area = 1500 m x 1500 m

and wireless-phyExt. The extension is based on the original simulation of 802.11 protocol in ns-2, but did a major modification to the original code, aiming at a significantly higher level of simulation accuracy.

The 802.11Ext module includes the following key features:

- Structured design of MAC functionality modules, which includes transmission, reception, transmission coordination, reception coordination, backoff manager and channel state monitor modules.
- Cumulative SINR computation.
- MAC frame capture capabilities, preamble capture.
- Multiple modulation scheme support.

In this set of simulations, we simulate two different scheduling algorithms to find out the effect of changing the scheduling policy on the performance of Token-DCF protocol. The simulated scheduling policy are:

1. LQF (Longest Queue First): In LQF scheduling, a transmitting station chooses its neighbor with the longest queue as the next privileged station.
2. Random selection: In this scheduling policy, a transmitting station considers its neighbors with backlogged queue and chooses one of them uniformly at random to be the next privileged station.

We note that in Section 3.4, we only simulated LQF scheduling algorithm. We also simulate the following two algorithms for adapting the probability p

Table 3.3: Parameters of simulations in 802.11Ext module

Transmit power	15 dBm
Thermal Noise	-93 dBm
Carrier sense threshold	-91 dBm
SIFS	10 μsec
DIFS	28 μsec
Slot time	9 μsec
CWmin	15
CWmax	1023
Data Rate	54 Mbps
SINR threshold	23 dB

to investigate the effect of changing the parameter p on the performance of Token-DCF protocol. The simulated algorithms are:

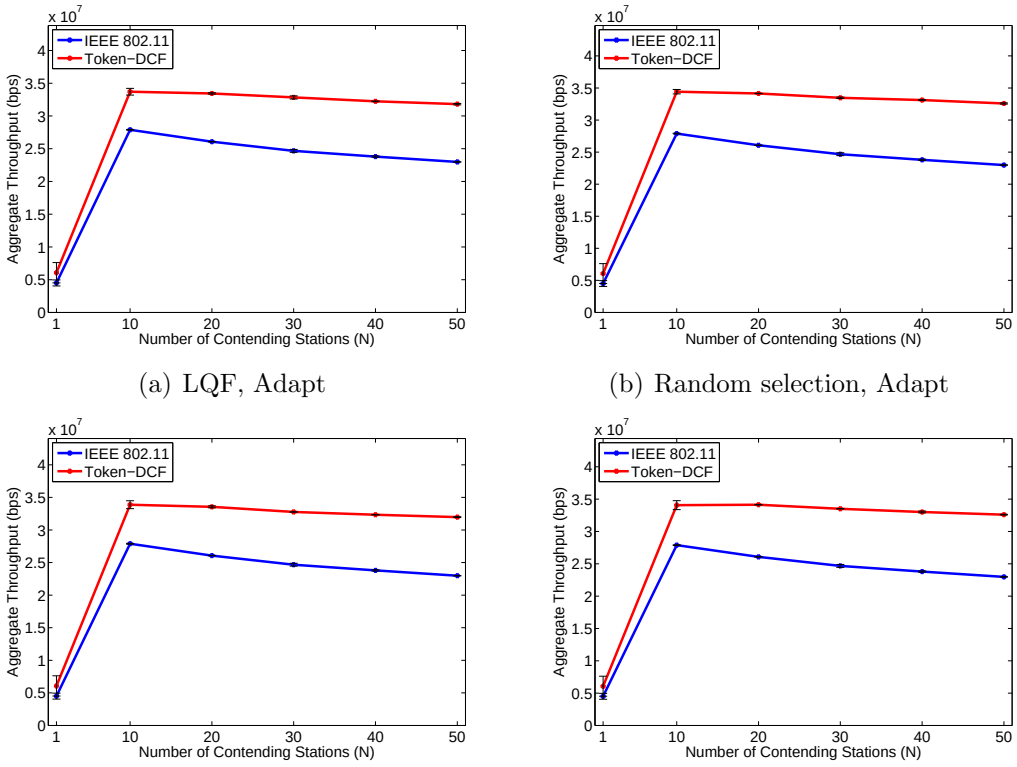
1. Adapt (Procedure 16)

2. Simple Moving Average (Procedure 17)

Table 3.3 reports the configuration parameter values used in this section. The parameter values of Table 3.3 are chosen based on the values reported in [34] and [35]. The parameter values of Adapt procedure are reported in Table 3.2.

The considered network is a single contention domain of size $150m$x$150m$, where at each time instance only one transmission is possible in the network. Signal propagation model is two ray ground. Transmitting stations are placed uniformly at random in the area. The receiving station corresponding to each transmitter is placed at a distance of $100m$ from the transmitter. We vary the number of transmitting stations from 1 to 50. Each transmitting station generates its packets independently. Packet payload size is 1500 bytes.

Figure 3.10 shows the effect of changing the scheduling algorithm as well as the algorithm for adapting p on the performance of Token-DCF. Traffic arrival model at each station is an On/Off traffic with average burst times and average idle times taken from Pareto distribution. During on periods, packets are generated in a station at a fixed rate of $10^7 Mbps$. No packets are generated during off periods. Average on time of the traffic generator is



(a) LQF, Adapt

(b) Random selection, Adapt

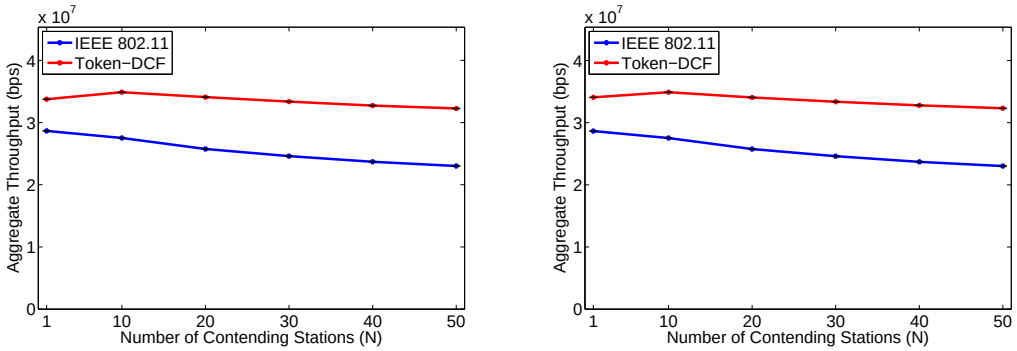
(c) LQF, SMA

(d) Random selection, SMA

Figure 3.10: ON/OFF traffic

500ms. Its average off time is also set to 500ms. This traffic model simulates dynamic arrival and departure of traffic in the network, since each station has backlogged traffic during a randomly chosen on period, and it does not have any traffic during a randomly chosen off period.

In Figures 3.10(a) and 3.10(b), we consider two scheduling algorithms, i.e., LQF in Figure 3.10(a) and random selection in Figure 3.10(b). In these two figures, the algorithm for changing the probability p is fixed to Adapt (Procedure 16). As these two figures show, for both scheduling algorithms, Token-DCF achieves the same throughput. The reason is that both LQF and random selection give the token to some station with backlogged queue. As far as a station with backlogged traffic receives the token, that station can transmit immediately after the channel becomes idle, without going through the backoff mechanism. This results in removing the backoff mechanism and increasing throughput. We conclude that as far as the token is given to a station with backlogged traffic, the choice of the scheduling algorithm does not affect the network throughput of Token-DCF.



(a) LQF, Adapt

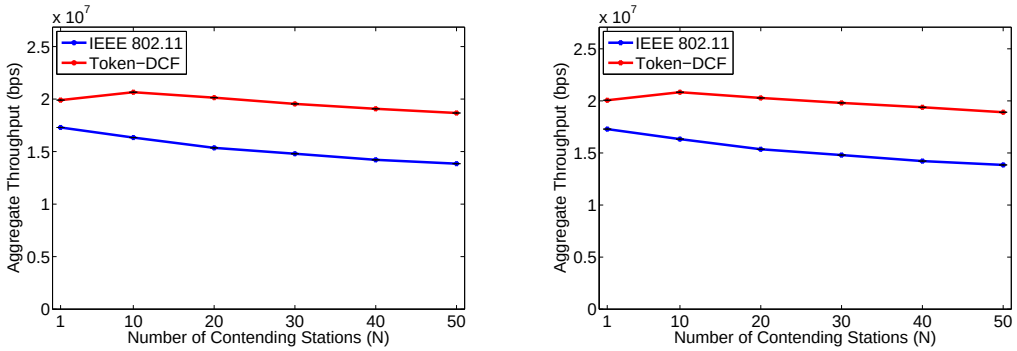
(b) LQF, SMA

Figure 3.11: Full buffer CBR traffic

Figure 3.10 also shows that changing the algorithm for adapting the parameter p does not change the throughput of Token-DCF protocol. In Figures 3.10(a) and 3.10(c) the scheduling algorithm is fixed to LQF. The algorithm for changing p is Adapt in Figure 3.10(a) and SMA in Figure 3.10(c). These two figures show the same throughput for Token-DCF protocol. In Figures 3.10(b) and 3.10(d), we fix the scheduling algorithm to random selection and we vary the adaptation algorithm. We again observe that changing the p adaptation algorithm does not change the throughput of Token-DCF protocol. The reason is that in both Adapt and SMA, parameter p is changed at a time scale much shorter than the time scale by which the traffic arrives and departs. As a result, both p adaptation algorithms can quickly converge to an appropriate value of p, after the traffic in the network becomes stable, i.e., quickly after a traffic arrives in the network or departs the network, the parameter p is adapted to a suitable value.

Figure 3.11 shows the throughput of 802.11g and Token-DCF, when the traffic is full buffer CBR. The algorithm for adapting p is Adapt in Figure 3.11(a) and SMA in Figure 3.11(b). The scheduling algorithm is fixed to LQF for both Figures 3.11(a) and 3.11(b). We observe that in both figures, Token-DCF has the same throughput. With full buffer CBR traffic, queues of network stations are always fully backlogged, meaning that network traffic is static and does not vary. In such a scenario, both Adapt and SMA set p to a value close to 1. As a result, both Adapt and SMA result in same throughput for Token-DCF.

Figure 3.12 shows the throughput of 802.11g and Token-DCF, when the traffic is TCP. The scheduling algorithm is LQF in Figure 3.12(a) and ran-



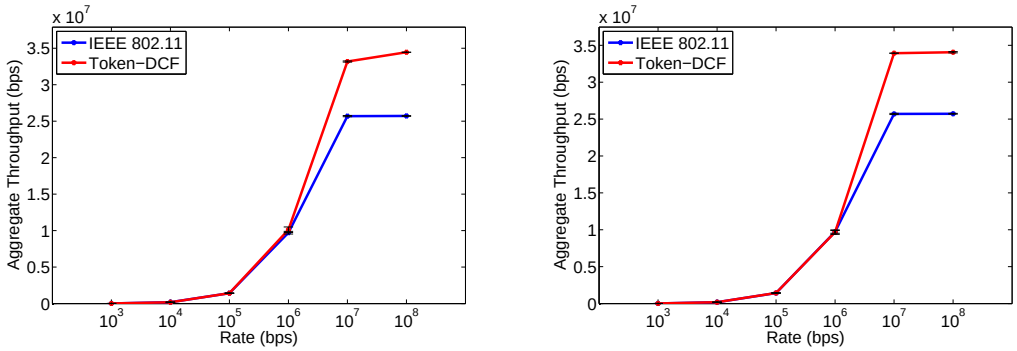
(a) LQF, SMA

(b) Random selection, SMA

Figure 3.12: TCP traffic

dom selection in Figure 3.12(b). The algorithm for adapting p is SMA in both figures. We observe that in both figures, Token-DCF has the same throughput. With TCP traffic, although queues of network stations might not be always fully backlogged, some stations have backlogged packets in their queues. We note that in Token-DCF, each station estimates the queue length of its neighbors via overhearing and a transmitting station gives the token to one of its neighbors with non-zero queue length. Since both LQF and random selection give the token to a neighbor with non-zero queue length, in both algorithms, a station with backlogged queue receives the token and transmits without going to the backoff mechanism. This results in removing the back off mechanism and increasing network throughput, for the case of both LQF and random selection.

Figure 3.13 shows the throughput of 802.11g and Token-DCF, when the traffic is On/Off. In this figure, the number of transmitting stations is fixed to 20. Sending rate during on time, called Rate, is varied between 10^3 *bps* and 10^8 *bps*. With Rate = 10^3 *bps* and 1500 bytes packet size, the traffic demand is far below the network capacity. When gradually varying Rate from 10^3 to 10^8 bps, offered load is increased from small to very large. The scheduling algorithm is LQF in Figure 3.13(a) and random selection in Figure 3.13(b). The algorithm for adapting p is Adapt in both figures. We again observe that in both figures, Token-DCF has almost the same throughput. The reason is that both LQF and fair selection give the token to a station with estimated non-zero queue length.



(a) LQF, Adapt

(b) Random selection, Adapt

Figure 3.13: ON/OFF traffic

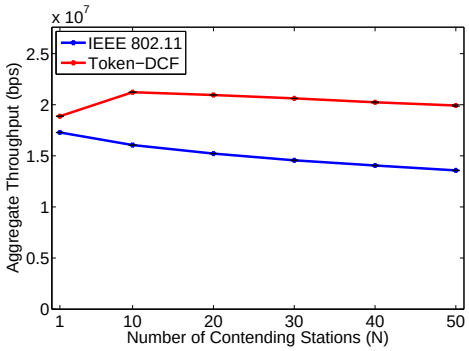
3.5.1 Simulating Performance in Different Topologies

In this section, we investigate the effect of changing topology on performance of Token-DCF. Throughput of Token-DCF and 802.11g for 5 different topologies of size $150m$x$150m$, is shown in Figure 3.14. Throughput of Token-DCF and 802.11g for 5 different topologies of size $1500m$x$1500m$, is shown in Figure 3.15. We note that a network of size $150m$x$150m$ is a single contention domain, while a network of size $1500m$x$1500m$ is a multi-hop network. Two-ray ground radio propagation model is assumed. Traffic type is TCP. The scheduling algorithm is LQF and the algorithm for adapting p is SMA. As we observe in Figures 3.14 and 3.15, throughput of Concurrent-MAC and 802.11 might be different in different topologies, but token-DCF achieves throughput improvement in all the considered topologies, by decreasing idle time and collision time of the wireless channel.

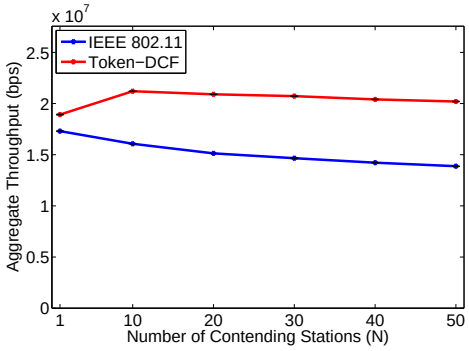
3.5.2 Simulating Performance in Indoor Environments

[36], [37], [38] have developed a path loss model for the wireless indoor channel. Their measurements and evaluations show that log-distance propagation model most closely matches the signal propagation in indoor environments. In log-distance path loss model, received power Pr (in dBm) at distance d from the transmitter is given by

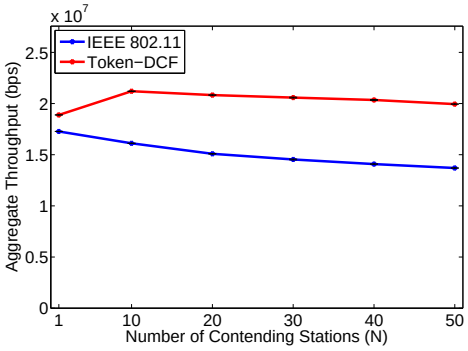
$$Pr(d) = Pr_0 - 10\alpha \log(d) + X_\sigma \tag{3.2}$$



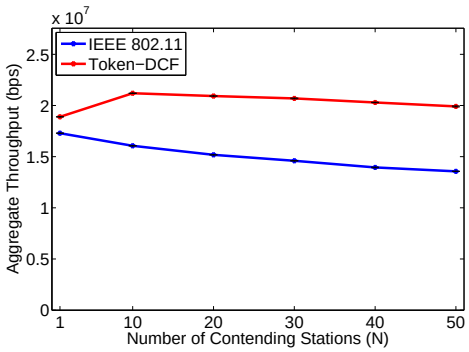
(a) Topology number 1



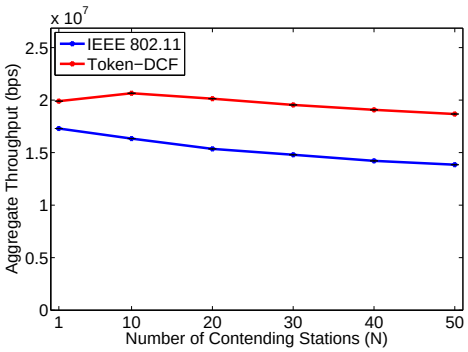
(b) Topology number 2



(c) Topology number 3



(d) Topology number 4

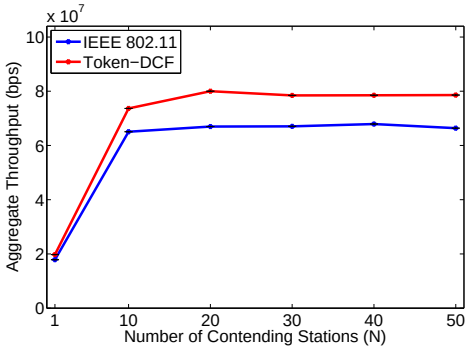


(e) Topology number 5

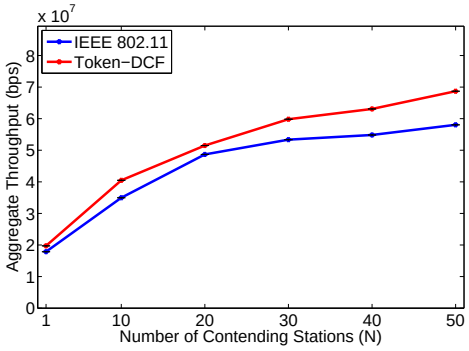
Figure 3.14: TCP traffic (area = 150 m x 150 m, LQF, SMA)

where Pr_0 is the signal power at 1 meter from the transmitter, α is the path loss exponent, and X_σ is a Gaussian random variable with mean of 0 and standard deviation of σ dB. $Pr(d)$ is the signal strength d meters from the transmitter.

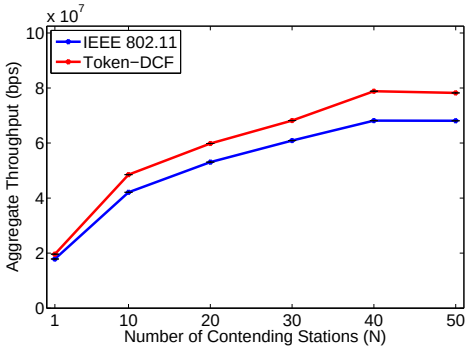
[36], [37], [38] have reported empirical values for path loss exponent α in the indoor environment to be in the interval $[1.8 - 5]$, where 1.8 corresponds to lightly obstructed environments, and 5 corresponds to multi-floored build-



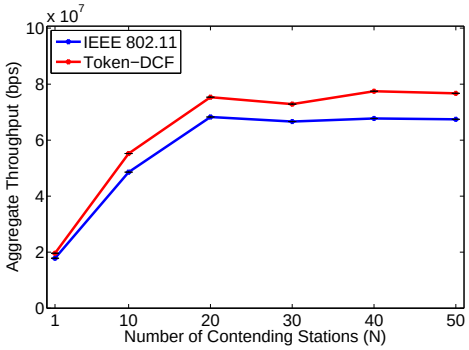
(a) Topology number 1



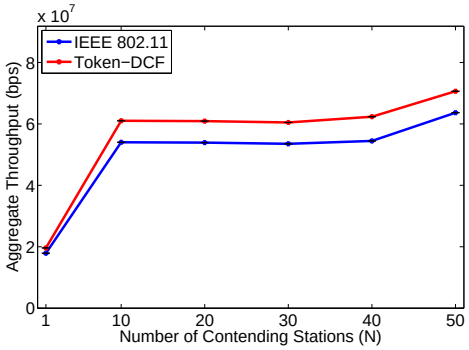
(b) Topology number 2



(c) Topology number 3



(d) Topology number 4

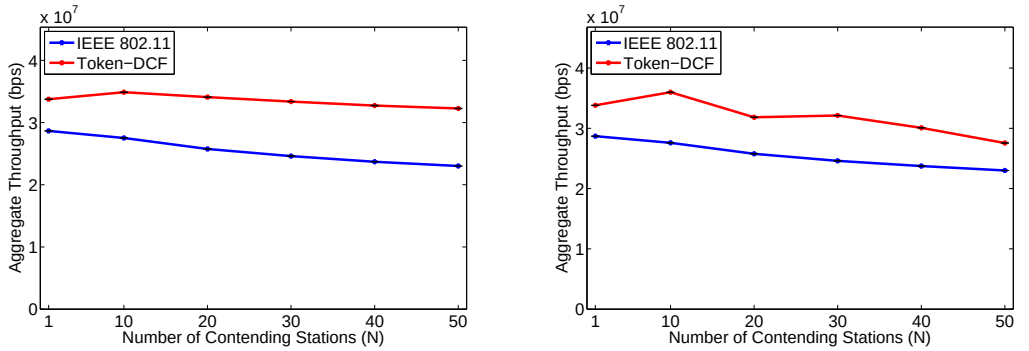


(e) Topology number 5

Figure 3.15: TCP traffic (area = 1500 m x 1500 m, LQF, SMA)

ings. Standard deviation σ is reported to be in the range of $[4-12]$ in indoor environment. In our simulations, we set the path loss exponent α to 4.02 and Standard deviation σ to 7.36 dB for indoor attenuation. These values are reported values in [38] for IEEE 802.11g hardware (operating at 2.4 GHz) in indoor environments.

The throughput of 802.11g and Token-DCF for two ray ground and indoor attenuation are shown in Figure 3.16. Traffic is full buffer CBR. Scheduling



(a) Two ray ground, area = 150 m x 150 m

(b) Indoor attenuation, area = 17 m x 17 m

Figure 3.16: Two different signal propagation models

algorithm is LQF and algorithm for adapting parameter p is Adapt. In Figure 3.16(a), signal propagation model is two ray ground and area size is 150 m x 150 m. In Figure 3.16(b), signal propagation model is indoor shadowing and area size is 17 m x 17 m. In both figures, based on our simulation parameters, the network is a single contention domain. Comparing Figure 3.16(a) and 3.16(b), we observe that the improvement obtained from Token-DCF is smaller for the case of indoor wireless channel. The reason is that signal attenuation is higher in indoor environment. As a result, when a station gives a token to one of its neighbors, the neighboring station might not receive the token, which means that network stations run the 802.11 back off mechanism to access the channel. Not receiving the assigned token, degrades the performance of Token-DCF.

3.6 Future Work

The Token-DCF protocol considers an ad hoc setting in which queue based scheduling algorithms are implemented. One possible future direction to explore will be to generalize the Token-DCF protocol to wireless LANs in which APs are present in the network. In wireless LANs, in most cases, the dominant traffic is downlink, which means that the APs should be given more chance for transmission compared to the client stations. Design of suitable scheduling algorithms for WLANs which might not necessarily be a queue based scheduling algorithm is an interesting future direction of this research.

Furthermore, all our evaluations of Token-DCF protocol were performed in ns-2 simulator. Implementing Token-DCF in a wireless testbed is another interesting future direction of this work.

3.7 Conclusion

In this chapter, we presented the design and performance evaluation of Token-DCF. Token-DCF is a distributed MAC protocol that uses an opportunistic overhearing mechanism to schedule network stations for transmission on the channel. The main design goal of Token-DCF is to reduce both idle time and collision time by introducing an implicit token passing algorithm. Our simulation results show that Token-DCF successfully decreases the overhead of 802.11 back off mechanism and, as a result, improves system throughput and channel access delay compared to 802.11 DCF.

CHAPTER 4

CONCURRENT-MAC: INCREASING CONCURRENT TRANSMISSIONS IN DENSE WIRELESS LANS

Deployment of wireless local area networks (WLANs) has grown rapidly in the past few years. IEEE 802.11 DCF is the MAC protocol commonly used in wireless LANs. 802.11 DCF employs *Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)* scheme. In 802.11 DCF protocol, a node willing to transmit senses the wireless channel to determine if the channel is busy or idle. If the channel is sensed busy, the node has to defer its transmission until the medium becomes idle. A node can only start transmission when the channel is idle. The main drawback of carrier sensing mechanism is that, the information regarding which node is transmitting on the channel is not considered. Some nodes might be eligible for concurrent transmissions while some might not, which is directly related to the SINR values at the receiver. Concurrent transmissions are transmissions that overlap in time. The ideal MAC protocol must

1. Prevent concurrent transmissions by interfering links. Otherwise, one or more of the transmissions will fail.

2. Allow concurrent transmissions by non-interfering links.

Maximizing the number of successful concurrent transmissions results in maximizing the aggregate throughput of wireless networks. Concurrent transmissions increase the spatial reuse of the wireless channel and improve network throughput.

All CSMA/CA based MAC protocols suffer from the well-known "hidden terminal" and "exposed terminal" problems. The hidden terminal problem refers to terminals that do not sense each other's transmission, although their concurrent transmission results in collision at their corresponding receivers, due to excessive interference. The exposed terminal problem occurs where a node refrains from transmission on sensing the channel busy, even though its transmission can succeed concurrently with the ongoing transmission. As

we will discuss in this chapter, increasing the density of stations and APs in WLANs increases the number of exposed terminals. A MAC protocol that identifies exposed terminals, and enables concurrent transmissions by exposed terminals, improves network throughput in dense WLANs.

IEEE 802.11 protocol is designed based on a "single AP-multiple stations" architecture, in which each AP serves multiple stations and each station is associated with only one access point at a time. But in current deployments of WLANs, we observe that, in many cases, multiple APs are present in the vicinity of each station. [8] reports that in most hotspots, $3 - 20$ APs are present within the transmission range of each client. As wireless APs become cheaper, dense WLANs are more commonly found. In dense WLANs, where a station is covered by multiple APs, due to the broadcast property of the wireless channel, packets sent by a station can be received by any of the APs present in the station's transmission range. The design of a MAC protocol that efficiently utilizes the presence of multiple APs to increase network throughput is an important problem in future deployments of WLANs.

In this chapter, we design a MAC protocol, called *Concurrent-MAC*, which identifies and enables concurrent transmissions to improve network throughput in dense WLANs. As we show in this chapter, in dense WLANs, there are many instances in which nearby nodes can transmit concurrently. Although the concurrent packets might collide in some of the APs, they can be received or captured successfully by some other APs. Our protocol, Concurrent-MAC, exploits the presence of the infrastructure to measure the path loss between network nodes. Based on path loss information, the central controller calculates the signal-to-interference-and-noise ratio (SINR) for different sets of transmitters to exactly find out which nodes can transmit concurrently. Each node is then given an accurate list of its concurrent neighbors. Whenever a node gains access to the channel, it gives a privilege to one of its neighboring exposed nodes to transmit concurrently. In dense WLANs, two nearby nodes might not be eligible for concurrent transmission if they are restricted to transmit to their associated APs. On the other hand, two nearby stations might be eligible for concurrent transmissions if their packets can be received by any AP. Our protocol, Concurrent-MAC, enables stations to transmit concurrently if their concurrent transmissions can be received or captured successfully by the backhaul of APs.

The rest of this chapter is organized as follows. We first review some related work in Section 4.1. In Section 4.2, we describe the architecture of the wireless LAN we are considering in this chapter. We then present our protocol, Concurrent-MAC, in Section 4.3. We compare the performance of Concurrent-MAC with IEEE 802.11a in Section 4.4 and finally present concluding remarks in Section 4.5.

4.1 Related Work

The problem of coordinating multiple APs in WLANs has received significant attention over the past decade. Miu et al. [39] have designed the *Multi-Radio Diversity (MRD)* wireless network, which uses path diversity to improve throughput of WLANs. Path diversity refers to the method of using two or more paths with different characteristics to decrease packet loss ratio and improve throughput. Path diversity is based on the fact that the packet is transmitted over several different propagation paths and individual paths experience different levels of fading and interference. Due to this property, packet losses are often statistically independent among different paths. MRD has proposed a *frame combining* method, which attempts to find the correct version of the transmitted frame even if it is received erroneously at all APs.

Zhu et al. [40] have proposed an AP association algorithm for deciding which APs to associate with. In uplink, their proposed heuristic selects an AP to be included in the set of associated APs of a station, if the addition of that AP will increase the throughput of the station by more than a threshold. In downlink, one of the associated APs is selected, according to the channel condition and the traffic load, to transmit data packets to the considered user. The AP selection algorithm in downlink tries to balance the traffic load among the neighboring APs. The objective of the AP selection algorithm in downlink is to minimize the delay, which is the sum of queueing delay and access delay.

Different solutions have been proposed in the literature to solve the hidden and/or exposed terminal problem of 802.11 DCF protocol [41], [42], [43], [44]. 802.11 protocol [41] defines an optional mechanism called RTS/CTS handshake to enhance virtual carrier sensing and reduce collisions caused by the hidden nodes. [42] builds upon the RTS/CTS mechanism to alleviate the

exposed terminals problem, although its proposed solution does not solve the exposed terminals problem completely.

Chen et al. [43] have proposed a self-learning collision avoidance protocol, called SELECT, to mitigate the hidden/exposed receiver problem in 802.11 wireless networks. SELECT is based on the assumption that the received signal strength (RSS) at the sender and receiver are highly correlated. SELECT builds a relationship between the measured RSS at the sender and the channel access success ratio. A station willing to transmit considers the associated success ratio with the current RSS, based on which, the station decides either to transmit or defer its transmission.

[44] proposes *CMAP (Conflict Maps)*, a MAC protocol for increasing the number of concurrent transmissions in wireless networks. CMAP's channel access method does not rely on carrier sense multiple access. Instead, CMAP tries to find out which stations can transmit concurrently and which cannot. In CMAP, initially, all stations transmit concurrently, even if their transmissions collide. Stations then measure the loss probability to figure out which nearby stations are interfering stations and which are exposed stations, based on which, the stations build conflicting transmissions map. A station willing to transmit on the channel considers the current transmitters and consults the conflict maps to decide whether to transmit or defer.

Our protocol, Concurrent-MAC, is based on the idea that increasing the density of stations and APs increases the number of exposed terminals in the network. Exploiting the presence of the infrastructure, Concurrent-MAC accurately identifies the nodes that can transmit concurrently, based on channel gain measurements and SINR computation. Each node is given an accurate list of its exposed neighbors. Whenever a node gains access to the channel, it gives a privilege to one of its exposed neighbors to transmit concurrently on the channel.

4.2 Dense WLAN Architecture

Figure 4.1 shows the dense WLAN architecture we consider in this chapter. Similar architecture has been proposed for dense WLANs in the literature [39], [40]. APs are connected to a central component called *controller* via a wired backbone. In our protocol, Concurrent-MAC, the stations are not

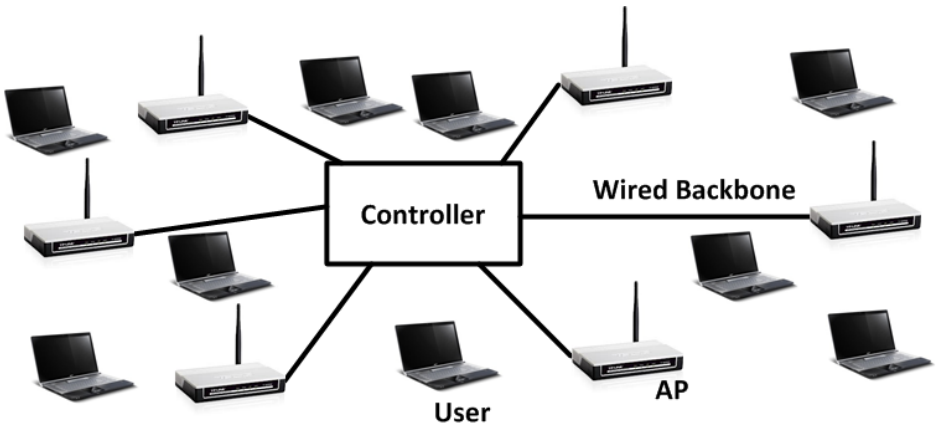


Figure 4.1: System architecture

explicitly associated with the APs and a station's packets might be received by any of the nearby APs. The APs are configured such that they promiscuously receive packets transmitted by close by stations. An AP forwards all the packets it thus receives to the controller, which filters redundant packets received by multiple APs and forwards only one copy to the higher network layers.

4.3 Concurrent-MAC Design

In this section, we first provide a high-level overview of Concurrent-MAC and then describe the details of the protocol.

4.3.1 Overview

Concurrent-MAC runs an opportunistic token passing protocol, where a token (or a privilege) is given by a transmitting node to one of its neighbors. In this chapter, we use the terms privilege and token interchangeably. A node transmitting on the channel selects one of its neighbors and gives it a privilege to transmit concurrently. The privileged neighbor starts transmission immediately, if it overhears the privilege and if it has packets to transmit. The transmitting node announces the privileged neighbor in the privileged field of the MAC header of the data packets it transmits. By overhearing data packets, a privileged node is informed that it can transmit concurrently.

If opportunistic overhearing does not work, i.e., privilege is not received by the privileged node, Concurrent-MAC operates similar to 802.11 DCF. But when the privileged node overhears the token, it can transmit concurrently. Signaling mechanism in Concurrent-MAC is done via embedding the information regarding queue length and privileged neighbor in the header of data packets by the source node and overhearing the packets to retrieve such information by the neighboring nodes. When a packet is transmitted, the concurrent neighbor and the queue length of the transmitter are embedded in the MAC header of the packet. Once a packet is received or overheard, the queue length of the source of the packet is retrieved. Furthermore, a neighboring node checks the privileged field to find out if it is privileged or not. In Concurrent-MAC, no extra control messages are transmitted to obtain the queue length information and to pass the privileges. Collecting the queue length information, giving privilege to the neighboring nodes and obtaining the privilege by the privileged nodes are all opportunistically done through overhearing.

Concurrent-MAC has two major components:

1. A probe phase to determine the sets of concurrent transmitters.
2. An opportunistic token passing protocol to enable concurrent transmissions.

4.3.2 Probe phase

In this phase, stations transmit a number of probe packets in a round robin manner. APs located in the reception range of each station receive the probe packets based on which the channel gain between stations and APs is measured. APs forward the channel gain information to the controller, based on which, the controller finds the sets of nodes that can transmit concurrently. A node might have more than one exposed neighbor. Each node is then given a list of concurrent neighbors, where a concurrent neighbor of a node can reliably transmit concurrently with the node. Nodes use concurrent neighbor information to schedule their neighbors for concurrent transmissions. Concurrent lists are computed periodically in order to prevent the stale information about the current channel conditions from degrading the protocol performance.

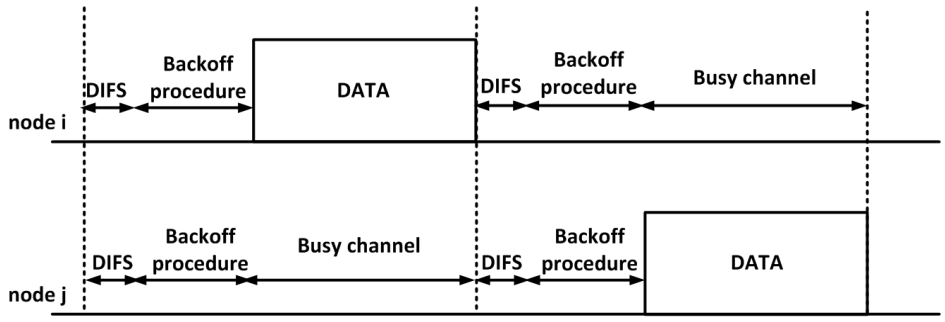


Figure 4.2: Access method of IEEE 802.11 DCF

4.3.3 Enabling concurrent transmissions

Concurrent-MAC enables concurrent transmissions by assigning privileges to network nodes. When a node transmits data packets, it gives a privilege to one of its neighbors to transmit concurrently. The scheduling component of Concurrent-MAC determines which neighbor is chosen as the privileged (or concurrent) node. The privileged node starts transmitting immediately after overhearing the token. Non-privileged nodes follow the carrier sensing mechanism of IEEE 802.11 DCF and defer transmission till the end of the NAV period.

Carrier sensing mechanism of 802.11 DCF is shown in Figure 4.2. 802.11 DCF defines two types of carrier sensing: physical carrier sensing and virtual carrier sensing. In physical carrier sense, if the total received power on the channel is greater than *carrier sensing threshold*, the channel is sensed as busy. On the other hand, if the total received power is below the carrier sense threshold, the channel is determined to be idle. The virtual carrier sensing mechanism of IEEE 802.11 is implemented using *Network Allocation Vector (NAV)*. The MAC header of each packet contains a *Duration* field that specifies the duration of transmitting the packet on the channel, during which the channel will be busy. The nodes listening on the channel read the *Duration* field and set their NAV to the duration of the current transmission, which determines how long a node must wait before attempting to transmit on the channel.

Channel access mechanism of Concurrent-MAC is shown in Figure 4.3. As explained before, each node is given a list its concurrent neighbors. In Concurrent-MAC, when node i transmits on the channel, it gives privilege

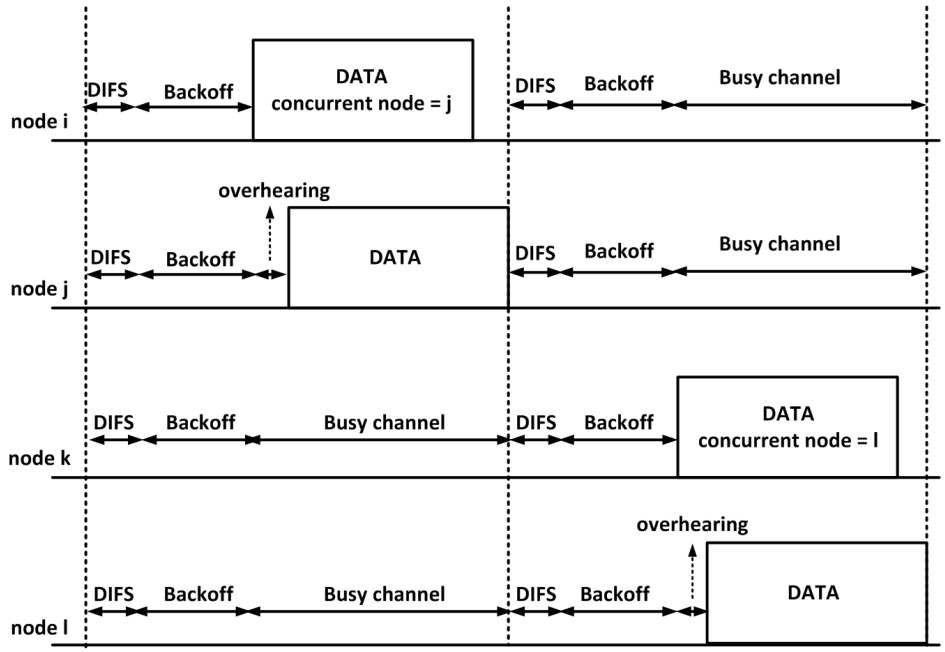


Figure 4.3: Access method of Concurrent-MAC protocol

to one of its concurrent neighbors, e.g. node j. If privileged node j was sensing the channel as idle before transmission of i starts and if it overhears the privilege given to it by node i, it will start transmitting on the channel immediately. Non-privileged nodes (e.g. node k in Figure 4.3) have to defer their transmission till when the channel becomes idle. This process of giving a privilege to a set of neighbors repeats in each transmission. Whenever a node transmits on the channel, a concurrent transmission might start. On the other hand, in IEEE 802.11 DCF protocol, when a node starts transmission, its nearby nodes do not transmit concurrently, since they sense the medium as busy, although their concurrent transmissions might be successfully received. Compared to the 802.11 DCF protocol in which nodes determine the channel as idle or busy based merely on some heuristic such as whether received power is more than a threshold, or nearby nodes are transmitting, Concurrent-MAC provides an accurate method for enabling concurrent transmissions which will result in successful reception.

The scheduling algorithm of Concurrent-MAC provides a mechanism for choosing a concurrent neighbor from all of its possible concurrent neighbors computed by the controller. Each node keeps track of queue length of its neighboring nodes. A node that gains access to the channel chooses one of

its concurrent neighbors with non-zero queue length (i.e., with backlogged traffic). If privilege is assigned to a node with an empty queue, the privileged node would not take its chance to transmit concurrently. This simply results in under-utilization of the underlying wireless channel. As long as the privilege is assigned to a node with backlogged traffic, the privileged node can immediately transmit a packet concurrently with the current transmitter.

Different scheduling algorithms can be used for choosing the privileged neighbor. Here, we present two example scheduling policies. In the first policy, a transmitting node chooses one of its concurrent neighbors uniformly at random. Another policy is to assign appropriate probabilities to different concurrent neighbors of each node such that each node has the same probability of being chosen as the concurrent neighbor. We note that a node might be concurrent neighbor of many nodes, while another node might be concurrent neighbor of only one node. In such a scenario, if concurrent neighbor is chosen uniformly at random, the probability of choosing each node, as the concurrent transmitter, is not uniformly distributed.

4.3.4 Probe Phase Details

The sets of nodes that can concurrently transmit with a node are determined by the controller during the probe phase. In this phase, each node sends few probe packets. Nodes transmit probe packets in a round robin manner. All nodes within the reception range of the transmitting node receive the probe packets. Comparing the transmit signal power of the packet transmitted by node i and the received signal power by node j, channel gain between i and j is calculated. We note that transmit power of a packet is included in the header of the packet. Channel gain information is collected by the set of stations and APs and is given to the central controller. Having channel gain information, the central controller calculates the set of the nodes that can transmit concurrently such that their concurrent transmission can be received or captured successfully.

Knowing the channel gain between nodes, the central controller considers the *signal-to-interference-and-noise ratio (SINR)* for different combinations of nodes to find out if their concurrent transmission can be received or captured by different APs. The details of finding the concurrent neighbor of each

18 Finding Concurrent Pairs

1: **for** $i_1 = 0$ to `numUsers` **do**
2: $\quad$ $concurrentList_{i1} = \{\}$
3: $\quad$ **for** $i_2 = 0$ to `numUsers` **do**
4: $\quad\quad$ **if** i_1 and i_2 are in the carrier sense range of each other **then**
5: $\quad\quad\quad$ $SNR_{i_1 i_2} = \frac{P_t g_{i_1 i_2}}{N}$
6: $\quad\quad\quad$ **if** $SNR_{i_1 i_2} \geq \texttt{thresh}_0$ **then**
7: $\quad\quad\quad\quad$ **for** $k_1 = 0$ to `numAPs` **do**
8: $\quad\quad\quad\quad\quad$ $SINR_{i_1 k_1} = \frac{P_t g_{i_1 k_1}}{P_t g_{i_2 k_1} + N}$
9: $\quad\quad\quad\quad\quad$ **if** $SINR_{i_1 k_1} \geq \texttt{thresh}_{i_1}$ **then**
10: $\quad\quad\quad\quad\quad\quad$ **for** $k_2 = 0$ to `numAPs` **do**
11: $\quad\quad\quad\quad\quad\quad\quad$ $SNR_{i_1 k_2} = \frac{P_t g_{i_1 k_2}}{N}$
12: $\quad\quad\quad\quad\quad\quad\quad$ $SINR_{i_1 k_2} = \frac{P_t g_{i_1 k_2}}{P_t g_{i_2 k_2} + N}$
13: $\quad\quad\quad\quad\quad\quad\quad$ $SINR_{i_2 k_2} = \frac{P_t g_{i_2 k_2}}{P_t g_{i_1 k_2} + N}$
14: $\quad\quad\quad\quad\quad\quad\quad$ **if** $SNR_{i_1 k_2} < \texttt{thresh}_0$ and $SINR_{i_2 k_2} \geq \texttt{thresh}_{i_2}$ **then**
15: $\quad\quad\quad\quad\quad\quad\quad\quad$ Add $\{(i_1, r_1), (i_2, r_2)\}$ to $concurrentList_{i1}$
16: $\quad\quad\quad\quad\quad\quad\quad$ **else if** $SNR_{i_1 k_2} \geq \texttt{thresh}_0$ and $SINR_{i_1 k_2} < \texttt{thresh}_{i_1}$ and $SINR_{i_2 k_2} \geq \texttt{captureThresh}_{i_2}$ **then**
17: $\quad\quad\quad\quad\quad\quad\quad\quad$ Add $\{(i_1, r_1), (i_2, r_2)\}$ to $concurrentList_{i1}$
18:

station is given in Procedure 18. In this procedure, `numUsers` denotes the total number of users and `numAPs` denotes the total number of APs. If two stations i_1 and i_2 are located in the carrier sense range of each other, they might be eligible for concurrent transmissions in our protocol, Concurrent-MAC, although 802.11 protocol does not allow them to transmit concurrently. From all stations i_2 in the carrier sense range of i_1, the ones that can successfully receive the header of a packet transmitted by station i_1 are able to receive a privilege given to them by i_1. We note that the information regarding which station is the privileged station for concurrent transmission is included in the MAC header of the packet. This means that a necessary condition for being able to transmit concurrently with station i_1 is

$$SNR_{i_1 i_2} \geq \texttt{thresh}_0 \tag{4.1}$$

where

$$SNR_{i_1 i_2} = \frac{P_t g_{i_1 i_2}}{N} \tag{4.2}$$

In the above equations, P_t is the transmit power, $g_{i_1 i_2}$ is the channel gain between i_1 and i_2, N is the noise floor and $SNR_{i_1 i_2}$ is the signal-to-noise ratio of a packet transmitted by station i_1 and received by station i_2. $\texttt{thresh}_0$ denotes the SINR threshold required to successfully receive the header of a packet. For station i_2, it is enough to receive the header of the packet transmitted by station i successfully, to find out if station i_2 has the privilege for transmitting concurrently or not.

If station i_2 can successfully receive the token transmitted by station i_1, the SINR values at APs are considered to determine if two different APs k_1 and k_2 can receive the concurrent transmission of i_1 and i_2, respectively. Packet of i_1 can be successfully received by an AP k_1, if

$$SINR_{i_1 k_1} = \frac{P_t g_{i_1 k_1}}{P_t g_{i_2 k_1} + N} \geq \texttt{thresh}_{i_1} \tag{4.3}$$

where $\texttt{thresh}_{i_1}$ denotes the SINR threshold required to reliably receive packet of station i_1. We note that SINR threshold might be different for different stations, since stations might transmit at different rates which requires different SINR thresholds. Equation 4.3 means that the packet of i_1 can be received by AP k_1, with high enough SINR, in the face of the interference caused by station i_2.

In the ns-2 model, packet of station i_2 can be successfully received by an AP k_2 in two situations:

1. If

$$SNR_{i_1 k_2} = \frac{P_t g_{i_1 k_2}}{N} < \texttt{thresh}_0 \tag{4.4}$$

and

$$SINR_{i_2 k_2} = \frac{P_t g_{i_2 k_2}}{P_t g_{i_1 k_2} + N} \geq \texttt{thresh}_{i_2} \tag{4.5}$$

Equation 4.4 means that k_2 does not start reception of the packet transmitted by i_1, since the packet header is received with SNR below the threshold required for reliable header reception, $\texttt{thresh}_0$. Equation 4.5 means that the packet of i_2 can be received by k_2, with high enough SINR, in the face of the interference caused by i_1.

2. If

$$SNR_{i_1k_2} = \frac{P_t g_{i_1k_2}}{N} \geq \texttt{thresh}_0 \tag{4.6}$$

and

$$SINR_{i_1k_2} = \frac{P_t g_{i_1k_2}}{P_t g_{i_2k_2} + N} \leq \texttt{thresh}_{i_1} \tag{4.7}$$

and

$$SINR_{i_2k_2} = \frac{P_t g_{i_2k_2}}{P_t g_{i_1k_2} + N} \geq \texttt{captureThresh}_{i_2} \tag{4.8}$$

Equation 4.6 means that k_2 starts reception of the header of i_1's packet, since the packet header is received with SNR more than the required threshold for successful reception of packet header, $\texttt{thresh}_0$. Equation 4.7 means that the packet of i_1 cannot be successfully received by k_2, when both stations i_1 and i_2 are transmitting, since packet of i_1 is received with SINR below the threshold $\texttt{thresh}_{i_1}$. In this case, packet of i_1 is dropped at k_2. At this point, k_2 is able to capture packet of i_2, if Equation 4.8 holds, meaning that if $SINR_{i_2k_2}$ is more than $\texttt{captureThresh}_{i_2}$, which is the capture threshold required for capturing the packet of i_2.

At this point, $\{(i_1, r_1), (i_2, r_2)\}$ is added to $\textit{concurrentList}_{i1}$. r_1 and r_2 denote the transmission rate of stations i_1 and i_2, respectively. Including rate information is redundant in this procedure. Since later we will present a similar procedure for finding the concurrent transmitters at different rates, in order to have a consistent presentation, we include the rate information in Procedure 18 as well. At the end of Procedure 18, $\textit{concurrentList}_{i1}$ includes all neighboring stations i_2 that can concurrently transmit with station i_1.

We note that Procedure 18 does not consider transmitting acknowledgements by APs in its computations. In Concurrent-MAC, we do not transmit the acknowledgement to nodes i_1 and i_2 concurrently. The reason is that transmission of acknowledgements concurrently might result in collision. Instead, we transmit the acknowledgements sequentially. In this method, first the acknowledgment to station i_1 is transmitted and then, when transmission of acknowledgment to station i_1 is finished, the acknowledgement to station i_2 is transmitted. The delay in transmitting MAC acknowledgment is included in the packet header transmitted by stations i_1 and i_2 to inform their corresponding receivers about the delay in transmitting the acknowl-

edgement. The acknowledgement from AP k_1 to station i_1 is delayed by a time duration equal to the transmission time of packet header. The acknowledgement from AP k_2 to station i_2 is delayed by a time duration equal to transmission time of one MAC acknowledgement packet plus SIFS time. The delay for transmitting MAC acknowledgment is also added to *duration* field of the packets transmitted by stations i_1 and i_2. We note that the MAC layer frame headers contain a *duration* field that specifies the transmission time required for the frame, in which time the medium will be busy. The neighboring stations listening on the wireless channel read the *duration* field and set their NAV, which is an indicator for a station on how long it must defer from accessing the channel.

Although Procedure 18 presents the algorithm for finding the concurrent stations, it is not restricted to uplink transmissions only. Similar computation can be done to find sets of concurrent APs. Sets of concurrent APs can be found out by considering all APs, instead of all users, in lines 1 and 3 of Procedure 18, and also considering all users, instead of all APs, in lines 7 and 10 of Procedure 18. The rest of the computation is the same as Procedure 18.

4.3.5 Protocol Details

The detailed Concurrent-MAC protocol is presented in this section. Each node maintains the following local variables:

- *flag*: a Boolean variable denoting if the node has a privilege for transmitting concurrently on the channel or not. *flag* equals to `true` means that the node is allowed to transmit concurrently with the current transmitter. *flag* is initially set to `false`, meaning that no node is privileged initially.

- *concurrentList*: maintains the list of the neighboring nodes which can transmit concurrently with this node.

- *qLen*: maintains the queue length of the neighboring nodes.

Two fields are added to the MAC header of the packets:

- *concurrentNode*: denotes the node that can transmit concurrently with this packet.

- *qLength*: denotes the current queue length of the source of the packet.

Procedure 19 is executed right before a data packet is transmitted on the channel. If *flag* of the node equals to `false`, the node gives a privilege to one of the nodes in its *concurrentList*. A node can choose one of its neighbors as *concurrentNode*, only if its *flag* is equal to `false`, meaning that the node has gained access to the channel via the backoff mechanism and not via obtaining privilege from another node. A privileged node (i.e., a node with *flag* equal to `true`) is not allowed to assign privilege to other nodes, since, due to excessive interference, the transmission by the union of the privileged nodes might not result in successful reception.

19 Transmitting a packet

1: **if** *flag* == `false` **then**
2: *concurrentNode* = one of the nodes with *qLen* > 0 chosen uniformly at random from *concurrentList* .

Procedure 20 is called when a packet is overheard. Since the wireless channel is a shared medium, node i might overhear packets that are not intended for it, i.e., packets with destination address different from i. If the overhearing node is chosen as the *concurrentNode*, it sets its *flag* to `true`. At this point, the *concurrentNode* cancels its backoff counter, if its backoff counter is running or paused, and immediately transmits on the channel, if it has any backlogged traffic. Since nodes transmit at different rates, or they might be transmitting packets with different sizes, transmission time of one packet might be different for different nodes. From the time that *concurrentNode* sets its *flag* to `true`, it can transmit data packets for a duration of *txtime*, where *txtime* is the duration of one packet transmission by *src*. During this time, *concurrentNode* is allowed to transmit as many packets as it can. If one packet transmission by *concurrentNode* lasts longer than *txtime*, *concurrentNode* is not allowed to transmit concurrently with *src*.

Whenever the medium becomes idle, all nodes set their *flag* to *false*. In this way, a privileged node has the privilege to transmit concurrently only

20 Overhearing a packet from *src*

```
1: qLen[src] = qLength
2: if myId == concurrentNode then
3:     flag = true
4:     if backoff counter is running or paused then
5:         cancel backoff counter
6:         if backlog queue is not empty then
7:             transmit packets for a duration of txtime
8: else
9:     flag = false
10:    schedule NAV timer for the duration of the current transmission
```

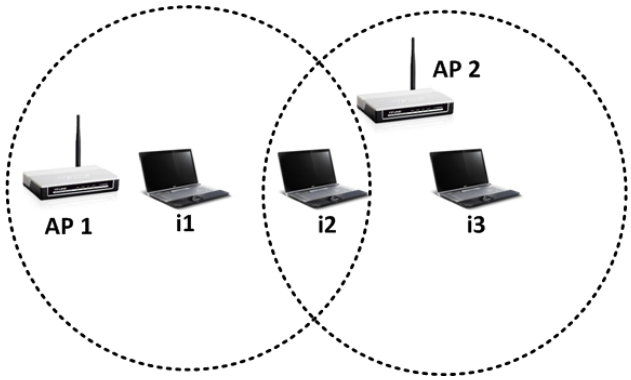


Figure 4.4: Example network showing that concurrent transmissions at lower rates in Concurrent-MAC is not better than 802.11

during the current transmission. Whenever the transmission finishes, the node does not have the privilege for concurrent transmission anymore.

4.3.6 Decreasing Transmission Rate to Increase Concurrent Transmissions

As we have observed in our simulations, decreasing transmission rate of stations might result in increasing the number of concurrent pairs in the network. The reason is that by decreasing transmission rate, the SINR threshold for reliable data reception decreases. In this case, nearby stations can transmit concurrently, at a lower rate, since their concurrent transmissions are received with SINR higher than the SINR threshold by the backhaul of APs. This means that some sets of stations that cannot transmit concurrently at their highest transmission rate, can transmit concurrently at lower rates.

In single-hop networks, if concurrent transmission at highest transmission rate of stations is not possible, and if the throughput of the stations transmitting concurrently, at lower rates, is more than the throughput of only one station transmitting at its maximum possible rate, concurrent transmissions at lower rates will increase aggregate throughput compared to 802.11 protocol. On the other hand, in multi-hop networks, decreasing transmission rate of a station to enable concurrent transmissions by its neighboring stations does not result in improving network throughput compared to 802.11 protocol. We explain why this happens in an example network shown in Figure 4.4. As shown in this figure, stations i_1 and i_2 are in the carrier sense range of each other. Station i_3 is in the carrier sense range of station i_2, but not in the carrier sense range of station i_1. We assume that all stations i_1, i_2 and i_3 can transmit at rate 54 Mbps individually. We assume stations i_1 and i_2 can transmit concurrently at rate 36 Mbps, but not at higher rates, such as 54 Mbps. We also assume that, considering network parameters, transmission rate of 54 Mbps results in throughput of 29.3 Mbps and transmission rate of 36 Mbps results in throughput of 22 Mbps. If stations i_1 and i_2 transmit concurrently, network throughput will be equal to 44 Mbps. We note that when station i_2 is transmitting, station i_3 cannot transmit, since it senses the channel as busy. On the other hand, in 802.11 protocol, stations i_1 and i_3 can transmit with rate 54 Mbps, since they are located far enough in space, which results in network throughput of 58.6 Mbps. This configuration demonstrates the trade-off between decreasing rate and increasing concurrent transmissions and explains why transmitting concurrently, at lower rates, is not necessarily better than transmitting not concurrently, with the highest possible rate.

The computations to find concurrent pairs transmitting at different transmission rates is slightly different from Procedure 18 and is presented in Procedure 21. The difference is that when multiple rates are available, for each transmitting station i_1, we consider station i_1 and its neighbors transmitting at different rates, with the corresponding SINR thresholds. Similar to Procedure 18, in Procedure 21, we consider stations i_1 and i_2, such that station i_2 is able to successfully receive packet header from station i_1 (Line 6 of Procedure 21), which means that station i_2 is able to receive the token given to it by station i_1. Then, as shown in Line 8 of Procedure 21, we consider all transmission rates r_1 between $\texttt{maxRate}_{i_1}$, and $\texttt{minRate}$. $\texttt{maxRate}_{i_1}$ denotes the

21 Finding Concurrent Pairs at Lower Transmission Rates

```
1:  for i_1 = 0 to numUsers do
2:      concurrentList_{i1} = {}
3:      for i_2 = 0 to numUsers do
4:          if i_1 and i_2 are in the carrier sense range of each other then
5:              SNR_{i1 i2} = P_t g_{i1 i2} / N
6:              if SNR_{i1 i2} ≥ thresh_0 then
7:                  for k_1 = 0 to numAPs do
8:                      for r_1 = maxRate_{i1} to minRate do
9:                          SINR_{i1 k1} = P_t g_{i1 k1} / (P_t g_{i2 k1} + N)
10:                         if SINR_{i1 k1} ≥ thresh_{r1} then
11:                             for k_2 = 0 to numAPs do
12:                                 for r_2 = maxRate_{i2} to minRate do
13:                                     SNR_{i1 k2} = P_t g_{i1 k2} / N
14:                                     SINR_{i1 k2} = P_t g_{i1 k2} / (P_t g_{i2 k2} + N)
15:                                     SINR_{i2 k2} = P_t g_{i2 k2} / (P_t g_{i1 k2} + N)
16:                                     if SNR_{i1 k2} < thresh_0 and
                                        SINR_{i2 k2} ≥ thresh_{r2} then
17:                                         Add {(i_1, r_1), (i_2, r_2)} to
                                            concurrentList_{i1}
18:                                     else if SNR_{i1 k2} > thresh_0
                                        and SINR_{i1 k2} ≤ thresh_{r1} and
                                        SINR_{i2 k2} > captureThresh_{r2}
                                        then
19:                                         Add {(i_1, r_1), (i_2, r_2)} to
                                            concurrentList_{i1}
20:
```

maximum rate with which station i_1 can transmit. `minRate` is an input parameter of the procedure and denotes the minimum rate with which a station is allowed to transmit. Furthermore, $\texttt{thresh}_{r_1}$ denotes the SINR threshold for successful reception corresponding to transmission rate of r_1. If any AP k_1 is able to receive packet of i_1 with SINR more than $\texttt{thresh}_{r_1}$, we then consider each neighbor of i_1, called i_2, where station i_2 transmits with rate r_2. r_2 can be any possible transmission rate between $\texttt{maxRate}_{i_2}$ and `minRate` (Line 12 of Procedure 21). SINR threshold corresponding to transmission rate r_2 is called $\texttt{thresh}_{r_2}$. Capture threshold corresponding to transmission rate r_2 is called $\texttt{captureThresh}_{r_2}$. If any AP k_2 is able to receive or capture packet of i_2, transmitted at rate r_2, we conclude that stations i_1 and i_2 can

transmit concurrently, with transmission rates of r_1 and r_2, respectively. We note that conditions for successfully receiving or capturing packet of i_2 (i.e., Lines 16 and 18 of Procedure 21) is similar to the conditions presented and explained before for Procedure 18 (i.e., Lines 14 and 16 of Procedure 18).

We note that stations i_1 and i_2 might be eligible for concurrent transmission at rates r_1 and r_2. On the other hand, stations i_1 and a third station i_3 might be eligible for concurrent transmission with other rates, such as r_3 and r_4. This means that the transmission rate of a station is not fixed; instead, it depends on the chosen concurrent pair. The other difference is that, when a station i_1 gives a privilege to another station i_2, it chooses its rate to be equal to the rate that is feasible for the chosen concurrent transmissions. The transmission rate of the privileged station, ***privilegedRate***, is included in the MAC header and the privileged station is only allowed to transmit with rate ***privilegedRate***.

At the end of the probe phase, the concurrent sets with transmission rates corresponding to each set are determined. The procedure for choosing the concurrent neighbor by a transmitting station i_1, when multiple rates are available, is slightly different from Procedure 19 and is presented in Procedure 22. In this procedure, $\texttt{thr}_1$ denotes the throughput corresponding to transmission rate of r_1, $\texttt{thr}_2$ denotes the throughput corresponding to transmission rate of r_2 and $\texttt{maxThr}_{i_1}$ denotes the throughput corresponding to transmission rate of $\texttt{maxRate}_{i_1}$, which is the maximum rate with which station i_1 can transmit. Since the goal of Concurrent-MAC is to increase network throughput, among all feasible $\{(i_1, r_1), (i_2, r_2)\}$ pairs in $\textit{concurrentList}_{i1}$, i_1 chooses the pair $\{(i_1, r_1), (i_2, r_2)\}$ which has the maximum sum throughput $\texttt{thr}_1$+$\texttt{thr}_2$, for concurrent transmissions. It is also required that $\texttt{thr}_1$+$\texttt{thr}_2 > \texttt{maxThr}_{i_1}$, since only under this condition, transmitting concurrently achieves higher throughput than transmitting individually.

22 Transmitting a packet by station i_1

1: **if** *flag* == `false` **then**
2: (*concurrentStation*,*privilegedRate*) = (i_2, r_2) for which $\textit{qLen}_{i_2} > 0$ and $r_1 \leq r_2$ and $\{(i_1, r_1), (i_2, r_2)\}$ has the highest $\texttt{thr}_1$+$\texttt{thr}_2$ in $\textit{concurrentList}_{i1}$ and $\texttt{thr}_1$+$\texttt{thr}_2 > \texttt{maxThr}_{i_1}$

4.3.7 Heuristic for choosing `minRate`

As we explained before, decreasing transmission rate might increase network throughput. Different transmission rates result in different number of concurrent transmissions, and consequently, network throughput. The question to ask is how to choose the transmission rate to increase network throughput. As we discussed in Section 4.3.6, decreasing rate might not always result in increased network throughput. Because of this, transmission rate of stations should be chosen very carefully.

We have designed a simple heuristic for choosing `minRate`. We note that `minRate` is the minimum transmission rate network stations can transmit with. Our heuristic is shown in Procedure 23. `minRate` is an input parameter of Concurrent-MAC, and network stations are not allowed to transmit with a rate smaller than `minRate`. In our heuristic, we consider all `minRate` values between `MINIMUMRATE` and `MAXIMUMRATE`, where `MINIMUMRATE` is the minimum rate a station can transmit with and `MAXIMUMRATE` is the maximum rate a station can transmit with. For example, in 802.11a protocol, `MINIMUMRATE` is 6 Mbps and `MAXIMUMRATE` is 54 Mbps. We then run Procedure 21 and count the number of stations with concurrent neighbors, i.e., the number of stations that can give privilege to other stations, for concurrent transmissions. We then choose the `minRate` value which results in maximum number of stations having a concurrent neighbor. We will evaluate the performance of this heuristic, later via simulations.

23 Heuristic For Choosing `minRate`

1: **for** `minRate` = `MINIMUMRATE` to `MAXIMUMRATE` **do**
2: Find number of stations with concurrent neighbors from Procedure 21
3: Choose `minRate` which results in maximum number of stations having concurrent neighbors

4.4 Evaluation

We simulate Concurrent-MAC and 802.11a in ns-2 to measure and compare performance of these two MAC protocols. Table 4.1 reports the configuration parameter values of the wireless network analyzed in this section. Transmis-

Table 4.1: Parameters used in the simulation study

Propagation	Shadowing
RTS/CTS	disabled
Transmit power	15 dBm
Thermal Noise	-93 dBm
Carrier sense threshold	-91dBm
SIFS	16μsec
DIFS	34 μsec
Slot time	9μsec
CWmin	15
CWmax	1023

sion options for IEEE 802.11a radios are reported in Table 4.2. Similar values are reported in [34], [35], [45]. SINR thresholds for reliable data reception and capture thresholds reported in Table 4.2 are chosen from [34]. SINR thresholds of Table 4.2 are the same as *Sender First (SF) capture thresholds* reported in [34]. Capture thresholds of Table 4.2 are the same as *Sender Last (SL) capture thresholds* reported in [34].

The network is a wireless LAN in which nodes are placed uniformly at random in a square area. We run the simulations for different network sizes. IEEE 802.11 RTS/CTS mechanism is turned off. We use a log-distance path loss model with path loss exponent of 4.02 to simulate the indoor office environment [38]. Transmission range corresponding to each data rate in our simulations is reported in Table 4.3. With our simulation parameters, carrier sense range is 25 m. Packet payload size is 1500 bytes. Each simulation lasts for 30 seconds and the presented results are averaged over 5 runs. As we have observed in the simulations of this section, standard deviation of the measured throughput is too low and because of that, we have not plotted standard deviations in the figures of this section.

4.4.1 Performance Evaluation in Example Networks

We first study an example network with two stations and two APs placed in an area of 15 m x 15 m, as shown in Figure 4.5. Traffic is uplink, meaning

Table 4.2: IEEE 802.11a transmission options

Data rate (Mbps)	SINR threshold for reliable data reception (dB)	Capture threshold (dB)	Modulation	Coding rate
6	0.5	10	BPSK	1/2
9	3	10	BPSK	3/4
12	3.5	10	QPSK	1/2
18	7	10	QPSK	3/4
24	13	14	16-QAM	1/2
36	17	18	16-QAM	3/4
48	21	22	64-QAM	2/3
54	23	26	64-QAM	3/4

Table 4.3: Transmission ranges in our simulations

Data rate (Mbps)	range for reliable data reception (m)
6	32
9	28.4
12	27.6
18	22.6
24	16
36	12.7
48	10
54	8.9

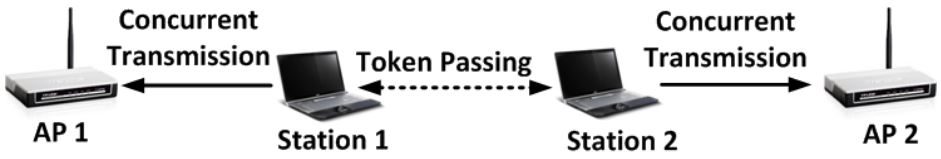


Figure 4.5: Stations 1 and 2 transmit concurrently

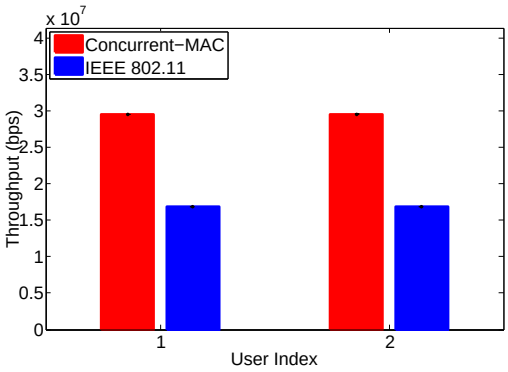


Figure 4.6: Throughput (2 stations in an example network)

that stations transmit data packets to APs. All flows are always backlogged. The goal of this example is to demonstrate how Concurrent-MAC successfully exploits the possibility of concurrent transmissions to improve the throughput. Concurrent-MAC is designed based on the idea that the nearby stations can transmit concurrently, although they sense each other, in case there are APs present in the area that can receive/capture their concurrent transmission reliably. The two stations are placed at the distance of $8m$ from each other. The two APs are placed at the distance of 2 m from the stations as shown in Figure 4.5. Individual throughput of the stations in 802.11 and Concurrent-MAC are shown in Figure 4.6. Under our simulation parameters, the carrier sense range is 25 m. Since the two stations are in the carrier sense range of each other, at each time instance only one of them transmits in IEEE 802.11 protocol. On the other hand, our protocol, Concurrent-MAC, enables concurrent transmission of the two stations, which results in increasing the throughput. In Concurrent-MAC, in this example network, any station that gains access to the channel gives a privilege for concurrent transmission to the other station. In this case, both stations transmit concurrently on the channel. We note that two stations can transmit concurrently on the channel

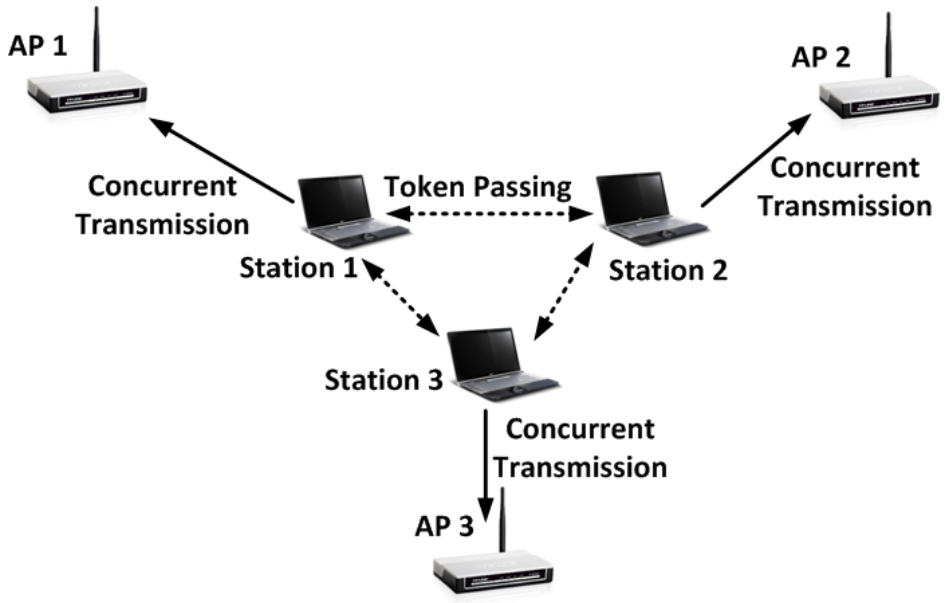


Figure 4.7: 3 Stations transmit concurrently

if the following two conditions are met:

1. The stations should be close enough that they can receive privilege from each other. This distance should be no more than the transmission range corresponding to the rate with which the packet header is sent (32 m under our simulation parameters).

2. Two APs should be present in the area and be positioned such that the concurrent transmission of the stations can be received reliably, i.e., with high enough SINR, by them. An example placement of APs is shown in Figure 4.5.

A placement with three stations and three APs in which all three stations can transmit concurrently is shown in Figure 4.7. The individual throughput of stations in this network is shown in Figure 4.8. As we expected, Concurrent-MAC achieves $3X$ improvement over 802.11. The reason is that although the stations are all in the carrier sense range of each other, since their concurrent transmissions can be received by the three APs with high enough SINR, they are able to transmit concurrently. This example shows that for networks with static stations, where stations location is fixed, the APs can be placed close enough to the stations and far from each other, in order to enable concurrent transmissions and to improve throughput.

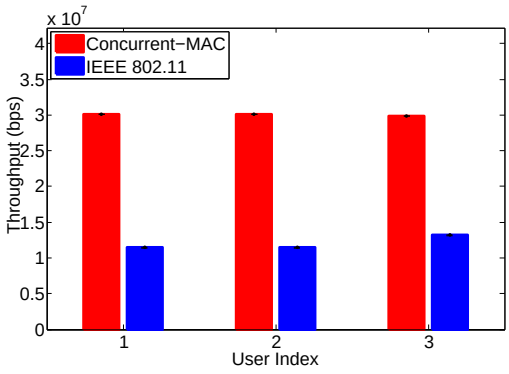


Figure 4.8: Throughput (3 stations in an example network)

4.4.2 Performance Evaluation in Saturated Single Contention Domain

We now simulate our protocol, Concurrent-MAC, and IEEE 802.11a in single contention domain with random placement of stations and APs to demonstrate the performance of Concurrent-MAC and how aggregate throughput can be improved by increasing number of APs. In a single contention domain, every node is in carrier sense range of every other node and at each time instance, at most one node can transmit in 802.11 protocol. In this section, we describe how Concurrent-MAC exploits the presence of multiple APs, to identify the nodes that can transmit concurrently, although they are in the carrier sense range of each other, if their concurrent transmissions can be received successfully by the backhaul of APs. We place 10 nodes in a square area of 17 m x 17 m. With our simulation parameters, in this area size, every node senses every other transmission and only one transmission is possible in IEEE 802.11 protocol, at each time instance. We note that carrier sense range in our simulations is 25 m. We vary the number of APs from 10 to 90 to investigate the effect of increasing number of APs. Traffic is full buffer CBR, meaning that there is always backlogged traffic in the transmission queue of each transmitter. Transmission queue of each node holds up to 50 packets and when the buffer is full, newly arrived packets get dropped.

For this network, aggregate throughput of Concurrent-MAC and 802.11 is shown in Figure 4.9. As we see in this figure, the throughput is increased by 15%, when we have 10 APs, and is increased by 40% - 44% with 50 - 90

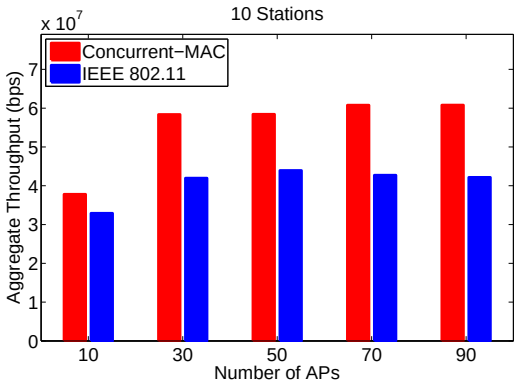


Figure 4.9: Topology number 1 (10 stations, area = 17 m x 17 m)

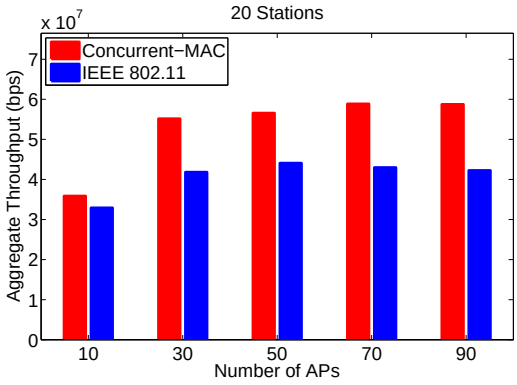


Figure 4.10: Topology number 1 (20 stations, area = 17 m x 17 m)

APs. Considering the transmitting stations and their concurrent neighbors, we found out that, for this topology, with 90 APs present in the area, 8 out of 10 stations can give privilege for concurrent transmission to other stations. We picked one concurrent neighbor for each of these 8 stations and counted the number of APs needed to successfully receive all of these concurrent transmissions. We found out that, from all the 90 APs present in the network, only 11 APs are sufficient to successfully receive all concurrent transmissions and to achieve the maximum throughput in Concurrent-MAC, for this topology. The reason is that the 90 APs are placed randomly in the area, and not all of them are required for successful reception of all concurrent transmissions.

We then place 20 stations in the area of 17 m x 17 m. We increase number of APs from 10 to 90. The aggregate throughput of Concurrent-MAC

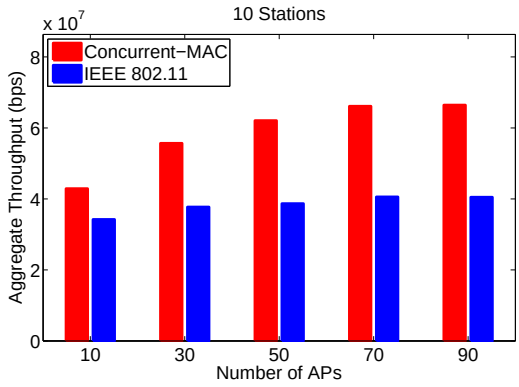


Figure 4.11: Topology number 2 (10 stations, area = 17 m x 17 m)

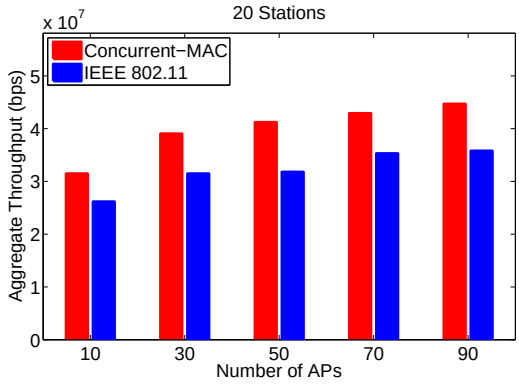


Figure 4.12: Topology number 2 (20 stations, area = 17 m x 17 m)

and 802.11a is shown in Figure 4.10. We observe that, with 20 stations, throughput gain obtained by Concurrent-MAC compared to IEEE 802.11a is between 9% and 39%. In this topology, 19 APs are sufficient to achieve maximum concurrency.

Throughput of Concurrent-MAC and 802.11a for 4 other topologies, with 10 and 20 stations and 10 - 90 APs placed uniformly at random in a square area of 17 m x 17 m is shown in Figures 4.11 - 4.18. As we observe in these figures, throughput improvement achieved by Concurrent-MAC is different for different topologies. The reason is that network topology determines the number of concurrent transmitters in the network, and throughput of Concurrent-MAC is a function of number of concurrent transmitters in the network.

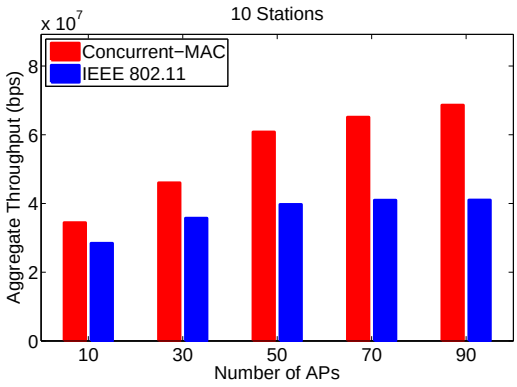


Figure 4.13: Topology number 3 (10 stations, area = 17 m x 17 m)

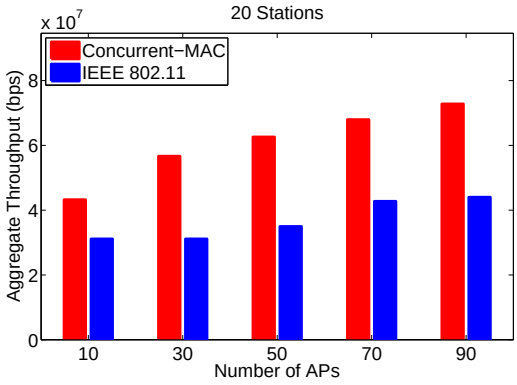


Figure 4.14: Topology number 3 (20 stations, area = 17 m x 17 m)

4.4.3 Performance Evaluation in Saturated Multi-Hop Networks

Throughput of Concurrent-MAC and 802.11a for 5 different topologies, with stations and APs placed uniformly at random in a square area of 50 m x 50 m is shown in Figures 4.19 - 4.23. All flows are always backlogged. As shown in Figure 4.19(a), in this network topology and with 10 transmitting stations, throughput is not improved by Concurrent-MAC protocol. The reason is that, in this scenario, due to low density of stations, the number of concurrent transmissions in Concurrent-MAC and 802.11 is the same. We note that since the network is multi-hop, concurrent transmissions happen in 802.11 protocol as well, since stations not in the carrier sense range of each other can transmit concurrently. As we observe in Figures 4.19 - 4.23, throughput improvement achieved by Concurrent-MAC is different for different topolo-

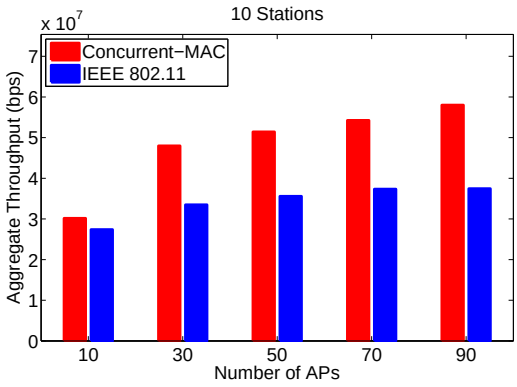


Figure 4.15: Topology number 4 (10 stations, area = 17 m x 17 m)

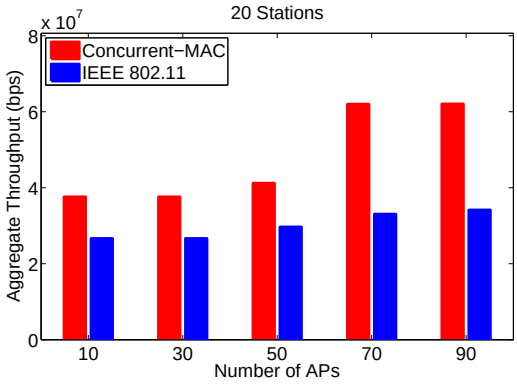


Figure 4.16: Topology number 4 (20 stations, area = 17 m x 17 m)

gies. As we explained before, network topology determines the number of concurrent transmitters in the network, and throughput of Concurrent-MAC is a function of number of concurrent transmitters in the network.

Throughput improvement for 50 and 100 stations and 20-100 APs placed uniformly at random in a square area of 100 m x 100 m is shown in Figure 4.24. As we observe in this figure, when the number of transmitting stations increases, throughput of Concurrent-MAC increases because number of stations being able to transmit concurrently increases with increasing number of stations. We also observe that, with fixed number of transmitting stations, increasing the number of APs might increase the throughput of Concurrent-MAC. The reason is that increasing the number of APs might increase the number of concurrent transmitters.

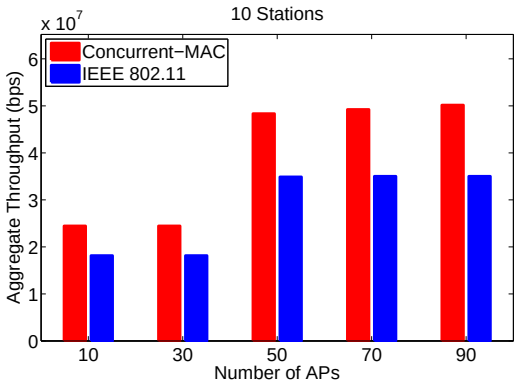


Figure 4.17: Topology number 5 (10 stations, area = 17 m x 17 m)

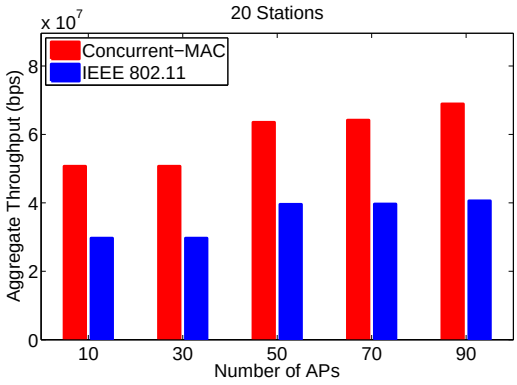
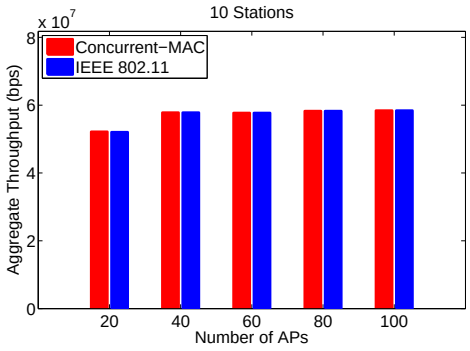


Figure 4.18: Topology number 5 (20 stations, area = 17 m x 17 m)

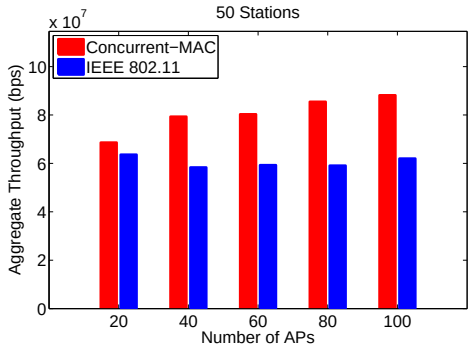
4.4.4 Networks with TCP Traffic

Having shown the performance improvement of Concurrent-MAC over 802.11 for full buffer CBR traffic, we further identify its performance in networks with TCP traffic. We perform simulations for networks of different sizes, i.e., 17 m x 17 m, 50 m x 50 m and 100 m x 100 m. APs and stations are placed uniformly at random in the area. In this set of simulations, any node might transmit concurrently with any other node, where a node might be a station transmitting TCP packets or an AP transmitting TCP acknowledgements.

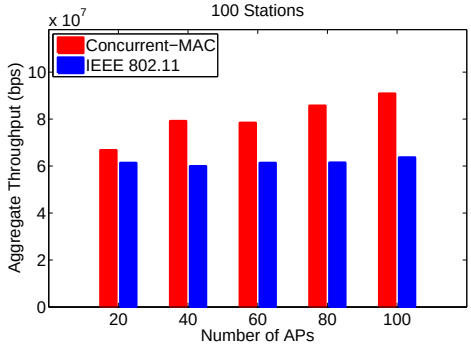
Figures 4.25 - 4.27 show the aggregate throughput of 802.11a and Concurrent-MAC for these three different area sizes. These figures show that Concurrent -MAC improves the aggregate throughput by 3.8% - 90.8%, compared to 802.11 protocol. When traffic is TCP, although the buffer of stations and APs might not be fully backlogged, network nodes might have few packets



(a) 10 stations

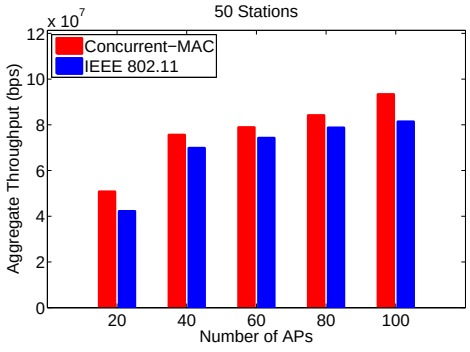


(b) 50 stations

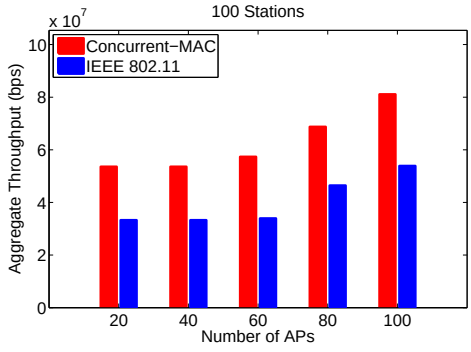


(c) 100 stations

Figure 4.19: Topology number 1, CBR traffic, area = 50 m x 50 m



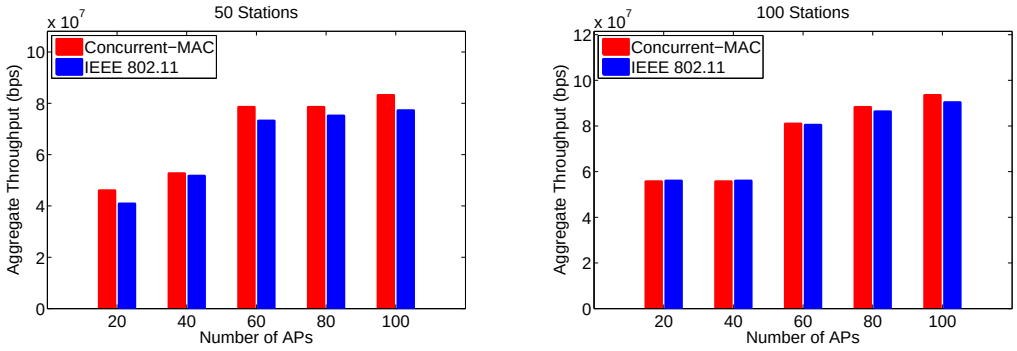
(a) 50 stations



(b) 100 stations

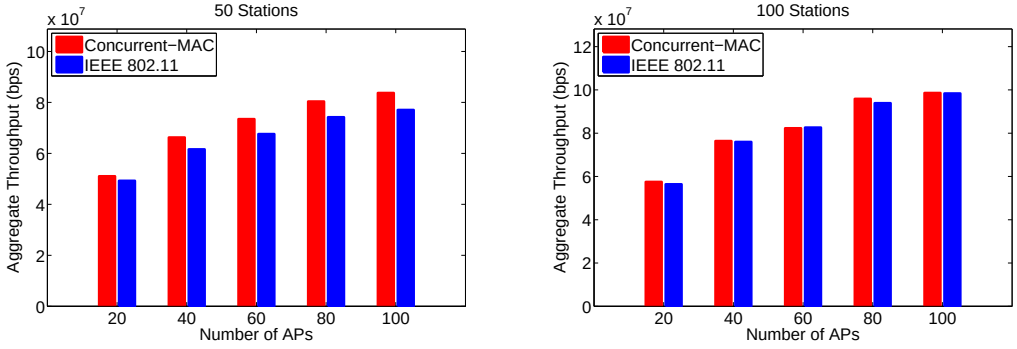
Figure 4.20: Topology number 2, CBR traffic, area = 50 m x 50 m

backlogged in their transmission queue, in which case privilege can be given to neighboring nodes for concurrent transmissions. This results in increased network throughput achieved by Concurrent-MAC, compared to 802.11 protocol.



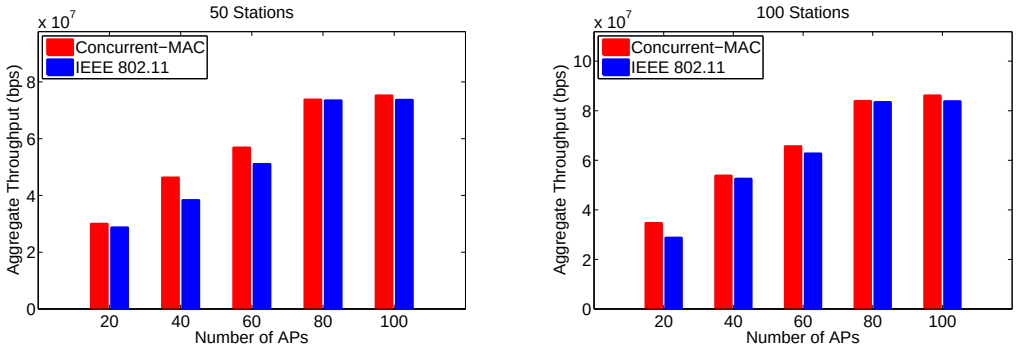
(a) 50 stations (b) 100 stations

Figure 4.21: Topology number 3, CBR traffic, area = 50 m x 50 m



(a) 50 stations (b) 100 stations

Figure 4.22: Topology number 4, CBR traffic, area = 50 m x 50 m

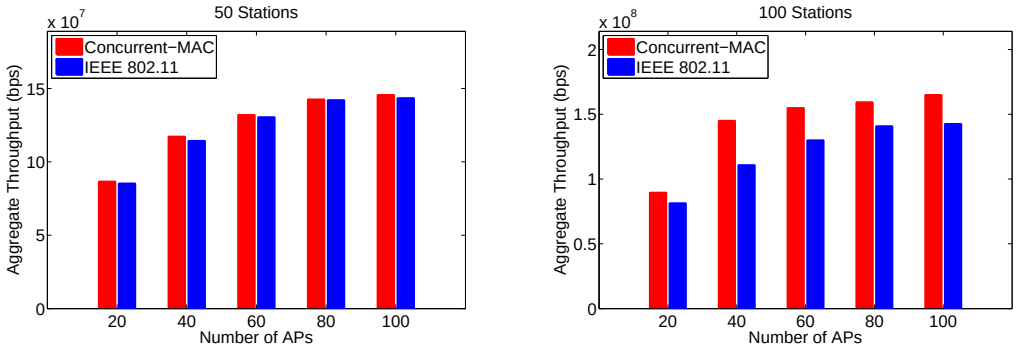


(a) 50 stations (b) 100 stations

Figure 4.23: Topology number 5, CBR traffic, area = 50 m x 50 m

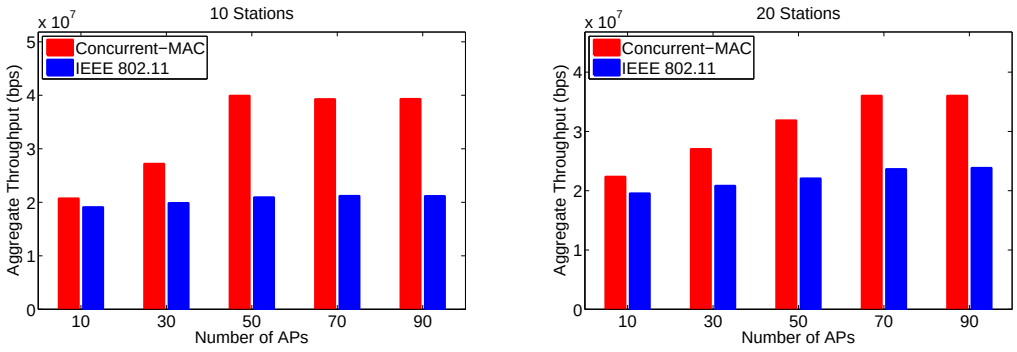
4.4.5 Performance Evaluation in Grid Topology

In this section, we evaluate the performance of Concurrent-MAC, where APs are placed in a grid structure, with equal distant between neighboring APs,



(a) 50 stations (b) 100 stations

Figure 4.24: Aggregate throughput (area = 100 m x 100 m)



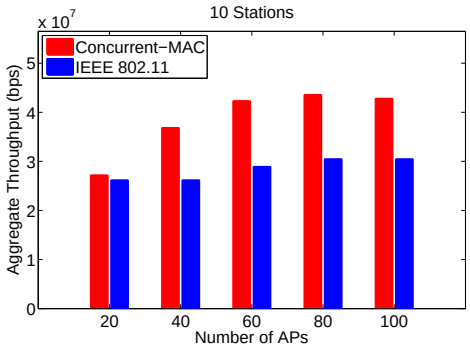
(a) 10 stations (b) 20 stations

Figure 4.25: Aggregate throughput (TCP traffic, area = 17 m x 17 m)

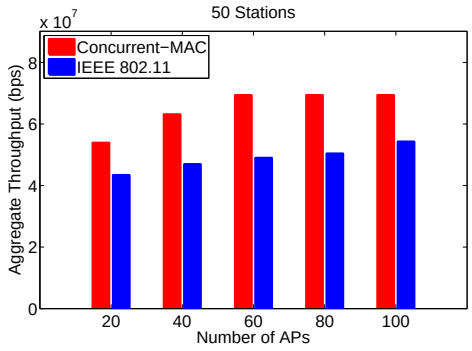
both horizontally and vertically. Stations are placed uniformly at random in the area and the placement is different for different topologies. Traffic is full buffer CBR. The aggregate throughput of Concurrent-MAC and 802.11 for 5 different gird topologies with area equal to 17 m x 17 m are shown in Figures 4.28-4.32. The number of transmitting stations is 10 and 20 is these figures. The performance improvement obtained from Concurrent-MAC, compared to 802.11, for these topologies is up to 100%.

Figures 4.33-4.35 show the improvement for a grid topology of size 50 m x 50 m. Figures 4.36 and 4.37 show the improvement for a grid topology of size 100 m x 100 m. We observe that the improvement obtained from Concurrent-MAC is different for different topologies and different numbers of stations and APs, since number of stations with concurrent neighbors changes based on nodes location.

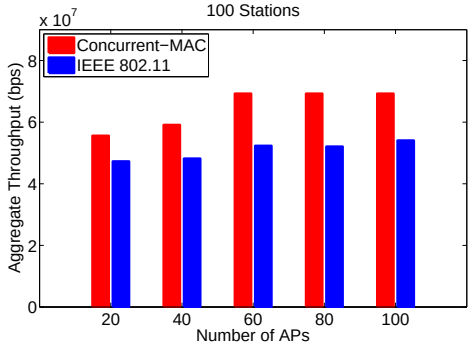
Comparing Figures 4.28-4.37 with Figures 4.9-4.24, we note that, for area



(a) 10 stations

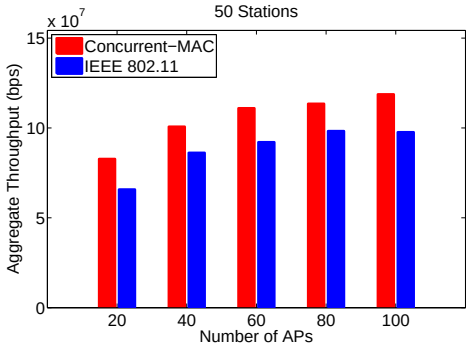


(b) 50 stations

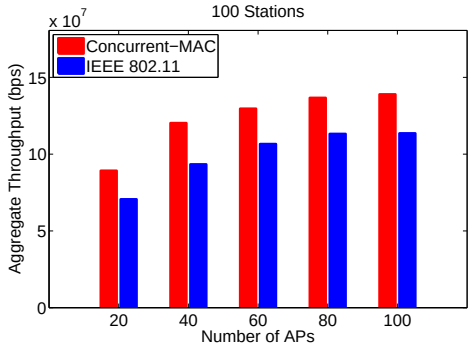


(c) 100 stations

Figure 4.26: Aggregate throughput (TCP traffic, area = 50 m x 50 m)



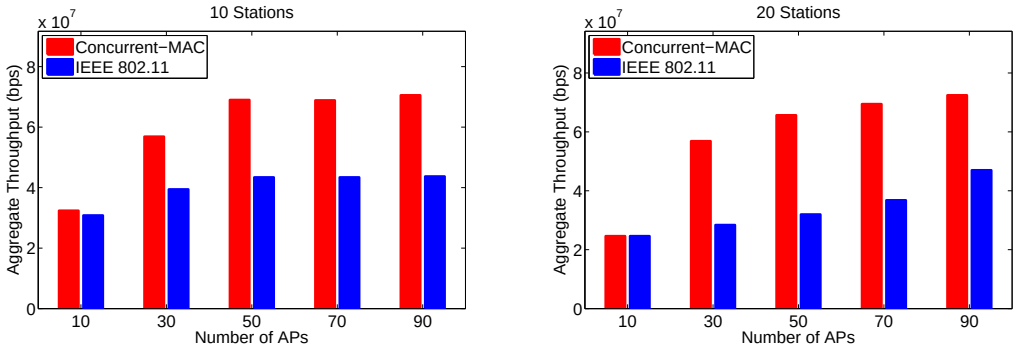
(a) 50 stations



(b) 100 stations

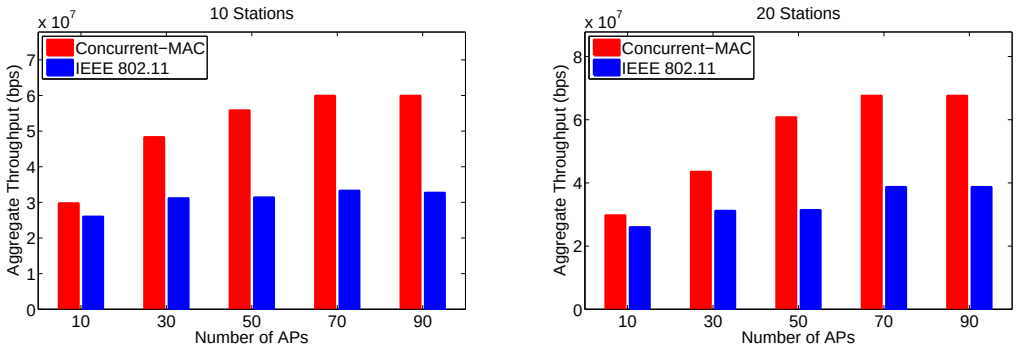
Figure 4.27: Aggregate throughput (TCP traffic, area = 100 m x 100 m)

size of 17 m x 17 m, the performance of Concurrent-MAC in grid topology is similar to its performance in topologies with random placement of APs. For area sizes of 50 m x 50 m and 100 m x 100 m, the improvement obtained by Concurrent-MAC is less for the case of grid topology. As we explained before, the improvement obtained by concurrent-MAC depends on network



(a) 10 stations (b) 20 stations

Figure 4.28: Grid topology number 1, CBR traffic, area = 17 m x 17 m



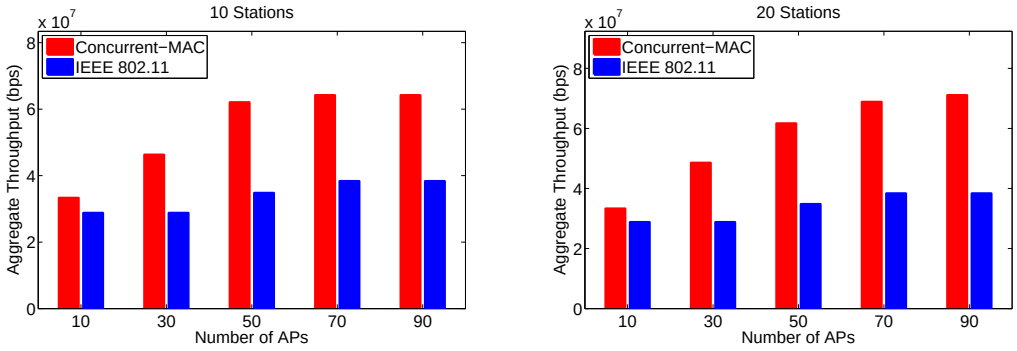
(a) 10 stations (b) 20 stations

Figure 4.29: Grid topology number 2, CBR traffic, area = 17 m x 17 m

topology, which determines the number of nodes with concurrent neighbors. For the topologies considered in Figures 4.33-4.37, the placement of APs and stations is such that not many nodes have concurrent pairs and as a result the improvement obtained by Concurrent-MAC is negligible in these topologies.

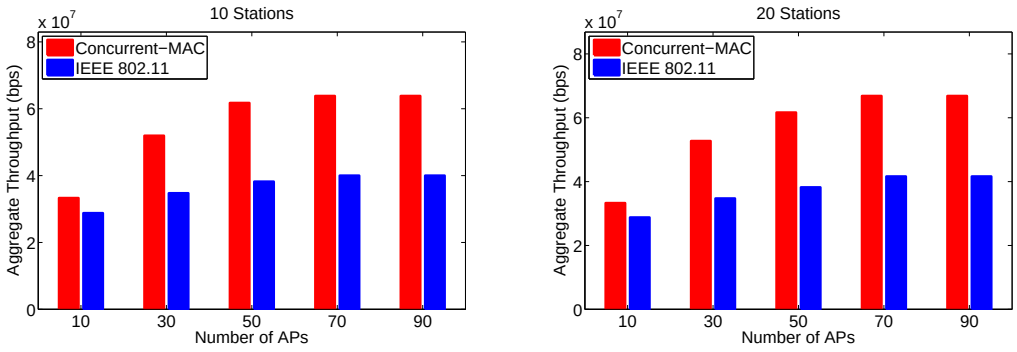
4.4.6 Performance Evaluation in Hexagon Topology

This section presents the simulation results in which APs are placed in a hexagon topology. As we know, hexagons can be tiled or tessellated in a regular pattern on a flat two-dimensional plane. That is, a hexagon can be bordered by six other hexagons, which can themselves be bordered by six hexagons (including each other), with no empty space left over. In this set of simulations, the considered area is a square, in which hexagons are tiled to fill the area. One AP is placed at the center of each hexagon. The number



(a) 10 stations (b) 20 stations

Figure 4.30: Grid topology number 3, CBR traffic, area = 17 m x 17 m

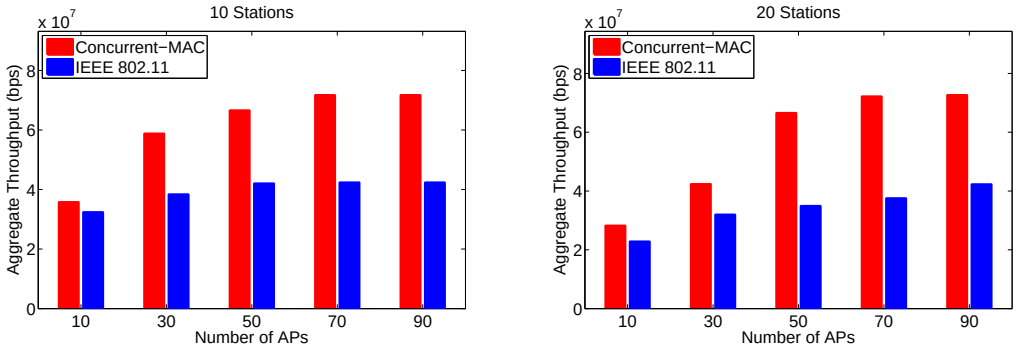


(a) 10 stations (b) 20 stations

Figure 4.31: Grid topology number 4, CBR traffic, area = 17 m x 17 m

of APs determines the diameter of the hexagons. Users are placed randomly in the area. Traffic is full buffer CBR.

The aggregate throughput of Concurrent-MAC and 802.11 for the hexagon topology with total area of size 17 m x 17 m is shown in Figures 4.38 and 4.39. Number of transmitting stations is 10 in Figure 4.38 and 20 in Figure 4.39. The performance improvement obtained from Concurrent-MAC, compared to 802.11, for this topology is up to 68%. Figures 4.40-4.42 show the improvement for an area of size 50 m x 50 m, in which APs are placed based on a hexagon tessellation. Figures 4.43 and 4.44 show the improvement for an area of size 100 m x 100 m with hexagon tessellation. These figures show that, with the planned placement of APs based on hexagon tessellation, Concurrent-MAC might result in throughput improvement, compared to 802.11 protocol.



(a) 10 stations (b) 20 stations

Figure 4.32: Grid topology number 5, CBR traffic, area = 17 m x 17 m

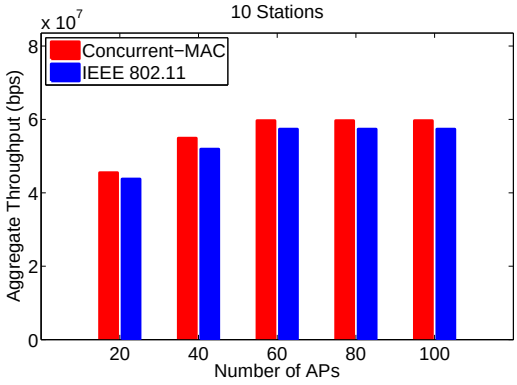


Figure 4.33: Grid topology, CBR traffic, 10 stations, area = 50 m x 50 m

4.4.7 Performance Evaluation in Networks with Few Stations and Few APs (Home Setting)

In this section, we evaluate the performance of Concurrent-MAC in small area size networks with few stations and few APs. This set of simulations considers home setting, in which few stations and APs are present in a small area. We consider an area of size 6 m x 6 m, with 5 users and 5 APs placed randomly in the area. Figure 4.45 shows the throughput of Concurrent-MAC and 802.11 for 5 such topologies. We note that sometimes Concurrent-MAC achieves higher throughput than 802.11, because for some of the topologies, placement of stations and APs is such that some nodes are concurrent transmitters, i.e., they can transmit concurrently. For some other topologies, no improvement is achieved by Concurrent-MAC, simply because there are no concurrent transmitters in these topologies.

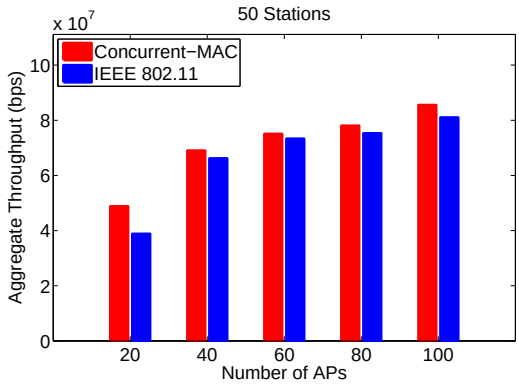


Figure 4.34: Grid topology, CBR traffic, 50 stations, area = 50 m x 50 m

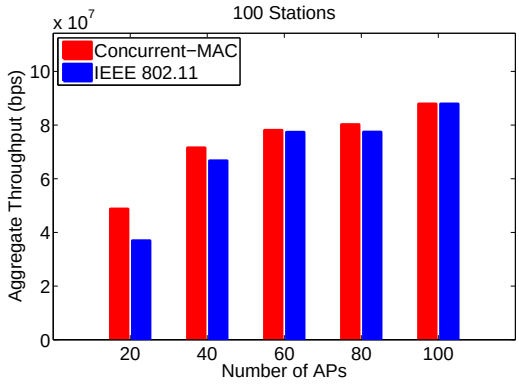


Figure 4.35: Grid topology, CBR traffic, 100 stations, area = 50 m x 50 m

4.4.8 Decreasing Transmission Rate to Increase Concurrent Transmissions

As we explained in Section 4.3.6, decreasing transmission rate of nodes might result in increasing the number of concurrent transmissions in the network. In a single contention domain, if total throughput of nodes transmitting concurrently, at lower rates, is more than the throughput of nodes transmitting individually, with their maximum possible rate, concurrent transmission at lower rates increases aggregate throughput of Concurrent-MAC, compared to 802.11.

In Figure 4.46(a), we consider a single contention domain in which two stations and two APs are placed as shown in Figure 4.5. The distance between stations and APs is such that concurrent transmission is not possible at highest transmission rate of stations, i.e., 54 Mbps. But if stations decrease

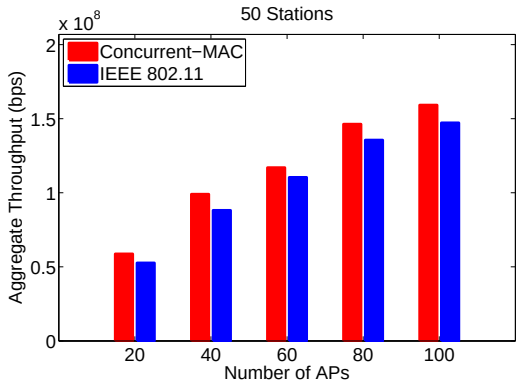


Figure 4.36: Grid topology, CBR traffic, 50 stations, area = 100 m x 100 m

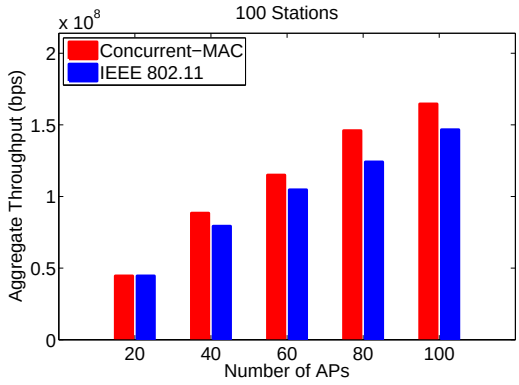


Figure 4.37: Grid topology, CBR traffic, 100 stations, area = 100 m x 100 m

their transmission rate to 36 Mbps, they are able to transmit concurrently. As shown in Figure 4.46(a), throughput of both stations is improved by Concurrent-MAC, when stations decrease their transmission rate from 54 Mbps to 36 Mbps.

Figure 4.46(b) shows the throughput of three stations placed in a single contention domain, as shown in Figure 4.7, where no concurrent transmission is possible at rate 54 Mbps, but all 3 stations are able to transmit concurrently, if transmission rate is decreased to 24 Mbps. The reason for improving throughput in this scenario is that, by transmitting concurrently, stations do not share the channel in time; instead, they transmit all the time on the channel.

We then consider 5 different topologies in which 5 stations and 5 APs are placed, uniformly at random, in an area of size 6 m x 6 m. This area size is

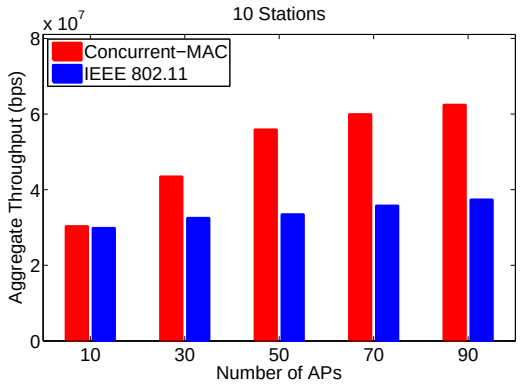


Figure 4.38: Hexagon topology, CBR traffic, 10 stations, area = 17 m x 17 m

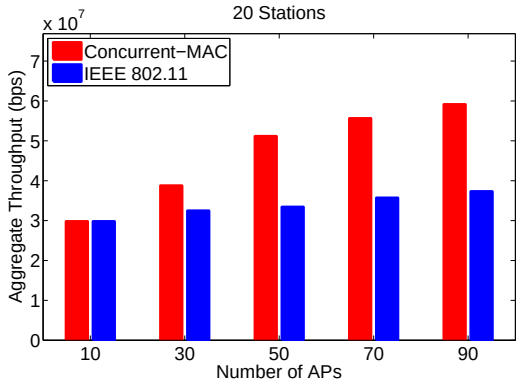


Figure 4.39: Hexagon topology, CBR traffic, 20 stations, area = 17 m x 17 m

close to the size of an office or a conference room. These considered 5 topologies are such that concurrent transmission at rate 54 Mbps is not possible in Concurrent-MAC protocol. We note that since concurrent transmission at rate 54 Mbps is not possible in these topologies, Concurrent-MAC and 802.11 preform the same, if nodes only transmit at rate 54 Mbps. Figure 4.47 shows the aggregate throughput of Concurrent-MAC and 802.11 protocol, where transmission rate of nodes is chosen based on Procedures 21 and 23. Figure 4.47 shows an improvement of 11%-56%, which is achieved by decreasing transmission rate to increase concurrency.

As we have found out in our simulations, in multi-hop networks, decreasing transmission rates of stations to increase concurrency does not increase network throughput. We have explained why this happens in Section 4.3.6.

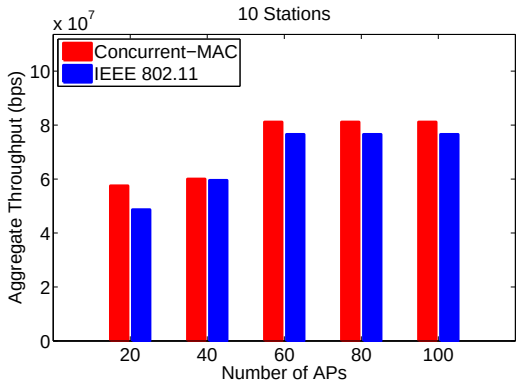


Figure 4.40: Hexagon topology, CBR traffic, 10 stations, area = 50 m x 50 m

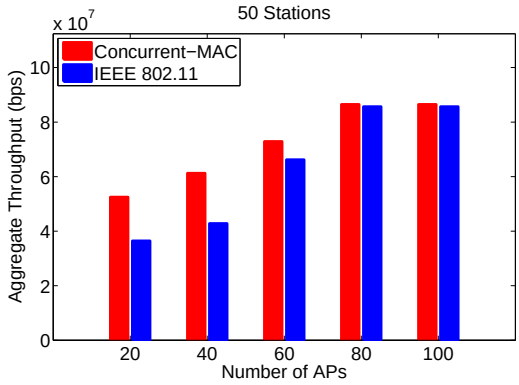


Figure 4.41: Hexagon topology, CBR traffic, 50 stations, area = 50 m x 50 m

4.5 Conclusion and Future Work

In this chapter, we presented the design and performance evaluation of *Concurrent - MAC*. Concurrent-MAC is a MAC protocol that uses an opportunistic overhearing mechanism to schedule network nodes for concurrent transmissions in dense WLANs. The main design goal of Concurrent-MAC is to increase aggregate throughput by allowing concurrent transmissions that can be received or captured successfully. Our ns-2 simulations show that Concurrent-MAC can sometimes, but not always, achieve significant improvement in system throughput compared to 802.11 DCF. The performance of Concurrent-MAC depends on the underlying network topology, which determines the number of nodes with concurrent neighbors. In networks where

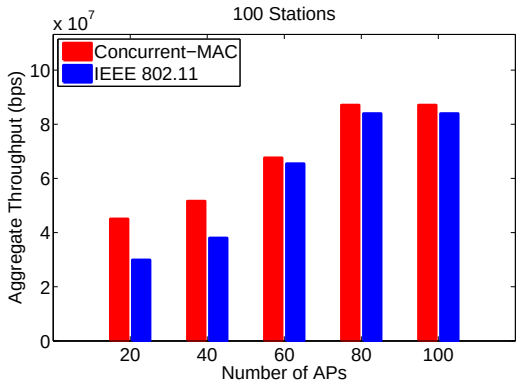


Figure 4.42: Hexagon topology, CBR traffic, 100 stations, area = 50 m x 50 m

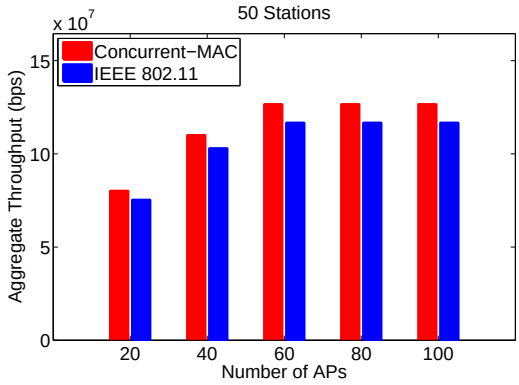


Figure 4.43: Hexagon topology, CBR traffic, 50 stations, area = 100 m x 100 m

every node has concurrent neighbors, Concurrent-MAC performs the best, since every transmitting node can give privilege to one of its concurrent neighbors.

Future work can implement Concurrent-MAC in a wireless testbed to evaluate its performance in a real wireless channel. Furthermore, as we discussed in this chapter, by special placement of APs, two or three nearby stations might be able to transmit concurrently. In the future, designing the placements of APs in a WLAN running Concurrent-MAC, with the goal of maximizing network throughput, can be an interesting extension to this work.

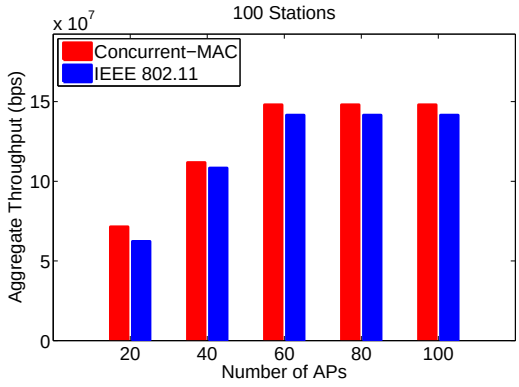


Figure 4.44: Hexagon topology, CBR traffic, 100 stations, area = 100 m x 100 m

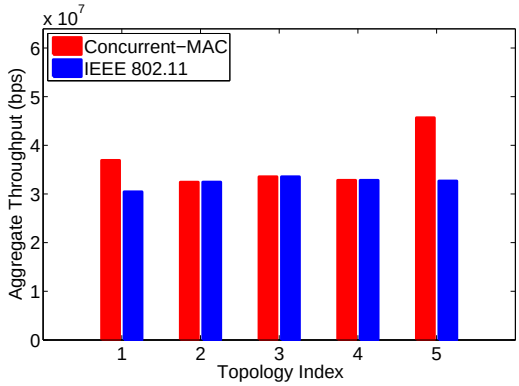
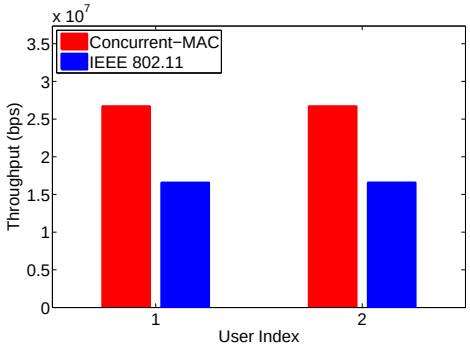
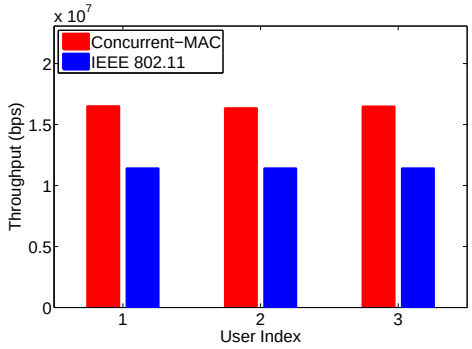


Figure 4.45: Home setting, CBR traffic, 5 stations, 5 APs, area = 6 m x 6 m



(a) Two concurrent transmitters at rate 36 Mbps



(b) Three concurrent transmitters at rate 24 Mbps

Figure 4.46: Decreasing transmission rate to increase concurrency

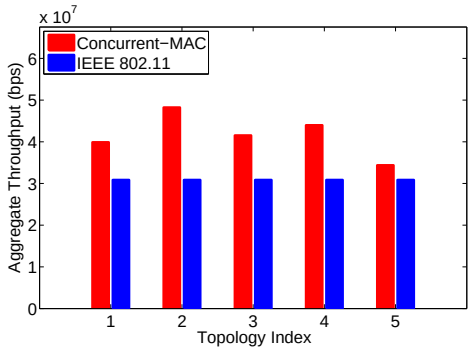


Figure 4.47: Aggregate Throughput (5 topologies of size 6 m x 6 m)

CHAPTER 5

CONCLUSION

In this thesis, we have presented protocols and algorithms that exploit wireless broadcast property and opportunistic overhearing to improve performance in wireless networks. Our protocols are based on the fact that the wireless channel is a shared medium and messages transmitted on the channel might be overheard by the nearby stations. We have developed mutual exclusion algorithms and MAC protocols based on wireless broadcast property which enables opportunistic message overhearing, one of the main characteristics of wireless networks. We have identified settings in which wireless broadcast property can be beneficial for improving performance. We now provide a brief chapter-level summary in the following paragraphs.

In Chapter 2, we have designed two distributed token based mutual exclusion algorithms for wireless networks, called TOA and TROA. Our algorithms exploit the shared nature of the wireless channel in which nodes can overhear the messages not intended for them. The design goal is to decrease the number of transmitted messages and delay to enter the critical section, based on the information obtained from overheard messages. We have measured the performance of our algorithms through ns-2 simulations in networks of different sizes and under various rates of the demand for the token. We have discussed under what conditions the performance of the considered mutual exclusion algorithms is increased by exploiting message overhearing. We have also implemented our algorithms in a wireless testbed and have measured the performance improvement in an indoor office environment. Our simulations and measurements show that the highest improvement is achieved in single-hop networks, where message of every station might be overheard by any other station. In single hop networks, by exploiting message overhearing, the number of transmitted messages and delay per CS entry might be decreased by half. As the size of the network grows, the improvement achieved by message overhearing is also decreased.

In Chapter 3, we have presented Token-DCF, a distributed and dynamically adaptive MAC protocol for wireless networks. The main focus of Token-DCF is on reducing idle and collision times by introducing an implicit token passing algorithm. In Token-DCF, a transmitting station schedules one of its neighboring stations for the next transmission using a distributed opportunistic token passing algorithm. Furthermore, packet overhearing is employed to exchange scheduling information across the network. Our simulation results show that Token-DCF can achieve significant improvement in system throughput and channel access delay compared to 802.11 DCF for most network configurations.

In Chapter 4, we have presented the design and performance evaluation of Concurrent-MAC. Concurrent-MAC is a MAC protocol, designed for infrastructure-based wireless networks, which uses an opportunistic overhearing mechanism to enable concurrent transmissions in dense WLANs with large number of APs. The main design goal of Concurrent-MAC is to increase aggregate throughput by allowing concurrent transmissions that can be received successfully by the backhaul of APs. Based on SINR values between stations and APs, sets of concurrent transmitters are identified by the backhaul of APs. Network stations schedule their neighbors for concurrent transmissions via an opportunistic token passing mechanism. Our simulation results show that Concurrent-MAC can improve network throughput in dense deployments of wireless LANs.

The MAC protocols and mutual exclusion algorithms presented in this thesis can be useful in designing next generation wireless systems.

REFERENCES

[1] K. Raymond, "A tree-based algorithm for distributed mutual exclusion," *ACM Transactions on Computer Systems*, vol. 7, pp. 61–77, 1989.

[2] M. Naimi and M. Trehel, "A distributed algorithm for mutual exclusion based on data structures and fault tolerance," in *Proceedings of Sixth Annual International Phoenix Conference on Computers and Communications*, pp. 33–39, 1987.

[3] D. Manivannan and M. Singhal, "An efficient fault-tolerant mutual exclusion algorithm for distributed systems," in *Proceedings of the International Conference on Parallel and Distributed Computing Systems*, pp. 525–530, 1994.

[4] D. Agrawal and A. Abbadi, "An efficient and fault-tolerant solution for distributed mutual exclusion," *ACM Transactions on Computer Systems*, vol. 9, pp. 1–20, 1991.

[5] M. Singhal, "A heuristically-aided algorithm for mutual exclusion in distributed systems," *IEEE Transactions On Computers*, pp. 651-662, May 1989.

[6] J. Walter, J. Welch, and N. Vaidya, "A mutual exclusion algorithm for ad hoc mobile networks," *Wireless Networks*, vol. 7, pp. 585–600, 2001.

[7] W. Wu, J. Cao, and M. Raynal, "A dual-token-based fault tolerant mutual exclusion algorithm for manets," in *Proceedings of the 3rd international conference on Mobile ad-hoc and sensor networks*, pp. 572–583, 2007.

[8] A. Akella, G. Judd, S. Seshan, and P. Steenkiste, "Self-management in chaotic wireless deployments," in *MobiCom '05*, August 2005.

[9] G. Ricart and A.K. Agrawala, "On mutual exclusion in computer networks," *Communication of the ACM*, pp.147-148, February 1983.

[10] G. Le Lann, "Distributed systems, towards a formal approach," in *IFIP Congress*, Toronto, pp. 155-160, 1977.

[11] I. Suzuki and T. Kazami, "A distributed mutual exclusion algorithm," *ACM Transactions on Computer Systems*, vol. 3, pp. 344-349, 1985.

[12] Y. Chang, M. Singhal, and M. Liu, "A fault tolerant algorithm for distributed mutual exclusion," in *Proc. of 9th Symposium On Reliable Distributed Systems*, pp. 146-154, 1990.

[13] D.M. Dhamdhere and S.S. Kulkarni, "A Token based k-resilient mutual exclusion algorithm for distributed systems," *Information Processing Letters*, pp.151-157, 1994.

[14] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, pp. 558-565, July 1978.

[15] M. Maekawa, "A $\sqrt[2]{N}$ algorithm for mutual exclusion in decentralized systems," *ACM Transactions on Computer Systems*, vol. 3, pp. 145-159, May 1985.

[16] M. Benchaba, A. Bouabdallah, N. Badache, and M. Ahmed-Nacer. "Distributed mutual exclusion algorithms in mobile ad hoc networks: an overview," *ACM SIGOPS Operating Systems Review*, 2004.

[17] F. Cali, M. Conti, and E. Gregori, "Dynamic tuning of the ieee 802.11 protocol to achieve a theoretical throughput limit," *IEEE/ACM Trans. on Networking,* vol. 8, no. 6, December 2000.

[18] Y. C. Tay, K. Jamieson, and H. Balakrishnan, "Collision-minimizing csma and its applications to wireless sensor networks," *IEEE Journal on Selected Areas in Communications,* vol. 22, 2004.

[19] M. Heusse, F. Rousseau, R. Guillier, and A. Duda, "Idle sense: an optimal access method for high throughput and fairness in rate diverse wireless lans," *SIGCOMM Comput. Commun. Rev.,* vol. 35, no. 4, 2005.

[20] X. Yang and N. H. Vaidya, "A wireless mac protocol using implicit pipelining," *IEEE Trans. on Mobile Computing,* vol. 5, no. 3, 2006.

[21] Z. Zeng, Y. Gao, K. Tan and P. R. Kumar, "CHAIN: Introducing minimum controlled coordination into random access MAC," in *Proceedings of INFOCOM 2011,* pp. 2669-2677, 2011.

[22] "Wireless medium access control (MAC) and physical layer (PHY) specifications: Medium access control (MAC) enhancements for quality of service (QoS)," IEEE Std. 802.11e/Draft 5.0, July 2003.

[23] "802.11n: Next-generation wireless LAN technology," Broadcom Corporation, 2006.

[24] ANSI/IEEE Standard 802.4, "Token-passing bus access method and physical layer specifications," IEEE, 1985.

[25] M. Ergen, D. Lee, A. Puri, P. Varaiya, R. Attias, R. Sengupta, S. Tripakis, "Wireless token ring protocol," in *Proceedings of the Eighth IEEE Symposium on Computers and Communications (ISCC 2003)* pp. 710-715, 2003.

[26] E. E. Johnson, Z. Tang, M. Balakrishnan, J. Rubio, H. Zhang, and S. Sreepuram, "Robust token management for unreliable networks," in *Proceedings of the 2003 IEEE Conference on Military Communications, MILCOM'03,* vol. 1, 2003.

[27] L. Tassiulas and A. Ephremides, "Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks," *IEEE Transactions on Automatic Control,* vol. 37, no. 12, pp. 1936-1948, 1992.

[28] A. Dimakis and J. Walrand. "Sufficient conditions for stability of longest-queue-first scheduling: Second-order properties using fluid limits." *Advances in Applied Probability,* vol. 38, no. 2, pp. 505-521, 2006.

[29] S. Shakkottai, R. Srikant, and A. Stolyar. "Pathwise optimality of the exponential scheduling rule for wireless channels," *Advances in Applied Probability,* vol. 36, no. 4, pp. 1021-1045, 2004.

[30] A. Eryilmaz, R. Srikant, and J. Perkins. "Stable scheduling policies for fading wireless channels," in *Proceedings of IEEE International Symposium on Information Theory,* 2003.

[31] M. Andrews, K. Kumaran, K.Ramanan, A. Stolyar, R. Vijayakumar, and P. Whiting. "Providing quality of service over a shared wireless link," *IEEE Communications Magazine,* February 2001.

[32] F. P. Kelly. "Charging and rate control for elastic traffic," *European Transactions on Telecommunications,* vol. 8, no. 1, pp. 33-37, 1997.

[33] Q. Chen, F. Schmidt-Eisenlohr, D. Jiang, M. Torrent-Moreno, L. Delgrossi, and H. Hartenstein, "Overhaul of ieee 802.11 modeling and simulation in ns-2," in *Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of wireless and mobile systems*, pp. 159-168, October 2007.

[34] J. Lee, W. Kim, S.-J. Lee, D. Jo, J. Ryu, T. Kwon, and Y. Choi, "An experimental study on the capture effect in 802.11a networks," in *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization*, pp. 19-26, 2007.

[35] D. Qiao, S. Choi, and K. G. Shin, "Interference analysis and transmit power control in IEEE 802.11 a/h wireless LANs," in *IEEE/ACM Transactions on Networking*, 2007.

[36] T. S. Rappaport, *Wireless Communications: Principles and Practice*. Vol. 2. New Jersey: Prentice Hall PTR, 1996.

[37] S. Y. Seidel and T. S. Rappaport, "914 MHz path loss prediction models for indoor wireless communications in multifloored buildings," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 2, pp. 207-217, 1992.

[38] D. B. Faria, "Modeling signal attenuation in IEEE 802.11 wireless LANs - vol. 1," Tech. Rep. TR-KP06-0118, Stanford University, July 2005.

[39] A. Miu, H. Balakrishnan, and C. E. Koksal, "Improving loss resilience with multi-radio diversity in wireless networks," in *Proc. of ACM MobiCom*, pp. 16-30, August 2005.

[40] Y. Zhu, Z. Niu, Q. Zhang, B. Tan, Z. Zhou, and J. Zhu, "A multi-AP architecture for high-density WLANs: protocol design and experimental evaluation," in *Proc. of IEEE SECON,* pp. 28-36, June 2008.

[41] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications,* IEEE Standard 802.11, 2007.

[42] K. Mittal and E. Belding. "RTSS/CTSS: Mitigation of exposed terminals in static 802.11-based mesh networks," in Proc. of *IEEE WiMesh Workshop*, Sept. 2006.

[43] C. Chen, E. Seo, H. Kim and H. Luo, "Self-learning collision avoidance for wireless networks," in *INFOCOM*, April 2006.

[44] M. Vutukuru, K. Jamieson, and H. Balakrishnan, "Harnessing exposed terminals in wireless networks," in *USENIX NSDI*, June 2008.

[45] T. S. Kim, H. Lim, and J. C. Hou, "Improving spatial reuse through tuning transmit power, carrier sense threshold, and data rate in multihop wireless networks," in *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking*, pp. 366-377, September 2006.