Efficient Use of Resources in Mobile Ad Hoc Networks

by

Bora Karaoglu

Submitted in Partial Fulfillment of the
Requirements for the Degree
Doctor of Philosophy

Supervised by Professor Wendi B. Heinzelman

Department of Electrical and Computer Engineering
Arts, Sciences and Engineering
Edmund A. Hajim School of Engineering and Applied Sciences

University of Rochester
Rochester, New York

2013

Biographical Sketch

The author was born in Izmir, Turkey. He received B.Sc. degrees in Electrical and Electronics Engineering (major) and Industrial Engineering (double major) from Middle East Technical University, Ankara, Turkey in June 2006 and June 2007, respectively. From March 2007 to July 2007 he worked as a system engineer in ASELSAN Corp., Ankara, Turkey. He began doctoral studies in the Department of Electrical and Computer Engineering at the University of Rochester in 2007 and has been a member of the Wireless Communications and Networking Group at the University of Rochester since August 2007. He pursued his research in the field of wireless communications and networking under the direction of Prof. Wendi Heinzelman. He has been working on a joint research project supported in part by the University of Rochester Center for Emerging and Innovative Sciences and in part by Harris Corporation, RF Communications Division. He received the Master of Science degree from the University of Rochester in 2008. He has held the graduate chair position in the IEEE University of Rochester Student Branch since January 2009. He worked as a graduate intern at Harris Corporation, RF Communications Division from June 2009 to September 2009, and he worked as a graduate intern at the General Electric Global Research Center, Advanced Communication Systems Laboratory from September 2011 to December 2011. His current research interests lie in the areas of wireless communications and networking, mobile computing, and multimedia communications.

The following publications were a result of work conducted during doctoral study:

Karaoglu, Heinzelman, "Cooperative Load Balancing and Dynamic Channel Allocation for Mobile Ad Hoc Networks," IEEE Transactions on Mobile Computing. (under review)

Karaoglu, Heinzelman, "A Dynamic Channel Allocation Scheme Using Spectrum Sensing for Mobile Ad Hoc Networks," IEEE Global Communications Conference (GLOBECOM), 2012.

Karaoglu, Demirkol, Heinzelman, "Exploring the Benefits of Symbiotic Routing," International Conference on Computer Communications and Networks (ICCCN), 2011.

Karaoglu, Numanoglu and Heinzelman, "Analytical Performance of Soft Clustering Protocols," Elsevier Ad Hoc Networks: Special Issue on Multimedia Networks, 2010.

Karaoglu, Heinzelman, "Multicasting vs. Broadcasting: What are the Tradeoffs," IEEE Global Communications Conference (GLOBECOM), December 2010.

Karaoglu, Numanoglu, Heinzelman, "Adaptation of TDMA Parameters Based on Network Conditions," IEEE Wireless Communication and Networking Conference (WCNC), April 2009.

Acknowledgments

I would like to begin by expressing my sincere appreciation and gratitude to my thesis advisor, Professor Wendi Heinzelman, for her academic guidance and enthusiastic encouragement over the past six years. Among many other things, she spent endless hours proofreading my research papers and giving me excellent suggestions, which always resulted in excellent documents acknowledged by all the reviewers of our academic papers. Her professional yet caring approach and enthusiasm provided the support that I needed for my research.

I would like to thank Professors Mark Bocko and Muthu Venkitasubramaniam for acting as members of my thesis committee.

I would also like to thank to all my colleagues and friends at the University of Rochester. Specifically, I would like to thank all the members of the Wireless Communications and Networking Group. It was a pleasure to work in such a friendly environment.

I also want to express my thanks to all my friends in Rochester for making it feel like home. I am grateful for the invaluable friendship and support they provide, which I name only a few: Alejandro Gomez, Marina Feigenson, Tolga Numanoglu, Ilker Demirkol, Ulas Misirli, Basak Oztan, Selcuk Kose, Laura Ciammaruchi, Orhan Bulan and Lori White.

Above all I would like to thank my parents, Fusun and Mustafa, as well as my brother, Yigit, for their endless love, support, encouragement and patience. I

would like to acknowledge the sacrifices that they made to ensure that I had an excellent education. It is to them that I dedicate this dissertation.

Abstract

Efficient use of the resources in mobile ad hoc networks (MANETs) is of great importance to maintain the required quality of service and to prolong the network lifetime. The utilization of the resources such as bandwidth and energy depends on a number of conditions such as network size, node density, and load distribution. These conditions are uncontrollable and often vary throughout the operation of the network. In order to efficiently use the resources, the protocols that determine the behavior of the network should dynamically adapt to these changing conditions.

My thesis is that a protocol architecture for MANETs that dynamically adapts to changing conditions based on cooperation and information sharing leads to more efficient use of the system resources compared to competition based architectures. In particular, in this dissertation we explore the benefits of adaptation based on cooperation and information sharing at the medium access control (MAC) and network (routing) layers of the protocol stack.

At the MAC layer, we develop an analytical model that reflects the relationships between protocol parameters and the overall performance of the protocol under various network conditions. This model reveals that the protocol parameters at the MAC layer can be adjusted to make best use of the channel resources depending on the application requirements and network conditions obtained through information sharing, such as average network load density. In order to provide a dynamic system that adapts not only to changing conditions but also to spatially non-uniform traffic load distributions, a lightweight dynamic channel allocation

algorithm and a cooperative load balancing algorithm that facilitate efficient use of resources based on local information sharing are proposed. Through extensive simulations, we show that both dynamic channel allocation and cooperative load balancing improve the bandwidth efficiency under non-uniform load distributions compared with protocols that do not use these mechanisms as well as compared with the IEEE 802.11 uncoordinated protocol.

Properly routing the data over a MANET is another challenging topic due to the dynamic behavior of the network, yet it is also crucial in terms of efficient use of resources. Two important routing schemes, network-wide broadcasting and multicasting, are investigated for trade-offs and merged into a single framework. The framework allows the selection of the optimal routing scheme based on the network conditions obtained through information sharing, leading to the best use of the system resources in terms of spectrum efficiency and energy efficiency. The interaction of a network with other networks coexisting at the same site also strongly determines its efficiency. We developed an approach for symbiotic networking using hybrid nodes, and our results clearly show that symbiotic networking can provide vital support to co-located networks, which is especially important in resource-constrained networks such as MANETs.

Although theoretical analysis and simulations are efficient tools to comparatively evaluate the efficiency of different protocols, they cannot reflect many of the challenges for real implementation of these protocols, such as clock-drift, synchronization, imperfect physical layers, and interference from devices outside of the system. In order to prove the feasibility of the MAC and Network layer algorithms proposed in this thesis, a working prototype system that incorporates these algorithms is implemented on the Microsoft Research's SORA software defined radio (SDR) platform. The experiments with the prototype system show not only the viability of real time communications but also show the resilience of the system against interference.

To sum up, a variety of methods ranging from MAC layer techniques for optimal spatial reuse and dynamic channel allocation, to network layer techniques for optimal data dissemination schemes and symbiotic interactions with co-located networks are described in this thesis. These concepts enable protocol architectures for MANETs that dynamically adapt to changing conditions based on cooperation and local information sharing. The efficient use of the limited bandwidth and energy resources obtained through such protocol architectures with a realistic set of constraints ensure the viability of future applications.

Contributors and Funding Sources

This work was supervised by a dissertation committee consisting of Professors Wendi Heinzelman (advisor) and Mark Bocko of the Department of Electrical and Computer Engineering and Professor Muthu Venkitasubramaniam of the Department of Computer Science. The analytical model described in Chapter 3 is developed in collaboration with Dr. Tolga Numanoglu. The symbiotic routing described in Chapter 6 is developed in collaboration with Dr. Ilker Demirkol. All other work conducted for the dissertation was completed by the student independently. Graduate study was supported in part by the University of Rochester Center for Emerging and Innovative Sciences and in part by Harris Corporation, RF Communications Division.

Table of Contents

List of Tables

List of Figures

1 Introduction

Wireless communication has been around for over a century and has within the last decade become a regular mode of communication in people's everyday lives, thanks to the success of cellular and WiFi communication. Recently, researchers have focused on eliminating the need for fixed infrastructures in wireless communication, which has led to the development of ad hoc networks. A mobile ad hoc network (MANET) further considers node mobility within the ad hoc setting. Efficient use of resources and adaptation are vital in order to create a high performance MANET. This dissertation addresses the efficient use of network resources to obtain the desired quality of service and performance in MANETs.

1.1 Mobile Ad Hoc Networks

A mobile ad hoc network (MANET) is a self-configuring communication system that uses the nodes themselves as not only sources and sinks but also routers. Nodes in a MANET are typically battery operated devices with limited-range, half-duplex radios for communication. MANETs are easy to set up and use since their operation does not depend on any fixed infrastructure. There are many applications that can benefit from MANETs such as:

- Military tactical operations. A communication network that relies on a certain infrastructure is not desirable for military tactical operations, as it constitutes a soft spot in hostile environments. Elimination of the need for the hard/impossible to set up fixed infrastructure makes MANETs perfect candidates for such operations.

- Search and rescue missions. Oftentimes search and rescue missions are performed in remote locations with no communication infrastructure, such as the top of a mountain, the middle of a forest or inside a cave. MANETs are easy to use communication systems for such scenarios.

- Disaster relief. MANETs provide communication in environments where existing infrastructure is destroyed or left inoperable.

- Law enforcement. Law enforcement operations can be extended to include locations with no communication infrastructure. MANET systems provide fast and secure communication in such scenarios.

- Commercial use. MANETs can be used to support data exchange between people and applications in large meetings and conventions.

MANETs are unique among communication networks, as can be observed from the vital application areas. However, the unique characteristics required by these applications necessitate unique solutions and differentiate MANETs from other conventional networks. There are various challenges that have to be taken into account when designing a MANET.

First of all, the communication channel between the nodes in the network is highly unreliable. A MANET operates over wireless channels that incur higher bit errors compared to wired interfaces. MANET protocols have to be designed with the assumption of an erroneous channel. MANETs also are designed to work in any environment, whether it is a desert, forest or mountainous region. The lack of

a-priori knowledge about the propagation characteristics of the wireless medium also presents challenges to protocol design for MANETs.

Node mobility is another challenge in the design of MANETs. The topology of a MANET can change not only with changing propagation characteristics of the medium but also due to the mobility of the nodes in the network. In order to reliably convey information, MANET protocols have to include mechanisms for proper mobility management. Having limited storage and computational capabilities further restrict the range of algorithms that can be used in MANETs.

Moreover, MANETs have limited bandwidth and energy resources. The assumption of mobility inherently limits the energy supply available at each node. Thus, it is important for a MANET to be energy efficient and energy aware. Typically, the bandwidth available for the communication is also limited. The erroneous channel characteristics further decrease the channel capacity, making bandwidth a valuable resource for MANETs. Efficiency in using the bandwidth and energy resources and a carefully adjusted spatial reuse algorithm are some of the key criteria for the design of MANET algorithms.

Security (due to potentially hostile environments), quality of service requirements (due to demanding applications), and scalability can be counted among the other challenges in the design of a MANET.

1.2 Motivation and Goals

The previous section defined MANETs, their potential application areas, and the challenges in their design. In order to meet the demanding quality of service (QoS) and performance requirements, it is crucial for MANET protocols to adjust the utilization of bandwidth (a common resource) and energy (a node specific resource) according to the dynamic operating conditions. The overall efficiency of the system depends strongly on the careful adjustment of the resource usage in

all layers of the protocol stack, from the physical layer on up to the application layer.

The MAC protocol is the key element in the protocol stack that determines the ability of a wireless network to meet application requirements, since the MAC protocol has a direct impact on throughput, Quality of Service (QoS), energy dissipation, fairness, stability, and robustness [2, 3].

In particular, coordinated channel access schemes provide support for QoS, reduce energy dissipation, and increase throughput for low-to-mid noise levels and for dense networks [1]. MH-TRACE [4] and IEEE 802.15.3 [5] are examples of such coordinated protocols. The IEEE 802.15.3 protocol is specifically designed for high-rate and short range wireless personal area networks (WPANs) [6]. MH-TRACE is designed for mid-range medium-rate transmissions [4]. Both of those algorithms use a TDMA structure together with soft clustering of the nodes for channel access, as this approach has been shown to provide satisfactory performance in terms of QoS and energy dissipation.

Many of the protocol parameters in cluster-based protocols are set a-priori based on estimates of network conditions and based on a specific physical layer. TDMA parameters, which determine the amount of spatial reuse and interference, distribute the available bandwidth among clusters so as to reduce the interference throughout the network. Reducing the interference is a desirable goal since high interference leads to high error rates, decreasing the throughput as studied in [7]. However, reducing the available bandwidth per cluster also decreases the capacity per cluster. Non-uniform node distribution and node mobility may increase the local load above the cluster capacity, resulting in dropped packets and decreasing the throughput for real-time traffic [4]. The decision of how to set these parameters should thus be based on this trade-off and would be affected by various conditions such as node density and physical layer parameters. An analysis that describes the relationship between the protocol parameters and the performance metrics is

needed to ensure efficient use of the limited resources in MANETs, and we develop such a model in this dissertation.

The conditions in which a MANET operates may change over time. Non-uniform traffic loads are typical in MANETs due to intrinsic characteristics such as dynamically changing environment conditions and node mobility. The network designs should include dynamic channel allocation strategies in order to support non-uniform traffic. The objective of these strategies is to distribute the channel resources to the nodes that require channel access while taking the interference levels and spatial reuse into consideration. Although various dynamic channel allocation strategies have been proposed for other network types such as cellular networks, due to the specific characteristics of MANETs, these strategies are not directly applicable. In this thesis, we propose a dynamic channel allocation strategy that sets operating conditions on the fly for efficient resource utilization for MANETs. We further propose a cooperative load balancing algorithm for smoothing out the non-uniformity in the load distribution and combine it with the dynamic channel allocation strategy.

Data dissemination is another topic that is very important for reducing redundant usage of resources. The data generated in a MANET is oftentimes intended to be sent to more than one destination. One-to-many group communications are generally classified into two types: network-wide broadcasting and multicasting. In network-wide broadcasting the objective is to distribute the generated data to all the nodes in the network. However, the objective of multicasting is to deliver the data to a subset of the nodes in the network. In general, the overhead added to the packets in multicasting protocols is more than the overhead in network-wide broadcasting protocols. On the other hand, multicasting protocols prevent redundant transmissions on the parts of the network in which no multicast member resides. Investigating the trade-offs between multicasting and broadcasting in order to determine the conditions that make one of them preferable over the

other is needed to increase the efficiency and is discussed in this dissertation.

MANETs may operate in close proximity to other networks such as wireless sensor networks, cellular networks, or other MANETs. Optimizing networks internally, aiming to achieve individual objectives considering only individual network resources and using only local information about the network and ignoring co-located networks' resources and the effects the networks have on each other, results in sub-optimal overall performance. Symbiotic Networking, on the other hand, enables the mutual support of co-located networks through the symbiotic integration of otherwise independent networks. In symbiotic networking, networks not only can cooperate rather than compete in using common resources such as bandwidth but also help each other in routing the data following each network's individual goals. We examine how to exploit this cooperation in symbiotic networks in this dissertation.

Although simulations are efficient tools to comparatively evaluate the efficiency of the protocols, they cannot reflect many of the challenges for real implementation of these protocols, such as clock-drift, synchronization, imperfect physical layers, and interference from devices out of the system. Such issues may cripple a protocol that otherwise performs very well in software simulations. Thus, hardware implementation is essential for testing a protocol before any practical deployment. We focus on the implementation challenges of cooperation for a communication system and implement CDCA-TRACE protocol on a software defined radio platform.

It is clear that the efficient usage of the resources in MANETs is an important topic that is affected by various factors. This dissertation describes our analysis, design and simulations for MAC and Routing layer protocols that ensure efficient resource utilization in MANETs. My thesis is that a protocol architecture for MANETs that dynamically adapts to changing conditions based on cooperation and information sharing leads to more efficient use of the system resources com-

pared to competition based architectures. Contributions of these research efforts are summarized in the following section.

1.3 Contributions

This dissertation aims to achieve efficient bandwidth and energy utilization for MANETs and focuses specifically on the MAC and the routing layers. The contributions of my research include:

Medium Access Control Layer

- Developing a mathematical model that estimates the performance of soft clustering protocols for a set of parameters. This model describes the relationships between the defining operating conditions such as packet generation, node distribution, signal propagation and protocol parameters. The model provides estimates for performance metrics such as average packet drop probability, average number of transmitted packets, average number of collisions, average number of receptions, and average energy consumption. The values for the performance metrics obtained through simulation studies are compared to model results.

- Optimization of spatial reuse in soft clustering protocols for a given set of operating conditions. The degree of spatial reuse determines the trade-off between the number of collisions and the number of transmissions for a soft clustering protocol. We optimize the degree of spatial reuse in the MH-TRACE protocol for various operating conditions using the mathematical model for both maximum throughput and maximum energy efficiency. The optimal values are compared with simulation results.

- Predicting the performance variations of soft clustering protocols as the external conditions change. Typically, extensive simulation studies are used to

find the performance of a protocol under an arbitrary set of operating conditions. This approach requires excessive amounts of processing power and time. We use our mathematical model to predict the performance variations as the external conditions (such as data generation rate) vary.

- Proposing a cooperative load balancing algorithm that smooths the non-uniform load distribution among the clusters. While coordinated channel access protocols have been shown to be well suited for highly loaded MANETs, these protocols are in general not as well suited for non-uniform load distributions as uncoordinated channel access protocols. Based on MH-TRACE, we developed a MAC protocol, CMH-TRACE, in which the nodes in the network select their channel access provider based on the availability of the resources. CMH-TRACE improves the performance of MH-TRACE under non-uniform load distributions and can be integrated into the rest of the TRACE family of protocols for improved performance.

- Proposing a light weight dynamic channel allocation scheme based on spectrum sensing for cluster-based mobile ad hoc networks. Due to the dynamic nature of MANETs, the network has to be designed flexibly for dynamically changing conditions. We develop a dynamic channel allocation protocol, DCA-TRACE, based on MH-TRACE. DCA-TRACE improves the performance of MH-TRACE under non-uniform load distributions and can be integrated into the rest of the TRACE family of protocols for improved performance.

- Combining cooperative load balancing and dynamic channel allocation algorithms to provide better support for non-uniform load distributions. The problem of non-uniform load distribution is addressed by the cooperative load balancing and dynamic channel allocation algorithms from the perspective of the source nodes and the channel coordinators, respectively. We

combine both algorithms and design the CDCA-TRACE protocol that maximizes the improvements in the system.

Routing Layer

- Analyzing the trade-offs between data dissemination schemes for group communications. We compare a network-wide broadcasting protocol, NB-TRACE, and a multicasting protocol, MC-TRACE, in terms of their resource consumptions. We determine the conditions that make one of them preferable over the other one. Then, we combine unicasting, multicasting and network-wide broadcasting services in a single protocol called unified-TRACE, U-TRACE. U-TRACE chooses the most efficient data dissemination scheme depending on the requested service and network conditions.

- Exploring the benefits of symbiotic routing. The operating region of a MANET may intersect with the operating regions of other networks. We explore the potential performance gains and reduction in resource consumption of co-located networks that exploit hybrid nodes to enable network interactions that allow cross-network performance optimizations and cross-network information and service sharing. We use both simulation and mathematical frameworks that model the routing performances of both symbiotic and independent networking to quantify the routing benefits of symbiotic networking.

Practical Issues in Implementation

- Determining the challenges of implementing a MAC protocol on real hardware. Simulation studies do not accurately reflect many of the challenges encountered in real implementations such as limited processing power, clock drift, synchronization, imperfect physical layers, and cross band interference. We develop a reusable hardware framework to evaluate the performance of

wireless protocols, in particular the TRACE protocol for real-time communication in mobile ad hoc networks.

- Designing synchronization algorithms for cooperative mobile ad hoc networks. The limitations of the hardware platform and imperfect PHY layers make synchronization a non-trivial task. We propose two synchronization algorithms that address these issues. The algorithms enable implementation of TDMA based protocols and are used with the implementation of the TRACE protocol.

- Testing the TRACE implementation for packet losses. The operation of the TRACE protocol depends on the cooperation and control information exchange between the nodes in the network. On the other hand, packet losses in the system disrupt the availability of such information. We add packet loss compensation systems in the TRACE implementation to increase the robustness of the implementation against packet losses.

1.4 Thesis Organization

This thesis is organized as follows. Chapter 2 presents the related work on coordinated medium access control schemes used in MANETs, mathematical modeling of MAC performance, dynamic channel allocation strategies, data dissemination schemes and network symbiosis. After reviewing the literature in this area, we begin by exploring the effects of MAC layer decisions on resource utilization. In Chapter 3, we present a mathematical model that estimates the performance of soft clustering protocols, and we use that model for predicting the performance of the MH-TRACE protocol and for optimization of the spatial reuse according to the network conditions. Chapter 4 proposes a light weight dynamic channel allocation algorithm and a cooperative load balancing algorithm for cluster based

MANETs and incorporates these algorithms into the TRACE framework leading to the DCA-TRACE, CMH-TRACE and CDCA-TRACE protocols. We continue by exploring the effects of routing layer decisions on resource utilization. Chapter 5 presents U-TRACE, a protocol that combines unicasting, multicasting and network-wide broadcasting services in a single protocol that selects the appropriate data dissemination scheme depending on the requested service and network conditions. Chapter 6 explores the possible gains of network symbiosis for co-located networks that exploit hybrid nodes that allow cross-network performance optimizations. Then, we investigate the practical issues encountered in implementation in Chapter 7. Finally, Chapter 8 concludes the thesis and provides thoughts on future research in this area.

2 Related Work

2.1 Medium Access Control in MANETs

In wireless communications, the goal of the medium access control (MAC) protocol is to efficiently utilize the wireless medium, which is a limited resource. The effective use of the channel strongly determines the ability of the network to meet application requirements such as quality of service (QoS), energy dissipation, fairness, stability, and robustness [2], [3].

Based on the collaboration level, MAC protocols can be classified into two categories: coordinated and non-coordinated [8]. Channel access in non-coordinated protocols is typically based on a contention mechanism between the nodes. IEEE 802.11 [9] is an example of a non-coordinated protocol. Although it is easier to support non-uniform traffic with non-coordinated protocols, these protocols are unsuitable for highly loaded networks due to the contention mechanism. On the other hand, in coordinated channel access protocols, the medium access is regulated, making them better suited for networks where the network load is high. IEEE 802.15.3 [10], IEEE 802.15.4 [11], and MH-TRACE [4] are examples of such coordinated protocols. Coordinated channel access schemes provide support for QoS, reduce energy dissipation, and increase throughput for low-to-mid noise

levels and for dense networks. However, these protocols perform poorly under non-uniform traffic loads.

IEEE 802.15.3, IEEE 802.15.4 and MH-TRACE all manage the multiple access scheme through a TDMA structure, as this approach has been shown to provide satisfactory performance in terms of QoS and energy dissipation. MH-TRACE further uses a soft clustering approach where the clustering mechanism is utilized only for providing channel access to the member nodes. Hence, each node is capable of communicating directly with every other node provided that they are within communication range of each other. IEEE 802.15.3 and IEEE 802.15.4 only allow communication among the members of distinct clusters (piconets) in their peer to peer mode, while in star topology mode, nodes in distinct clusters can only communicate through their piconet controllers.

2.1.1 Clustering Approaches

Regardless of the partitioning scheme, the main consideration in forming clusters is the load distribution in the network. Clusters should be formed in such a way that they are able to meet the demand for channel access of the nodes in the cluster as much as possible. When the cluster is not able to meet the demand, either some of the transmissions are deferred (better suited for guaranteed delivery traffic) or the packets are dropped (better suited for best effort traffic). Thus, while designing a protocol or determining the performance of a specific protocol, the load distribution has crucial importance.

Clustering approaches may be classified as soft and hard clustering. In hard clustering approaches, such as GSM networks [12], nodes belong to the cluster in which they operate. Direct communication is only possible within the cluster. On the other hand, in soft clustering approaches such as MH-TRACE, nodes interact with clusterheads only to obtain channel access. There is no membership

relation between the nodes and the clusters. In other words, nodes are able to communicate directly with the nodes of other clusters and to choose the cluster from which they receive channel access. In general, soft clustering approaches are superior to hard clustering approaches in distributing the load evenly among clusters. On the other hand, soft clustering approaches tend to be more vulnerable to interference and collisions among co-frame clusters, since the boundaries of the clusters are not strict. In this dissertation, we consider soft clustering approaches.

Due to fading, two distinct transmissions may successfully operate over the same frequency, code and time range if they are well separated spatially. A successful protocol should employ this kind of spatial reuse for the sake of efficient use of the channel resources.

Clustering protocols, aim to maximize the distance between the clusters using the same portion of the channel. In cellular networks, the same set of frequencies may be assigned to cells (clusters) that are separated well enough depending on the frequency reuse factor employed [13, chap. 3]. Analogously, in MH-TRACE, each cluster operates in one of several frames separated in time. MH-TRACE has internal mechanisms that maximize the distance between clusters operating in the same frame (co-frame clusters) [4].

2.1.2 MH-TRACE Summary

In this work, we analyze the performance of soft clustering protocols to determine how to best set their parameters for efficient use of the channel resources. Specifically, we analyze the MH-TRACE protocol. Here we briefly explain the clustering mechanisms of MH-TRACE. A detailed description of MH-TRACE is available in [4].

In MH-TRACE, time is divided into superframes of equal length, as shown in Fig. 2.1, where the superframe is repeated in time and further divided into frames.

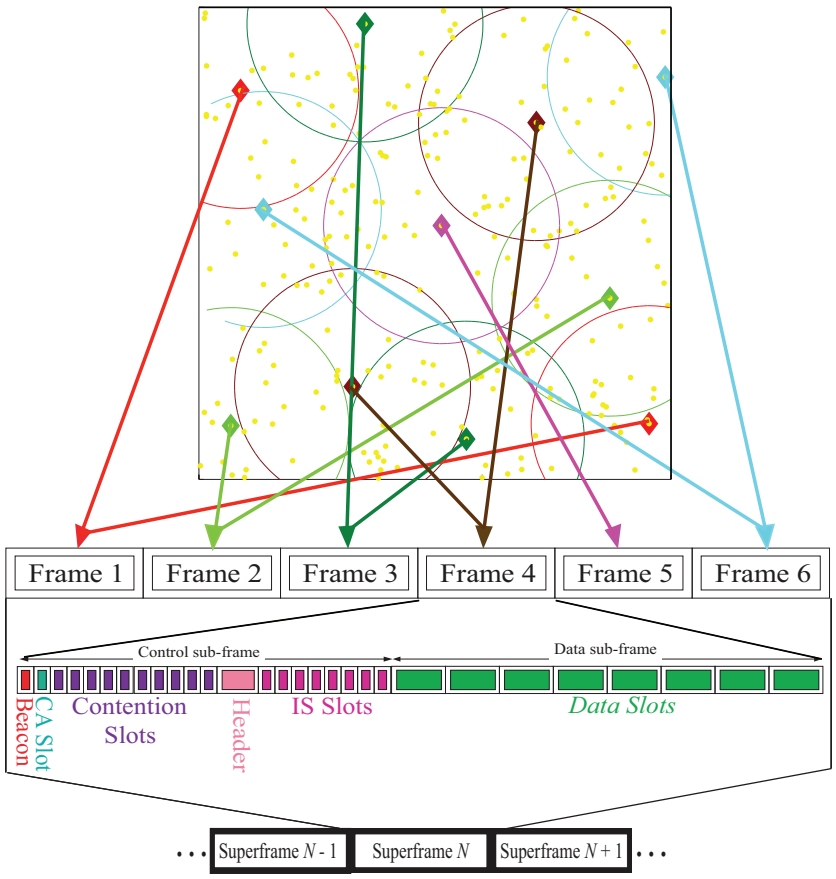


Figure 2.1: A snapshot of MH-TRACE clustering and medium access. Diamonds represent selected clusterheads and dots represent the nodes in the network. CH-frame matching, together with the contents of each frame, is depicted.

There are randomly chosen clusterheads that regulate the channel and provide channel access for the nodes in their communication range. Each clusterhead (CH) operates using one of the frames in the superframe structure. There is also a spatial reuse mechanism that allows more than one CH to operate in the same time frame provided that the interference is low.

Each frame in the superframe is further divided into sub-frames. The control sub-frame constitutes the management overhead. Beacon, cluster announcement (CA), and header slots of the control sub-frame are used by the CHs, whereas

contention slots and information summarization (IS) slots are used by the ordinary nodes.

At the beginning of the frame, the CH announces itself to the nearby nodes by sending a beacon message in the beacon slot of the control sub-frame. The CA slot is used for interference estimation for CHs operating in the same frame (co-frame CHs). During the CA slot, the CH transmits a message with a given probability and listens to the medium to calculate interference caused by other CHs operating in the same frame. Contention slots are utilized by the nodes to pass their channel access requests to the CH. A node that wants to access to the channel selects a contention slot randomly among the contention slots and sends a contention message in that slot. After listening to the medium during the contention slots, the CH becomes aware of the nodes that request channel access and forms the transmission schedule by assigning available data slots to the nodes. After that, the CH sends a header message that includes the transmission schedule that will be followed for the rest of the frame.

There are an equal number of IS slots and data slots in the remainder of the frame. During the IS slots, nodes send short packets summarizing the information that they are going to be sending in the order announced in the Header. By listening to the relatively shorter IS packets, nodes become aware of the information that are going to be sent and may choose to sleep during the corresponding data slots if they are not interested in (or the recipient of) the data.

2.1.3 Mathematical Modeling of MAC Performance

The most direct approach to determine the MAC performance is to obtain samples of field measurements on the performance metrics [14, 15]. However, the difficulty in implementation on real hardware and taking a large set of field measurements make this method impractical for most cases, and not the best approach in the

protocol design stage. It is easier and more convenient to implement a protocol on a simulation platform. Thus, simulation studies are the most widely used methods to evaluate the performance of protocols [16–18]. However, it is impractical to determine the performance of a protocol for large sets of conditions as simulations require excessive amounts of processing power and time. Analytical models are the most suitable tools to obtain insight into the performance of a MAC protocol.

Various analytical studies of protocol performance exist in the literature. These studies range from detailed protocol specific models to more general models that can be applied to a group of protocols.

One line of research is focused on more generalized analysis that covers a set of protocols that share a certain property. These studies use general assumptions and focus on certain descriptive performance metrics rather than trying to model the details of a certain protocol. For example, the authors in [19–21] study the optimal size of a cluster for cluster based data aggregation schemes without going into the protocol dependent parameters. Studies on the delay and the throughput characteristics of TDMA/CDMA MAC protocols, such as those described in [22, 23], also fall under this category.

Protocol specific studies approximate the protocols under concern in detail using protocol dependent assumptions. IEEE 802.11 is one of the most extensively studied protocols. Bianchi et al. [24] present a simple analytical model to compute the saturation throughput performance of the IEEE 802.11 Distributed Coordination Function. Later on, more detailed studies on IEEE 802.11 have been proposed, based on the protocol and its extensions [25–27] such as IEEE 802.11e, which has several quality of service (QoS) enabling features to support real time traffic.

Being one of the most widely used network types, GSM networks have also attracted considerable attention, with most of these works analytically deriving the performance of GSM networks. For example, in [28], a model for cell dimensioning

and performance evaluation is presented.

IEEE 802.15.4 is another widely studied protocol. The very first analytical study of IEEE 802.15.4 was described in [29] in the context of medical sensor body area networks. Various other papers [30–33] have been published since then, having different sets of assumptions and different operation modes of the protocol. For instance, in [30] the analysis is focused on the star topology, while in [31], a peer to peer beacon-enabled cluster-tree structure is considered. Similar analytical studies for IEEE 802.15.3 can be found in the literature, such as the ones in [34, 35].

In Chapter 3, we use an approach that lies in between these two lines of research for analyzing protocol performance. Even though we focus on a specific cluster based MAC protocol utilizing soft-clustering, namely MH-TRACE, the assumptions used in the analysis make the generalization of the model to other MAC protocols that employ a TDMA structure possible.

2.1.4 Dynamic Channel Allocation Strategies

The responsibility of the MAC layer is to coordinate the nodes' access to the shared radio channel, minimizing conflicts. In a multi-hop network, obtaining a high bandwidth efficiency is only possible through exploiting channel reuse opportunities. Indeed, efficient utilization of the common radio channel has been the center of attention since the early development stages of wireless communication [36].

Cidon et al. [37] present a distributed dynamic channel allocation algorithm with no optimality guarantees for a network with a fixed a-priori control channel assignment. Alternatively, there are various game-theoretic approaches to the channel allocation problem in ad hoc wireless networks [38, 39]. Gao et al. [38] model the channel allocation problem in multi-hop ad hoc wireless networks as a

static cooperative game, in which some players collaborate to achieve a high data rate. However, these approaches are not scalable, as the complexity of the optimal dynamic channel allocation problem has been shown to be NP-hard [40–43].

In a multi-hop wireless network, CSMA [44] techniques enable the same radio resources to be used in distinct locations in a network, leading to increased bandwidth efficiencies at the cost of possible collisions due to the hidden terminal problem [45]. Different channel reservation techniques are used to tackle the hidden terminal problem. Karn et al. [46] use an RTS/CTS packet exchange mechanism before the transmission of the data packet. 802.11 distributed coordination function (DCF) uses a similar mechanism but adds an ACK packet indicating the successful reception of the packet. Although this handshake reduces the hidden node problem, it is inefficient under heavy network loads due to the exposed terminal problem. Several modifications to the RTS/CTS mechanisms have been proposed to increase the bandwidth efficiency [47, 48] including use of multiple channels such as [49–51].

However, these approaches attempt to solve the problem of channel assignment when there is a single intended destination of each transmission and do not cover group communication. We are interested in MANET scenarios where the destination of the generated packet is not a specific node in the local neighborhood but all the nodes in the immediate neighborhood of the transmitter. Such a scenario is only covered with 802.11 DCF basic access mode where the RTS/CTS mechanism is disabled.

In coordinated MAC protocols, channel assignment is performed by channel coordinators. Channel reuse is incorporated into the system through use of the same channel by coordinators that are spatially separated. The cellular concept [52] that regulates channel access through fixed infrastructure called base stations also forms the basis of the widely deployed GSM systems [53].

The types of strategies for on-demand dynamic channel allocation used in

cellular systems can be divided into two categories: centralized and distributed schemes. In centralized dynamic channel allocation schemes [54], the available channels are kept in a pool and distributed to various cells by a central coordinator. Although quite effective in maximizing channel usage, these systems have a high overhead and cannot be applied to MANETs. Distributed dynamic channel allocation for cellular networks has also been studied extensively [55–57]. In distributed dynamic channel allocation, each cell is assigned a number of channels. These channels can be exchanged among adjacent cells through message exchange mechanisms between the channel regulators (cell towers) in an on demand basis. This approach, too, is not directly applicable to MANETs. Unlike in the cellular case, in MANETs, the message exchanges between the channel regulators also consume network resources. Due to node mobility and the dynamic behavior of the network, the large overhead associated with the frequent message exchanges overwhelms the network and decreases the bandwidth efficiency.

Dynamic channel allocation [55–57] and channel handoff [58] algorithms dynamically adjust the number of channels accessed by base stations proportional to their load and help keep bandwidth efficiency and service rate high simultaneously even under non-uniform load. However, such algorithms either require centralized control or frequent message exchanges between base stations and thus are not directly applicable to MANETs.

Load balancing has also been studied within the context of heterogeneous networks. In the case of excess demand, part of the network load can be offloaded to other networks using heterogeneous gateway nodes. Song et al. [59] present a policy framework for such resource management in a loosely coupled cellular/WLAN integrated network.

Although dynamic channel allocation and channel handoff are studied extensively within the context of cellular networks, they have not been studied much in the context of MANETs, where the bandwidth efficiency and load balancing

are mostly studied at the network layer [43, 60, 61]. Wu et al. [60] extend the AODV protocol to include a distributed system to infer the network status and to optimize routes considering bandwidth efficiency and stability. A centralized load aware joint channel assignment and routing algorithm is proposed in [43].

At the MAC layer, Tseng et al. [62] propose a location aware dynamic channel allocation scheme for MANETs. However, their protocol mandates that location information be provided to each node. Gireesan et al. [63] study the capacity of the IEEE 802.15.4 protocol for linear and grid topologies and calculate the optimal channel assignment yielding the maximum possible channel reuse. However, the results are not generalizable to the complex and dynamic topologies of typical MANETs. *Primary Collision Avoidance* type channel allocation algorithms [64–67] assign channels to the nodes one by one, mitigating the conflict relationships in a connection graph at each iteration. Finally, Chowdhury et al. [68] propose a dynamic channel allocation scheme for IEEE 802.15.4 systems using a single hop overlay weight-based clustering structure. Although the proposed system reduces the message exchanges over previously built *Primary Collision Avoidance* algorithms, the proposed system is entirely message driven and requires the construction of clusters. Also this system is susceptible to topology changes during the channel allocation phase.

To the best of our knowledge, our work in Chapter 4 is the first attempt to solve the dynamic channel allocation problem solely based on carrier sense measurements (i.e., spectrum sensing), greatly reducing the overhead. We analyze the dynamic adaptability of the channel allocation and non-uniform load distribution problem from both the perspective of the clusterheads and the member nodes. We introduce two algorithms, a dynamic channel allocation and a collaborative load balancing algorithm, and adopt these algorithms to the TRACE framework. By combining the dynamic channel allocation and collaborative load balancing algorithms, we propose the CDCA-TRACE protocol that has the highest bandwidth

efficiency among the TRACE family of protocols.

2.2 Routing in MANETs

2.2.1 Group Communications

Group communications is essential for many applications in mobile ad hoc networks, including supporting electronic classrooms, tactical military communication, and communication in disaster recovery missions. One-to-many group communications are generally classified into two groups: network-wide broadcasting and multicasting. In network-wide broadcasting the objective is to distribute the generated data to all the nodes in the network. The most basic network-wide broadcasting approach is simple flooding, where at each node the received data is retransmitted with the aim of reaching all nodes in a connected network. There are several more efficient network-wide broadcasting schemes that increase the efficiency by reducing the number of redundant retransmissions and/or collisions. On the other hand, the objective of multicasting is to deliver the data to a subset of the nodes in the network. By using a data dissemination structure, multicasting protocols limit the diffusion of the data to a certain subset of the entire network, namely to the multicast members. The data dissemination scheme used in multicasting protocols range from tree based routing strategies where the redundant transmissions are eliminated to mesh-based routing strategies that cope with frequent link breakages by controlled addition of redundant links. The authors in [69] and [70] present recent surveys on various group communication protocols.

There is additional overhead incurred in multicasting protocols compared to broadcasting protocols. In certain scenarios, the cost of collecting and processing the additional information overwhelms the gains in limiting the data dissemination structure to the multicast members. As one might expect, in scenarios where

the majority of the nodes are part of the multicast group, one can increase the efficiency by using a broadcasting protocol instead of using a multicasting protocol. In this dissertation, our objective is to investigate the trade-offs between multicasting and broadcasting in order to determine the conditions that make one of them preferable over the other.

There are similar studies that point out the trade-offs between multicasting and broadcasting. Researchers in [71] compare a selected multicast protocol, namely on demand multicast routing protocol (ODMRP), with a selected network-wide broadcast protocol, namely scalable broadcast algorithm (SBA). The authors report that while multicasting is preferable for small group sizes, as the group size increases, broadcasting becomes more efficient. However, the protocols considered in this work are quite different in nature. While the aim of SBA is to minimize the number of redundant transmissions, ODMRP follows a mesh-based approach in which redundant routes are created intentionally. Thus the comparison between those protocols does not provide a full understanding of the trade-offs between multicasting and broadcasting.

Towards the goal of investigating the trade-off between multicasting and broadcasting, we perform extensive simulation studies on a chosen protocol from each class: Network-wide broadcasting through time reservation using adaptive control for energy efficiency (NB-TRACE) [72] for broadcasting; and Multicasting through time reservation using adaptive control for energy efficiency (MC-TRACE) [73] for multicasting. The first reason for choosing these protocols is that they have been shown to outperform many other protocols in their class [72, 73]. Moreover, these protocols are built on top of the same MAC structure, and thus their sensitivity to MAC layer issues such as mobility and link errors is similar. Finally, the data maintained by the protocols are very similar to each other, and any additional burden of multicasting can be directly observed. Consequently, the protocols can be combined into a unique framework and coexist simultaneously. Ultimately,

this approach yields a unified protocol where the better approach (broadcasting or multicasting) can be used depending on the situation.

2.2.2 Group Communications in the TRACE Family of Protocols

The main purpose of this section is to give the reader insight into the differences between NB-TRACE and MC-TRACE. The details of the protocols, NB-TRACE and MC-TRACE, can be found in references [72] and [73], respectively.

Both MC-TRACE and NB-TRACE are cross layer approaches where the MAC layer and the routing layer functionalities are implemented together in a unique framework. The MAC scheme of the protocols follows from MH-TRACE [4], where the network is organized into overlapping clusters, each managed by a clusterhead (CH).

NB-TRACE

Routing in NB-TRACE makes use of the clustering structure. The protocol sends a copy of each data packet to all of the CHs, and the CHs retransmit these packets to their cluster members. Each data session starts with an initial flooding stage where each rebroadcasting node implicitly acknowledges its upstream node through IS packets as a part of its transmission. In the case of the existence of more than one upstream node, only one of them is selected and announced in the downstream node's IS packet. A node drops its relaying status and stops retransmitting the packets when it does not receive an acknowledgement for a certain amount of time. Only the CHs keep retransmitting the packets even when they do not receive any downstream acknowledgement. This behavior prunes the redundant retransmissions and creates a tree that starts from the source node and ends at the CHs. Since the CHs form a dominating set, this ensures that

once all the CHs transmit the message, all nodes in the network will receive the message. The dynamic behavior of the network is handled by a local branch repair mechanism.

MC-TRACE

MC-TRACE implements multicast routing on top of MH-TRACE using a mixed layer approach. Like NB-TRACE, MC-TRACE also starts with an initial flooding stage. Nodes that do not receive a downstream acknowledgement stop retransmitting. However, in MC-TRACE, CHs do not take a special role in routing. Instead, the member nodes keep sending an acknowledgement to their upstream node even when they do not receive any downstream acknowledgements. Therefore, the tree is kept alive directly by the group members.

Furthermore, in MC-TRACE, retransmitting nodes also choose and announce a downstream node in addition to their upstream node. The first node that sends an upstream acknowledgement is selected as the downstream node and announced in the following transmissions. The node that is announced as the downstream node is responsible for sending upstream acknowledgements and keeping the branch alive. With the help of this mechanism, in the case of more than one leaf member node receiving the data from the same branch, only one of them sends the acknowledgement messages. Although this mechanism eliminates redundant acknowledgements, the need for acknowledgements from the leaf nodes makes MC-TRACE consume considerably more resources compared to NB-TRACE when the multicast group members are spread throughout the region.

2.3 Network Symbiosis

There is a large diversity in recent communication devices, in terms of the purpose of the devices, the applications using the devices, and the networks the

devices support. Indeed, it is not uncommon to see devices supporting multiple networks [74, 75] and several of these networks co-existing in the same physical environment [76–78]. However, currently network optimizations are mostly done internally: aiming to achieve individual objectives considering only individual network resources and using only local information about the network, ignoring co-located networks' resources and the effects the networks have on each other. Optimizing networks with such an individual focus generally results in sub-optimal overall performance.

Recently, researchers have devised a paradigm to enable the mutual support of co-located networks through the symbiotic integration of otherwise independent networks, referred to as *Symbiotic Networks* [79–81]. This paradigm defines cooperation of networks to optimize individual network performances by cross-network sharing of resources, information and services. The shared information, such as channel state and congestion information, can be used to optimize MAC level parameters, whereas sharing network resources allows cross-network relaying and distributed processing. Clearly these different methods of cooperation can benefit both networks, enabling the sharing of node resources as well as local cooperation rather than competition for scarce network resources such as bandwidth [80].

There are several scenarios that motivate the need for symbiotic networking. For example, consider the scenario described by Poorter et al. in [80], for a Wireless Body Area Network (WBAN) consisting of physiological sensors to monitor heartbeat, body temperature and motion of a person. The person's WBAN can interact with another network to get warnings on air pollution. When the person is in his vehicle, his WBAN and the vehicle's network can integrate so that the vehicle can monitor the status of the driver using the WBAN sensors, and can alert the driver along with the surrounding vehicles if the driver falls asleep. The vehicle's network can detect the WBAN of pedestrians nearby, to reduce the chance of accidents. If the WBAN detects that the person is having a heart at-

tack, the network could automatically interact with the cellular network via his cell phone and call an ambulance. In the ambulance, the WBAN can integrate with the ambulance network, and when the patient arrives in the emergency room, the WBAN can interacts with the hospital network, enabling sensor monitoring information to be displayed on screens in the emergency room.

Symbiotic networking aims to achieve mutual and full integration of distinct networks that are defined by different physical layer communication abilities. The integration of wireless and wired networks has been studied and is still being studied in the community for specific networks. The solutions proposed are mainly network specific adaptations and not embracing the mutual integration of generic networks [76,82–84]. For instance, Wireless LAN (WLAN) and cellular network integration studies concentrate on the extension of cellular network coverage through WLAN access points [85–87], and other work on throughput capacity assumes that base stations are connected by a high-bandwidth wired network and act as relays for ad hoc wireless nodes [88, 89]. However, all these studies either investigate one-way integration (i.e., using the WLAN to extend cellular network coverage), or they consider single or two-hop networks and provide network-specific solutions for such integration.

The IEEE 802.21 standard has the goal of supporting handovers between specific network types, specifically cellular, WiFi, Bluetooth and IEEE 802.16. This standard provides tools for packet switching between networks. However, how to benefit from such multiple network environments is still an open issue that is not addressed by this standard.

Symbiotic Networks were first conceptually presented in [90]. Gedge proposes nodes in ad hoc networks to use other nodes' resources to extend the service areas of WLAN access points. In [81], symbiosis of a sensor network deployed along a highway with a vehicular ad hoc network is proposed. The data is transferred from the sensor network to the vehicular ad hoc network via a specific gateway. The

effect of the vehicle speeds on the overall throughput and delay are investigated. Finally, in [80], a detailed description of symbiotic networks is given along with several cases where it can be used. As a result, although the integration of different networks is not a recent research area, past studies concentrate on specific network tuples, and there are no generic quantitative models proposed to show potential performance gains with network-level symbiosis to integrate co-located networks, which is the aim of this dissertation.

3 Analytical Performance of Soft Clustering Protocols

3.1 Introduction

Effective resource utilization is particularly important in MANETs from the perspective of both the bandwidth and energy efficiency and is strongly related to the protocol used at the medium access control (MAC) layer. Hence, we begin our investigation of resource utilization at the MAC layer.

It is important not only to choose the correct protocol but also to adjust the parameters in accordance with the application requirements. Many of the parameters in cluster-based protocols are set a-priori based on estimates of network conditions and based on a specific physical layer. The true relationship between the parameters and the protocol performance can only be determined by analysis. Although simulation studies reveal the performance of a protocol for a certain set of conditions, the statistical accuracy of the simulation results is questionable unless repeated extensively. For large and dense networks, this approach requires excessive amounts of processing power. Moreover, results obtained from those studies are only valid for the selected parameters and do not reveal the full impact of these parameters on the performance of the protocol.

In order to address this problem, we have developed an analytical model that

reflects the relationships between protocol parameters and the overall performance of the protocol under different network conditions for a TDMA-based clustered protocol, MH-TRACE. Specifically, we develop a model that relates the TDMA frame parameters (number of slots per frame and number of frames per superframe) and the node density to the expected number of dropped packets and the expected number of collisions. This model enables us to find the set of parameters that maximize overall throughput or energy efficiency for TDMA-based clustered protocols such as IEEE 802.15.3 (peer to peer mode), IEEE 802.15.4 (peer to peer mode) and MH-TRACE. We use this model to analyze the MH-TRACE protocol.

This chapter is organized as follows. An analytical model that estimates the performance of soft clustering protocols for a given set of parameters is presented in Section 3.2. Section 3.3 discusses the validity of the analytical model by comparing the analytical results with those found via simulations for MH-TRACE. Using the analytical model, the degree of spatial reuse present in a network utilizing the MH-TRACE protocol is optimized for both maximum throughput and minimum energy consumption per generated packet in Section 3.4. In Section 3.5, the performance of the protocol is investigated as the transmission power and the data source model is varied. Section 3.6 summarizes the chapter with final comments.

3.2 Analytical Model

In this section, an analytical model for the MH-TRACE soft clustering protocol is presented [8, 91]. The model is simple enough to be evaluated for a large number of parameters and provides estimates of simulation averages, eliminating the need for complicated simulation studies.

As mentioned in Section 2.1.1, clustering protocols divide the available channel into a number of partitions and assign regulatory nodes, namely CHs, to each partition forming the clusters in the network. Each partition may be used by

more than one cluster depending on the spatial reuse factor. Using a high spatial reuse factor leads to having a smaller number of larger channel partitions and in turn a higher amount of concurrent traffic that can be supported per cluster, which decreases the number of dropped packets. On the other hand, since there are more clusters using the same partition simultaneously, the interference and the collisions increase.

In terms of throughput, the performance of a protocol is limited both by the dropped packets and by the collisions. Therefore, these effects must be taken into account in performance estimation.

3.2.1 Dropped Packets

In real-time communications, packet timing is one of the most critical factors. It is preferable to discard packets that are not transferred in a timely fashion. Hence, packets become obsolete after a predetermined amount of time and must be dropped from the transmit queue.

Any load over the maximum amount of concurrent traffic that can be supported, $MaxTraffic$, will eventually be dropped. The probability of dropping a packet can thus be calculated as

$$P_{dp} = \max\left(\frac{Load - MaxTraffic}{Load}, 0\right). \tag{3.1}$$

Analysis can most easily be performed on a per cluster basis, as the amount of traffic that can be supported by a CH is well defined. Hence, using the TDMA frame as the time unit, $Load$ and $MaxTraffic$ can be defined as the number of nodes that require channel access within a cluster per frame and the number of data slots available per frame, respectively.

$MaxTraffic$ directly follows from the parameters of the protocol and hence is deterministic. On the other hand, $Load$ is a stochastic process that depends on

various random variables. Since $MaxTraffic$ is known, calculating P_{dp} reduces to calculating the distribution of $Load$ for the clusters.

The load in the cluster is mainly determined by three probabilities:

- p_s: Probability of a node to be generating data,
- p_c: Probability of a node to be in the communication range of a CH.
- p_d: Probability of a node that is in the communication range of a CH to choose that CH as its channel access provider.

In this chapter, we assume that the network is supporting voice traffic. We further assume that each node generates independent voice packets according to a common voice traffic model where nodes are either in "spurt" or "gap" duration, with voice data being generated in the "spurt" duration and no data being generated in the "gap" duration [92] [93] [94]. Hence, p_s accounts for the fact that nodes will only have data to send when they are in the "spurt" duration.

Since we consider a soft clustering approach, nodes can access the channel from any CH within range. Therefore, if one CH has no available bandwidth, a node can request channel access from another CH in range. Hence, p_d takes into account the fact that a node's expected load to a cluster is distributed among those CHs in the communication range of that node.

Given these distributions and the number of nodes in the network, N, the load per cluster per superframe for a given CH is a binomial distribution with a success probability of $p_{dn} = p_s p_c p_d$ as

$$Pr(Load = k) = \binom{N}{k} {p_{dn}}^k (1 - p_{dn})^{(N-k)}. \tag{3.2}$$

Probability of a Node to be in Spurt Duration (p_s)

We assume a voice model with exponentially distributed spurt and gap durations (T_{spurt} and T_{gap}, respectively) that occur one after the other [92] [93] [94]. The

probability of finding a node in the spurt state, p_s, is calculated as

$$Pr\,(\text{Node } i \text{ in spurt}) = \frac{E[T_{spurt}]}{E[T_{spurt} + T_{gap}]} = p_s. \tag{3.3}$$

The mathematical model can be used with any packet generation model as long as this probability of a node generating data, p_s, can be calculated.

Probability of a Node to be in the Communication Range of a CH (p_c)

Nodes are assumed to be i.i.d. distributed according to a uniform distribution. Thus, the probability of a node to be in any given region is

$$Pr(\text{Node in Region}) = \frac{\text{Area of Region}}{\text{Simulation Area}}. \tag{3.4}$$

The coverage region of a cluster is assumed to be a circle centered at the CH's location having a radius equal to the maximum reliable communication distance at the selected power level. Part of a CH's coverage region may lie outside the network area depending on the CH's location, CH_{loc}. The network area can be divided into 3 regions, as shown in Fig. 3.1. The coverage area that lies within the simulation area, α_1, α_2, and α_3 for a CH in regions 1, 2 and 3, respectively, are calculated in [7] and repeated here for reference.

$$\alpha_1 = \pi r^2, \tag{3.5a}$$

$$\alpha_2 = \pi r^2 - 2I(d_0), \tag{3.5b}$$

$$\alpha_3 = \frac{3}{4}\pi r^2 - I(x_0) - I(y_0) + x_0 y_0, \tag{3.5c}$$

where x_0 and y_0 are the distances to the closest vertical (for x_0) and horizontal (for y_0) limits of the network area, $d_0 = min(x_0, y_0)$ and the function $I(x)$ is as given in [7].

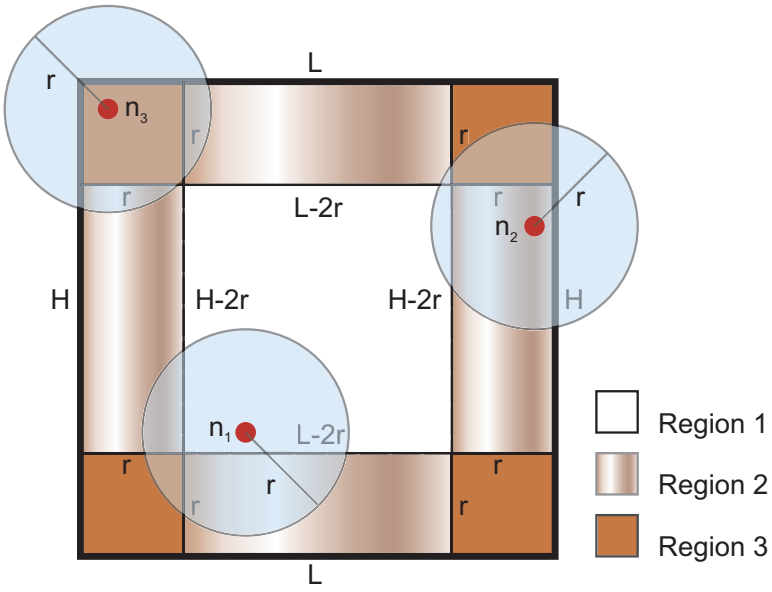


Figure 3.1: Partitioning of the simulation region. Part of the coverage region lies outside the network region for CHs located in regions 2 and 3 (adopted from [1]).

Replacing "Area of Region" with the appropriate term from (3.5), and averaging over the entire network area leads to the desired p_c for a given CH. The averaging process is done by taking a numerical integral over the region and dividing by the simulation area.

The expected number of nodes within the communication range, r_c, of a CH (and for any node) can be calculated using p_c as in (3.6).

$$N_c = Np_c \tag{3.6}$$

Probability of a Node to Choose the CH as the Channel Access Provider (p_d)

The entire simulation area will be fully covered for sufficiently large node densities. After that point, variations in the node density do not alter the expected number of CHs in the network and their positions. Thus, the number of CHs in the

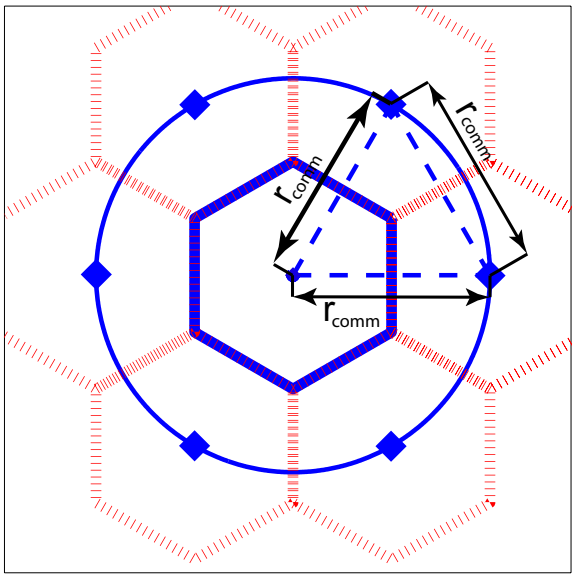


Figure 3.2: Alignment of CHs under maximum packing conditions.

communication range of each node is independent of node density for sufficiently large node densities. The easiest way to obtain this number is to simulate the CH selection mechanism of the protocol under concern for a sufficiently large node density. For MH-TRACE, protocol simulations indicate that 28% of the nodes are in the communication range of only one CH whereas 52%, 19%, and 1% of the nodes are in the communication range of two, three, and four CHs, respectively.[1] Nodes can select their channel access provider uniformly among the CHs that are in their communication range. Therefore, the expected load of a node to a CH, p_d, can be calculated as given in (3.7). Using the aforementioned values, for MH-TRACE, p_d is calculated as 0.6058.

$$p_d = \sum_{k=1}^{\infty} \frac{Pr(\#\text{ of CHs in } r_c \text{ range} = k)}{k} \tag{3.7}$$

Substituting p_s, p_c and p_d into (3.2), the statistical distribution of *Load* can

[1]This assumption holds as long as the network area is much larger than the area of a circle with a radius equal to the communication range. For the same simulation area (1000mx1000m), we have observed that changing r_c from 150 to 350 does not alter those values.

be obtained for a given CH. P_{dp} can be obtained using the distribution of *Load* and $P_{dp} = \max\left(\frac{Load - MaxTraffic}{Load}, 0\right)$.

3.2.2 Collisions

CHs manage the channel access within the cluster so as to prevent transmissions using the same frequency, code and time range. For frequency division networks, CHs prevent nodes from transmitting in the same frequency band while nodes are allowed to transmit simultaneously, whereas for TDMA based protocols like MH-TRACE, nodes can transmit using the same frequency band but are not allowed to transmit data simultaneously. Hence, data collisions within the cluster are completely eliminated. Collisions among nodes of clusters operating in disjoint parts of the channel are also not possible. The only remaining source of collision is the clusters operating in the same partition of the channel. For MH-TRACE, this corresponds to co-frame clusters that operate on the same frame within the superframe.

Our approach for calculating the number of collisions per superframe consists of two steps. The first step is to relate the number of frames per superframe (N_f) to the co-frame CH separation, and the second step is to relate the co-frame CH separation to the number of collisions per superframe.

Relation Between N_f and Co-frame CH Separation

As stated in Section 3.2.1, for large node densities, the network is fully covered by the CHs. The clustering structure and the number of clusters in the network become independent of the node density. Thus, the average separation between two neighboring CHs, d_{CH}, is independent of the node density and N_f. However, depending on the number of distinct clusters possible, which is equal to N_f, the separation between the co-frame CHs, d_{co-CH}, varies.

The number of collisions increase as d_{co-CH} decreases. Due to the CH creation mechanism, the minimum separation between two CHs is equal to the communication distance, r_c, in the worst case scenario. Under maximum packing conditions, a third CH is also separated by r_c from both of the previous CHs, as shown in Fig. 3.2. The lines connecting the 3 CHs form an equilateral triangle. At most 6 CHs can be located around the first CH, as shown in Fig. 3.2. Considering an imaginary cell boundary passing perpendicular to the lines connecting the neighboring CHs, a hexagonal cellular structure can be constructed in the form of the standard structure used in cellular systems [52]. The labeling structure that is used in cellular systems is applicable for the clustering structure under concern. Distinct labels correspond to clusters operating in distinct frames in the superframe structure. The separation between co-labeled cells and hence co-frame CHs is given as

$$d_{co-CH} = r_c \times \sqrt{N_f}. \tag{3.8}$$

We use this separation as a basis to calculate the expected number of collisions, even though a regular structure that maximizes the distance between co-frame CHs exists only for a certain set of N_f values. The worst case assumption can be relaxed by replacing r_c in (3.8) by d_{CH} leading to (3.9).

$$d_{co-CH} = d_{CH} \times \sqrt{N_f} \tag{3.9}$$

Relation Between Co-frame CH Separation and Number of Collisions

We begin the analysis by deriving a function, $f(d, r_1, r_2)$, that calculates the intersection area of two circles having radii r_1 and r_2, and whose center points are separated by d. This function will be used frequently in the later analysis. Fig. 3.3 shows two such circles. The intersection area, which is shown as the shaded area

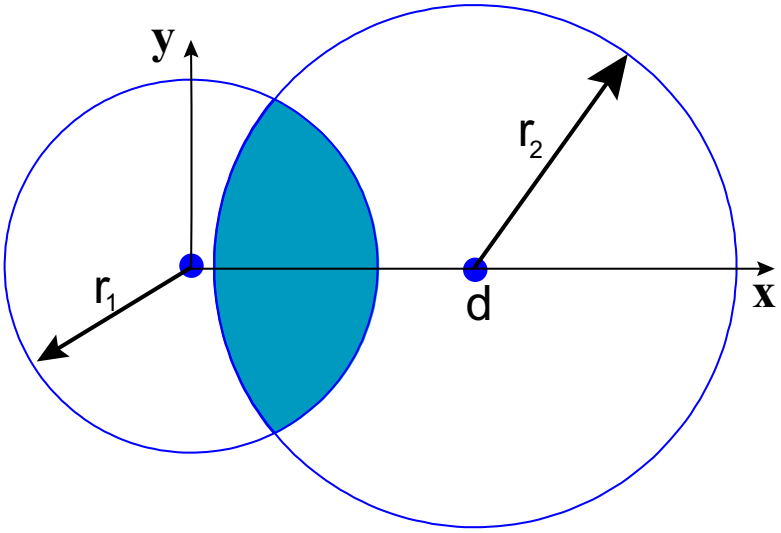


Figure 3.3: Intersection region for two circles separated by d with radii r_1 and r_2 respectively.

in the figure, can be calculated as

$$x_{co} = (r_2^2 - r_1^2 + d^2)/(2 * d) \tag{3.10}$$

$$\begin{aligned} f(d, r_1, r_2) &= 2 \int_{d-r_2}^{x_{co}} \int_{0}^{\sqrt{r_2^2-(x-d)^2}} dydx + 2 \int_{x_{co}}^{r_1} \int_{0}^{\sqrt{r_1^2-(x-d)^2}} dydx \\ &= 2r_1^2 \left(\frac{\sin(2\theta)}{4} + \frac{\theta}{2} \Big|_{\theta=\arcsin\left(\frac{d-x_{co}}{r_1}\right)}^{\theta=\frac{\pi}{2}} \right) + 2r_2^2 \left(\frac{\sin(2\theta)}{4} + \frac{\theta}{2} \Big|_{\theta=\arcsin\left(\frac{x_{co}}{r_2}\right)}^{\theta=\frac{\pi}{2}} \right). \end{aligned} \tag{3.11}$$

Nodes vulnerable to collisions should be located at most 2 times the communication radius, r_c, from a CH in order to receive a packet in that cluster's frame. Furthermore, those nodes should also be located within the sum of the communication radius and the interference radius, $r_c + r_{int}$, of a co-frame CH. Thus, the vulnerable region for two co-frame CHs is the area of intersection of two circles with radii of $2r_c$ and $r_c + r_{int}$ and centers located at the co-frame CH positions. The lightly shaded area in Fig. 3.4 depicts the vulnerable region for two CHs

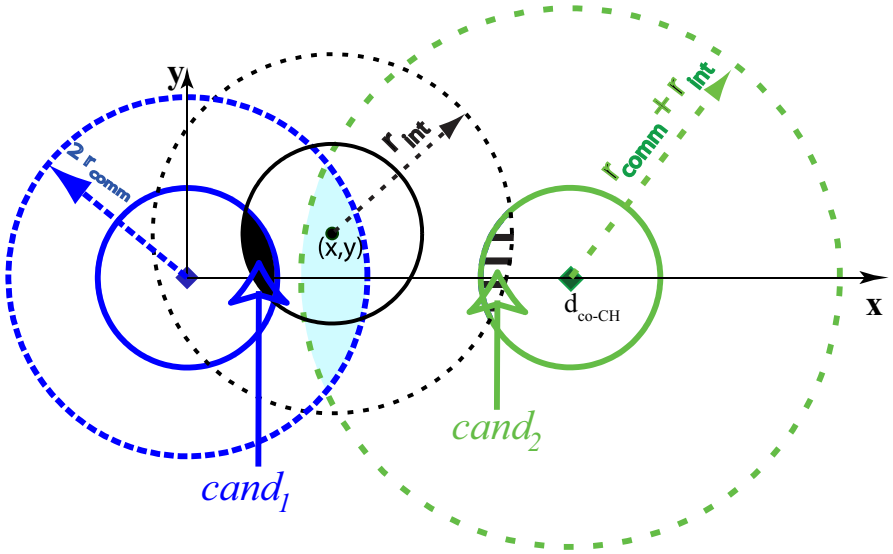


Figure 3.4: A typical alignment of co-frame clusters together with their vulnerable region (lightly shaded region). The dark shaded regions near the CHs are the regions of member nodes whose packets could collide at the node located at (x, y). The expected numbers of nodes in these regions are represented by $cand_1$ and $cand_2$, respectively.

(represented by the diamonds) separated by a distance of d_{co-CH}. The nodes in the vulnerable region are called vulnerable nodes.

The nodes whose packets may collide at a node in the vulnerable region should be in the communication range of that node. In Fig. 3.4, the little disk located at (x, y) represents an arbitrarily selected node among the vulnerable nodes. Packets transmitted by the nodes in the shaded regions will collide at the selected node when these packets are scheduled for the same data slot. The expected number of nodes in these regions for the selected node at (x, y), $cand_1(x, y)$ and $cand_2(x, y)$, corresponding to the left and right hand side clusters are:

$$cand_1(x, y) = \frac{f(\sqrt{x^2 + y^2}, r_c, r_c)}{\text{Simulation Area}} N \tag{3.12a}$$

$$cand_2(x, y) = \frac{f(\sqrt{(d_{ch} - x)^2 + y^2}, r_c, r_{int})}{\text{Simulation Area}} N \tag{3.12b}$$

where r_c and r_{int} are the communication radius and the interference radius at the given transmission power level.

For a sufficiently large node density, the expected number of data collisions per frame due to those candidate nodes, $E_{coll}(x, y)$, can be approximated as:

$$E_{coll}(x,y) = \frac{cand_1(x,y)}{N_c} * \frac{cand_2(x,y)}{N_c} * min(E[Load], MaxTraffic), \quad (3.13)$$

where $Load$, $MaxTraffic$ and N_c are as defined in Section 3.2.1.

Averaging $E_{coll}(x, y)$ over the vulnerable region, V, and multiplying by the expected number of nodes in a cluster leads to the expected throughput loss on the transmissions from the nodes of one cluster due to one co-frame cluster, $n_{coll_{pCH}}$ as calculated in (3.14).

$$n_{coll_{pCH}} = \iint_V E_{coll}(x,y)dxdy \frac{N}{\text{Simulation Area}} \quad (3.14)$$

The collisions between each co-frame CH pair decreases the number of packet receptions of the packets transmitted by the members of the CH by $n_{coll_{pCH}}$. Multiplying $n_{coll_{pCH}}$ by the number of permutations of co-frame CH pairs leads to the expected value for the total number of collisions, n_{coll} as

$$n_{coll} = n_{coll_{pCH}} \binom{N_{CH_{total}}}{2} 2, \quad (3.15)$$

where $N_{CH_{total}}$ is the expected value of the total number of CHs in the network. Note that for sufficiently large node density, $N_{CH_{total}}$ is independent of node density and can be calculated similar to the calculation of p_d.

Hence, the number of collisions per node per superframe, $n_{coll_{pN}}$, is calculated as in (3.16).

$$n_{coll_{pN}} = \frac{n_{coll}}{N} \quad (3.16)$$

3.2.3 Throughput Model

The voice application under concern generates one packet per superframe when the nodes are in spurt state. Therefore, the expected number of generated packets per superframe is equal to the expected number of nodes in spurt state per superframe and can be calculated as

$$N_{P_{gen}} = p_s * N. \tag{3.17}$$

Each generated packet is dropped with a probability of P_{dp} and transmitted otherwise. The expected number of dropped packets per superframe can be calculated as

$$N_{P_{drop}} = N_{P_{gen}} * P_{dp}. \tag{3.18}$$

The number of packets transmitted per superframe per node, Tx_T, can easily be obtained by subtracting the number of dropped packets from the number of generated packets and dividing by the number of nodes as in (3.19a). Note that Tx_T can also be interpreted as the probability of a node to get access to the channel in an arbitrary superframe. This definition is indicated implicitly by the simplified form of the equation given in (3.19b) as the multiplication of the probability of not dropping a packet with the probability of being in spurt duration.

$$Tx_T = \frac{N_{P_{gen}} - N_{P_{drop}}}{N} \tag{3.19a}$$

$$= (1 - P_{dp}) * p_s \tag{3.19b}$$

If there were no collisions, each transmitted packet would have been received by all the neighboring nodes. Therefore, for an ideal case of no collisions, the number of packets received per node per superframe would be

$$Rx_{T_{ideal}} = Tx_T \, (p_c N) \tag{3.20}$$

However, collisions prevent some packets from being received successfully. The number of collisions per node per superframe, $n_{coll_{pN}}$, is calculated in Section 3.2.2.

Subtracting $n_{coll_{pN}}$ from $Rx_{T_{ideal}}$ yields the expected number of packet receptions per node per superframe, Rx_T, as in (3.21).

$$Rx_T = Rx_{T_{ideal}} - n_{coll_{pN}} \tag{3.21}$$

3.2.4 Energy Model

In addition to throughput, energy dissipation is an important parameter to evaluate for a given protocol. Thus, here we describe our energy model that is built off our throughput model described in the previous sections.

In our energy model, at any instant nodes should be in one of five different states, namely transmission state (S_{T_x}), successful reception state (S_{R_x}), carrier sensing state (S_{CS}), idle state (S_I), and sleep state (S_S). At each state, the power consumed by the nodes differ.

S_{T_x} is the state where nodes transmit a message. A node can be in S_{R_x}, S_{CS}, or S_I when it is listening to the medium. S_I is the state where nodes listen to the medium but cannot identify or carrier sense any message. In S_{CS}, nodes sense some ongoing transmission but cannot extract the message contents since the transmission is not powerful enough to overcome the noise. When the transmission is powerful enough for the node to identify its message contents, the node is assumed to be in S_{R_x}. Depending on the physical layer design of the radio, the power consumption of these three states may differ and are thus differentiated in our analytical model. S_S is the state where nodes, turn off their communication circuitry to save energy.

The aim of the analytical model is to calculate the average time spent in each state per node. Since the data sub-frame constitutes the largest part of the frame, it is crucial to accurately determine the expected value of the throughput. Using the expected value of the throughput, the average time spent in each state

Table 3.1: Abbreviations used in the energy model.

Abbreviation	**Explanation**
$Beacon_L$	Length of beacon slot in seconds
CA_L	Length of clusterhead announcement slot in seconds
$Contention_L$	Length of a single contention slot in seconds
$Header_L$	Length of header slot in seconds
IS_L	Length of information summarization in seconds
$Data_L$	Length of a single data slot in seconds
$SFduration$	Length of the superframe in seconds
N_{dataS}	Number of data slots per frame
N_{contS}	Number of contention slots per frame
N_{CH_c}	Expected number of CHs in the communication range of a randomly selected node
$N_{CH_{int}}$	Expected number of CHs in the interference range of a randomly selected node
$txPower$	Average power consumption in transmitting state
$rxPower$	Average power consumption in receiving state
$csPower$	Average power consumption in carrier sensing state
$idlePower$	Average power consumption in idling state
$sleepPower$	Average power consumption in sleeping state

and, in turn, the average energy consumption per successful transmission, can be calculated for any set of parameters.

Calculation of the time spent in each state depends strongly on the MAC protocol. In this section, state durations are calculated based on the MH-TRACE protocol and the corresponding frame structure given in Fig. 2.1. The abbreviations that will be used in this section are listed in Table 3.1.

As mentioned in Section 2.1.2, randomly chosen CHs regulate the channel

during the frame in which they are operating for the nodes in their communication range. In performing this role, CHs transmit and receive additional overhead packets. Thus, their energy consumption differs from that of ordinary nodes. In order to reflect these changes, the state durations of the CHs will be calculated separately.

State Durations for Ordinary Nodes

The first step in calculating the state durations is to determine when the nodes are transmitting. Considering the frame structure in Fig. 2.1, it can be observed that the ordinary nodes only transmit during the contention, IS and data slots. The contention slots are used for passing channel access requests to the CHs. The nodes request channel access only in the beginning of the entire spurt duration, since the CH reserves a data slot for the node unless the node goes out of the communication range or sends an end of stream message in the corresponding IS slot. Considering the fact that the contention slots are short and only used in rare cases, their affect on the transmission duration of the nodes is neglected. For each data slot, nodes transmit one IS slot to announce the data that will be sent. Given Tx_T, which was calculated in the previous section, $NodeTx_t$ can be calculated as in (3.22).

$$NodeTx_t = Tx_T * (Data_L + IS_L) \tag{3.22}$$

Nodes listen to the Beacon, CA and all IS slots in all the frames unconditionally. The data slot is only listened if the corresponding IS packet is successfully received. A node listens to the header slot only if it has data to send, in other words when the node is in spurt state. Considering these facts, $NodeListen_t$ can be calculated as in (3.23).

$$\begin{aligned} NodeListen_t =& N_f * (Beacon_L + CA_L + IS_L * N_{dataS}) \\ &+ Rx_T * Data_L + p_s * Header_L \end{aligned} \tag{3.23}$$

$NodeListen_t$ can be partitioned into three parts: $NodeIdleTime$ where nodes are on but cannot hear any transmission that is strong enough even for carrier sensing, $NodeCSTime$ where nodes can hear and carrier sense the transmission but cannot decode the message it contains, and $NodeRxTime$ where nodes actually receive a message.

Nodes can receive Beacon and CA packets from the CHs in their receiving range. Unless nodes go out of the transmission range just after the beacon message, nodes can receive the corresponding header message as well. Each node receives Rx_T data messages on the average. For each data message it receives, it should have received also one information summarization message in the corresponding data slot otherwise the node would be in S_S during the data slot. Summing these up, $NodeRx_t$ can be calculated as

$$\begin{aligned} NodeRx_t = {} & N_{CH_c} * (Beacon_L + CA_L * p_{CA}) \\ & + Rx_T * (Data_L + IS_L) + Tx_T * Header_L, \end{aligned} \tag{3.24}$$

where p_{CA} is the probability of a CH to transmit a CA message in the CA slot.

Nodes are assumed to be in S_{CS} state when they are listening the transmissions that are originated from a transmitter that is out of the receiving range but in carrier sensing range. Those packets cannot be received successfully. Failure to receive a successful IS slot makes the nodes to go to sleep in the corresponding data slot. Hence, in general, rather than carrier sensing the data slot, nodes go to sleep.

Since the nodes listen to all the beacon, CA, and IS slots, carrier sensing takes place for these packets if the transmitter is within interference range but not within communication range. Beacon and CA slots are transmitted by the CHs. Hence, their length should be multiplied by the number of CHs within interference range but not within communication range and operating in distinct frames and added to $NodeCS_t$. Following a similar idea, IS slot length should be multiplied

by the number of transmitting nodes within interference range but not within communication range to calculate the time spent for carrier sensing in IS slots.

Summing up, $NodeCS_t$ can be calculated as

$$NodeCS_t = \left(min(N_{CH_{int}}, N_f) - min\left(N_{CH_c}, N_f\right)\right)\left(Beacon_L + CA_L\right) + Tx_T\left(N_{int} - N_c\right) * IS_L \quad (3.25)$$

where $N_{CH_{int}}$ and N_{CH_c} are the expected number of CHs in the interference range and communication range, respectively. If any of those numbers are above N_f, N_f should be considered in $NodeCS_t$ calculation since introducing co-frame CHs do not increase the carrier sensing duration. N_{int} is the expected number of nodes in the interference range of a node and calculated in the same way as N_c is calculated in (3.6).

The remaining listen time is allocated as the idle time as in (3.26).

$$NodeIdle_t = NodeListen_t - NodeRx_t - NodeCS_t \quad (3.26)$$

When nodes do not transmit or listen to the channel they are assumed to turn off their radios, and go into the sleep state. Thus, $NodeSleep_t$ can be calculated as in (3.27).

$$NodeSleep_t = SFduration - NodeListen_t - NodeTx_t \quad (3.27)$$

State Durations for CHs

In addition to the data and IS slots, CHs also transmit Beacon, CA and Header slots of their frame. However, CA messages are transmitted with probability p_{CA}; otherwise CHs listen to these slots. Thus, $CHTx_t$ can be calculated as in (3.28).

$$CHTx_t = Beacon_L + CA_L * p_{CA} + Header_L + Tx_T * (Data_L + IS_L) \quad (3.28)$$

CHs are in the listening state during all the beacon slots other than the one in its own frame, CA slots other than the one it is transmitting, contention slots of its frame, all IS slots in all frames, and as many data slots as for an ordinary node. Considering these facts, $CHListen_t$ can be calculated as in (3.29).

$$\begin{aligned} CHListen_t = &(N_f - 1) * Beacon_L \\ &+ (N_f - p_{CA}) * CA_L \\ &+ N_{contS} * Contention_L \\ &+ N_f * (IS_L * N_{dataS}) + Rx_T * Data_L \end{aligned} \tag{3.29}$$

$CHRx_t$ is equal to the $NodeRxTime$ other than the header slot since CHs are in the transmission state during that slot. Neglecting the effects of slots, $CHRx_t$ can be calculated as in (3.30).

$$\begin{aligned} CHRx_t = &N_{CH_c} * (Beacon_L + CA_L * p_{CA}) \\ &+ Rx_T * (Data_L + IS_L) \end{aligned} \tag{3.30}$$

$CHCS_t$, $CHIdle_t$, and $CHSleepTime$ are calculated in the same way as their corresponding node equivalents as follows:

$$\begin{aligned} CHCS_t = &(min(N_{CH_{int}}, N_f) - min\,(N_{CH_c}, N_f)) \times (Beacon_L + CA_L) \\ &+ Tx_T \left(N \frac{\pi r_{int}^2 - \pi r_c^2}{simarea} \right) \times IS_L \end{aligned} \tag{3.31}$$

$$CHIdle_t = CHListen_t - CHRx_t - CHCS_t \tag{3.32}$$

$$CHSleep_t = SFduration - CHListen_t - CHTx_t \tag{3.33}$$

3.3 Validation of the Analytical Model

The analytical model described in the previous section introduces a simple tool that estimates the performance of MAC protocols employing a TDMA frame

structure and a soft clustering a soft clustering scheme. The model includes simplifying assumptions about node distributions and communications among the nodes. The model approximates the behavior of the protocol rather than reflecting the complex relationships, as can be detailed in simulations, thereby providing an estimate of the throughput and energy achieved by the protocol. In this section, we investigate the performance of the MH-TRACE protocol using our model and compare it with the results of simulations using ns-2 [95] in order to verify the model.

3.3.1 Simulation Environment

For comparison purposes, we conduct ns-2 simulations of MH-TRACE under different network scenarios and for various number of frames per superframe, N_f, values.

The source application is conversational voice, coded at 32Kbps, which translates into 100B data packets for every 25ms. Voice conversation is simulated with "spurts" and "gaps", whose lengths are exponentially distributed and statistically independent, with mean durations of 1.0s and 1.35s, respectively [96, 97].

The channel rate is set to 2Mbps. Given these values, in order to maintain data packet timing, the superframe duration is adjusted to be as close as possible to 25ms, which is the voice packet generation period. Table 3.2 shows details of the frame timing for each value of N_f.

Beacon, CA, contention, and IS packets are all 4 bytes long. The header packet has a variable length of $(4 + 2N_d)$ bytes depending on the number of data slots per frame. There is a 4 byte header and 100 byte data packet associated with each data slot. Details of the various header packet lengths can be found in [4].

The energy model is based on the one used in [98]. We used a constant transmission power that yields a communication radius of 250m under the two

Table 3.2: Superframe Parameters

Number of Frames N_f	Number of Data Slots,N_d	Number of Contention Slots,N_c	Superframe Time T_{sf}
4	12	15	24,976
5	10	6	25,060
6	8	9	24,984
7	7	6	25,172
8	6	6	24,992

way ground propagation model. Nodes are assumed to have the ability to identify an ongoing transmission and measure its power, as long as the power of the packet (at the receiver) is higher than the carrier sensing threshold. The transmission power that we use translates into a carrier sensing radius of 550m.

In the case of more than one packet arriving at a receiver simultaneously, none of the packets can be received by the receiver unless one of them captures the receiver. In order to capture the receiver, the power of the packet (at the receiver) must be at least 10 times stronger than all other transmissions that arrive at the receiver during the reception duration of the packet. Furthermore, in order to be received successfully, the packet must also be stronger than the minimum successful reception power threshold (i.e., the packet must be originated from a node within the communication radius of the receiver.).

The power spent by each node varies according to the operation performed by the node. During successful reception, collision and carrier sensing periods, the node consumes power at the rate of the reception power level. There is also an idle state where only the power needed to run the circuitry is dissipated without any actual packet receptions. The nodes are assumed to turn off any circuitry

when they go into the sleep state, where the power consumption is minimal.

The random way-point mobility model [99] [100] is used in the mobile scenarios, with node speeds chosen from a uniform random distribution between 0.0 m/s and 5.0 m/s with zero pause time. All simulations are run in an area of 1km x 1km for 100 seconds. The simulations are repeated with the same parameters 10 times. All results are based on sample averages and sample standard deviations of those 10 iterations.

3.3.2 Validation

The model approximates various statistics about the protocol such as P_{dp}, Tx_T, n_{coll}, Rx_T, and E_c. In this part, the estimates of these statistics are compared to their counterparts obtained through simulation averages.

Any of those statistics is expected to be a stochastic number with a possibly complex distribution, since the process leading to the statistics is a combination of various random events. The aim of the analytical model presented in this chapter is to estimate the population mean for each statistic. Simulations provide samples from the population. From the central limit theorem, as the size of the simulation set increases, the statistical distribution of simulation averages approaches to a Gaussian distribution. Therefore it is almost impossible to obtain an exact match between the simulation averages and the population means (i.e., the probability of obtaining a simulation set whose average is equal to the population mean is zero.). Thus, any comparison between the simulations and estimated values should be done with respect to the standard deviations. As the parameters of the protocol are varied, the means of the statistics calculated with the model are expected to behave similar to the simulation averages, although the absolute values may not match exactly.

The mobility is handled by the CH creation and maintenance mechanisms in-

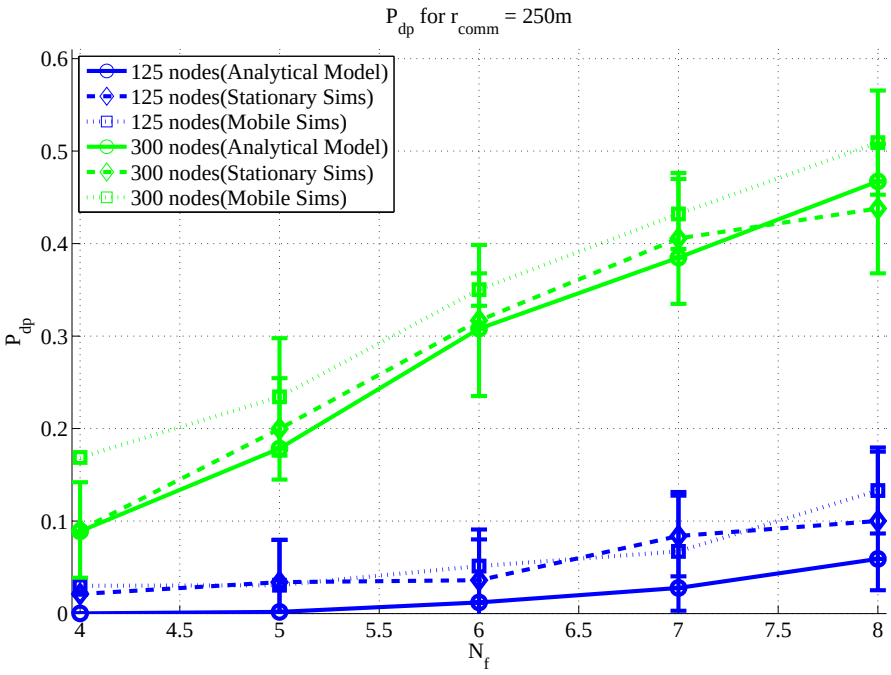


Figure 3.5: P_{dp} vs N_f for a communication radius of $250m$ and a network size of 1km$\times$1km.

herent to the protocol itself. Assuming these mechanisms are working fast enough, we expect the results of the mobile simulations to be statistically similar to those of stationary simulations. Thus, the analytical model presented in the previous section can also be used for mobile scenarios even though the model does not include any mechanisms to account for node mobility. The analytical results are compared to the simulation results for both stationary and mobile scenarios.

Throughout this section, the estimates on P_{dp}, Tx_T, n_{coll}, Rx_T, and E_c are compared to the simulation averages for both stationary and mobile cases for two different node densities (125 and 250 $nodes/km^2$) and five different N_f values (ranging from 4 to 8).

Probability of Dropping a Packet (P_{dp})

One of the main factors affecting the throughput is the dropped packets, as stated earlier. The dropped packets are the result of channel blockages and are important to consider while designing the protocol and setting its parameters. Fig. 3.5 compares the estimates on P_{dp} with the simulation averages and sample standard deviations for both stationary and mobile scenarios.

As the number of nodes in the network increases, the average node density and hence *Load* increases. An increased *Load* leads to a higher P_{dp}. This can be observed by the increase in P_{dp} as the number of nodes is increased from 125 to 250.

As N_f is increased, the superframe is divided into smaller partitions leading to shorter frames. Shorter frames can accommodate fewer data slots. Hence $MaxTraffic$ decreases for increased N_f. Under the same load, decreased $MaxTraffic$ leads to an increase in P_{dp}. This can be observed by the increase in P_{dp} as N_f is increased.

In the mobile case, CHs coming near to each other resign. Until, the new CHs are chosen, the nodes near the resigned CH would have fewer alternatives to reach the medium. Thus, P_{dp} in the mobile case is higher than the P_{dp} in the stationary case. Since the model does not take into account CH resignations, the estimates are lower than the mobile case results. Although the results of the mobile case are not as close to the model result as the stationary case, considering large standard deviations, the model can provide an estimate for P_{dp} in the mobile case as well.

Number of Data Transmissions per Node per Superframe (Tx_T)

The analytical model combines the estimates on the number of generated packets per superframe and P_{dp} to yield Tx_T. Fig. 3.6 presents simulation averages and estimates on Tx_T.

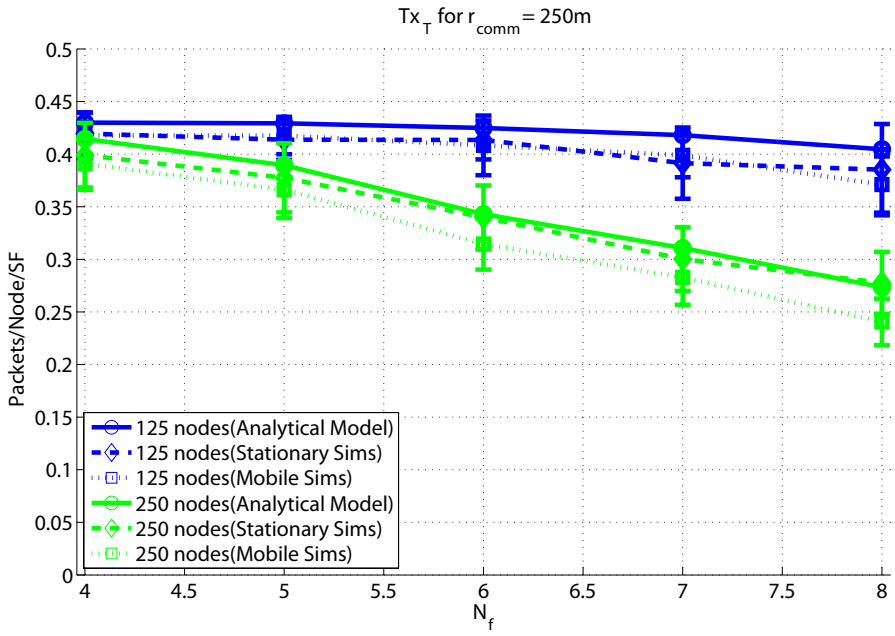


Figure 3.6: Tx_T vs. N_f for a communication radius of 250m and a network size of 1km×1km.

The number of nodes in the network inherently determines the number of generated packets, since each node also acts as a data source. As the number of nodes increases, the demand for channel access also increases. As a result, each node has to get a smaller share on the average from the channel. MH-TRACE maintains ongoing voice streams and denies any new access to the channel until the channel becomes available again. Thus, some nodes cannot be given channel access in a timely fashion and have to drop their packets. Thus, Tx_T decreases as the number of nodes in the network increases, as can be observed in Fig. 3.6.

Tx_T also decreases as N_f is increased. A larger N_f translates into shorter frames. Shorter frames can accommodate fewer data slots. Hence $MaxTraffic$ decreases for increased N_f. Under the same load, decreased $MaxTraffic$ leads to an increase in P_{dp} and hence a decrease in Tx_T. This can be observed by the decrease in Tx_T as N_f is increased.

For a 125 node network and an N_f value of 6, Tx_T is almost equal to the expected value of p_s (0.43), which indicates the nodes' ability to get channel access whenever they need. Thus, one cannot gain much from decreasing N_f from 6 to 4 for a 125 node network. On the other hand, for a 250 node network, the demand for channel access is much higher. Thus, for a 250 node network, decreasing N_f from 6 to 4 increases a node's chance to get channel access in a timely fashion and increases Tx_T from 0.34 to 0.4 packets/node/superframe.

Number of Collisions per Node per Superframe ($n_{coll_{pN}}$)

Another important factor in estimating the throughput is the collisions. In order to calculate the effect of collisions on the throughput metrics, we have to consider the collisions that would result in a packet reception if there were no interfering transmission. In other words, in the case of a collision, only those packets that originate from inside the communication radius should be taken into account. Other colliding packets are not considered as lost since they could not be received even if there were no collisions. The number of collision figures throughout this section include only such packets.

Fig. 3.7 presents the simulation averages and analytical model estimates for the number of collisions per node per superframe. As the number of nodes in the network increases, the expected value of *Load* increases. As long as the expected value of *Load* is below $MaxTraffic$, an increase in *Load* leads to a larger E_{coll} and hence a larger $n_{coll_{pN}}$. This can be observed by the increase in $n_{coll_{pN}}$ as the number of nodes is increased from 125 to 250.

CHs choose the frame with the least interference to operate in among the frames in a superframe. As N_f is decreased, the CHs have fewer alternatives to choose from. As a result, the expected value of the separation between co-frame clusters decreases. The nodes that reside in between two co-frame clusters tend

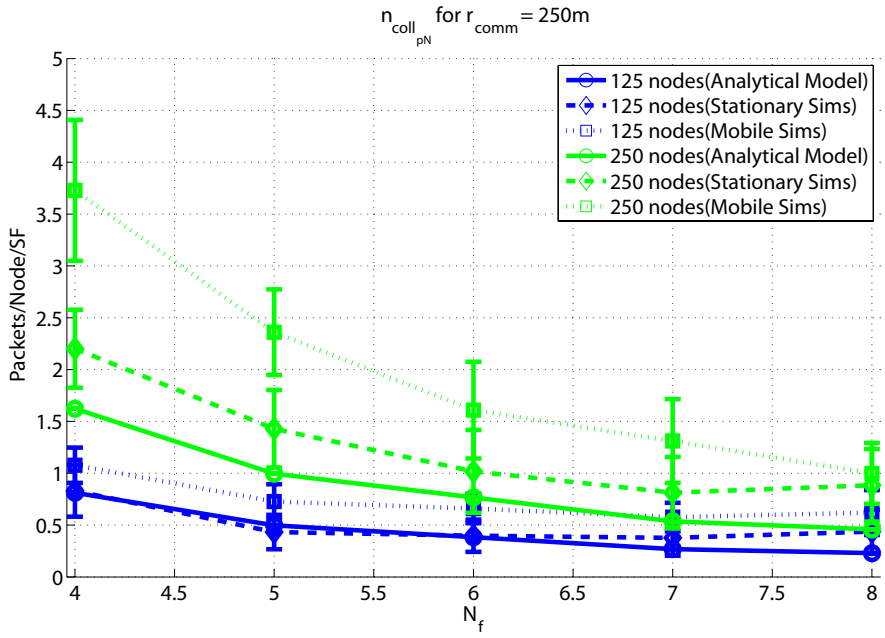


Figure 3.7: $n_{coll_{pN}}$ vs. N_f for a communication radius of 250m and a network size of 1km×1km.

to have more interference from one of the co-frame clusters, and hence $n_{coll_{pN}}$ increases. This can be observed by the increase in $n_{coll_{pN}}$ as N_f is decreased in Fig. 3.7.

Number of Data Receptions per Node per Superframe Rx_T

The last step in calculating the number of receptions is to incorporate the effect of collisions. The analytical model combines the estimates of $n_{coll_{pN}}$ and Tx_T to yield Rx_T. Fig. 3.8 presents the estimates on Rx_T together with the simulation averages and sample standard deviations for the number of successful data receptions per node per superframe.

Increasing the number of nodes in the network has a two-fold effect on the total number of receptions since each node acts as both a data source and a data sink. Thus, for the same Tx_T level, as the number of nodes doubles, Rx_T is also

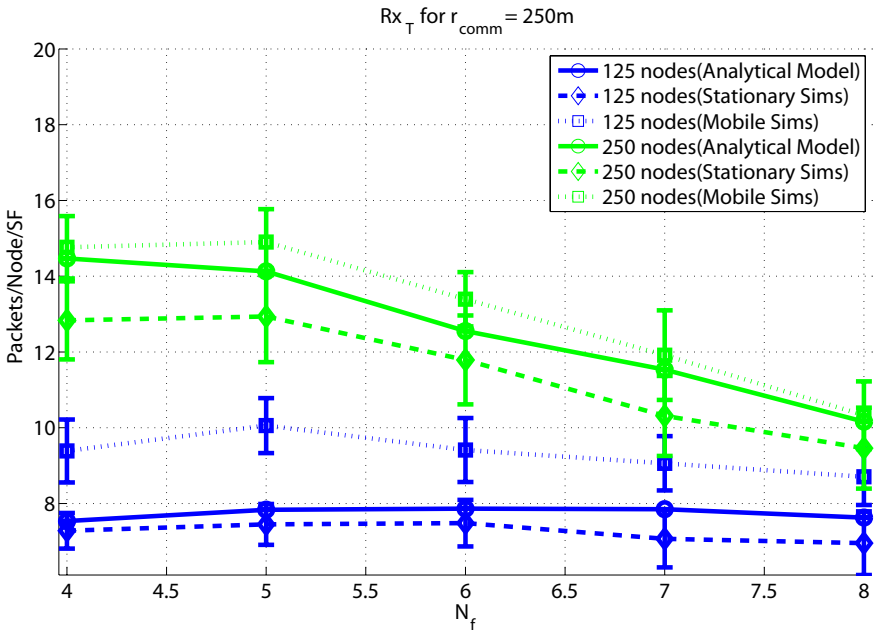


Figure 3.8: Rx_T vs. N_f for a communication radius of 250m and a network size of 1km×1km.

expected to increase by a factor of two. This can be observed at the N_f value of 4, where Tx_T values are almost identical, while Rx_T for the 250 node network is almost twice as large as Rx_T for the 125 node network.

Energy Consumption per Node per Superframe E_c

The analytical model calculates the energy consumption based on the throughput figures. The average energy consumption per node per superframe, E_c, obtained through the analytical model is compared to the simulation results in Fig. 3.9.

The average energy consumption is strongly related to Rx_T. This similarity can be observed by comparing Fig. 3.9 to Fig. 3.8. Increases in Rx_T directly translate into shorter $NodeIdleTime$ and longer $NodeRxTime$, which in turn increases energy consumption.

However, the energy spent for CH creation and maintenance is independent

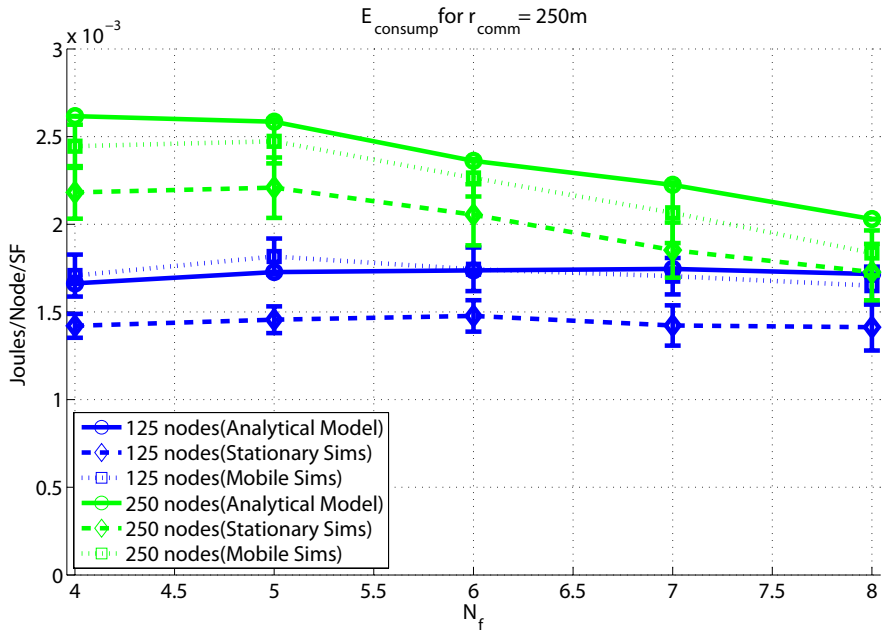


Figure 3.9: E_c vs. N_f for a communication radius of 250m and a network size of 1km×1km.

of the traffic load of a CH. Hence, unlike Rx_T, for the same Tx_T (N_f value of 4 yields approximately the same Tx_T for the 125 node and the 250 node networks), doubling the node density does not double the E_c.

Throughout this section we have observed that in all of the performance metrics, the standard deviations in the results are quite large. This is due the strong dependency of the model on the selection of CHs, which is done through a randomized scheme. It is difficult to get an exact match when modeling random variables with such high variances. The mathematical model presented in this chapter has been observed to estimate the number of received packets and the average energy, while following the same trends as the simulation results. In the following section, we investigate the suitability of the model for parameter optimization.

3.4 Optimization of Spatial Reuse

The degree of spatial reuse directly affects the performance of the protocol, both in terms of throughput and energy consumption. Reusing the channel for clusters separated enough increases the amount of traffic that can be relayed in a superframe and decreases the number of dropped packets. However, it also increases the interference and in turn the number of collisions. Thus, there is a trade-off between dropped packets and collisions, which must be arbitrated to make best use of the channel resources.

In TDMA based protocols such as MH-TRACE, the degree of spatial reuse is determined by the number of frames per superframe, N_f. Decreasing that parameter decreases the number of alternative frames that each CH can choose from and increases the interference in each frame. However, since there are fewer frames in the superframe, each frame becomes longer and can maintain more data slots, which in turn increases the $MaxTraffic$ per cluster.

The analytical model introduced in this chapter can be utilized to optimize N_f for an arbitrary node density. This parameter can be optimized for any statistic. In this section, we will determine the N_f value that maximizes throughput and the N_f value that maximizes energy efficiency. After presenting the results for a number of node densities, the method will be validated by comparing the results of simulations using the optimized N_f parameter to simulations using other N_f values.

3.4.1 Optimization Seeking Maximum Throughput

As $MaxTraffic$ per cluster increases, P_{dp} and in turn the number of dropped packets in the network decreases. A reduced number of dropped packets leads to a higher Tx_T as can be observed from (3.19). Therefore, setting N_f to a

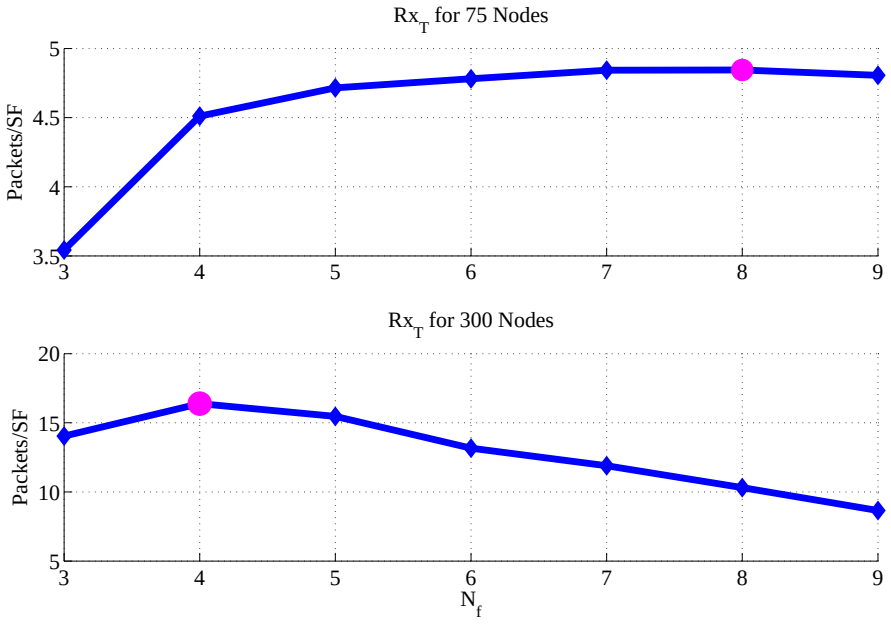


Figure 3.10: Rx_T vs. N_f for 75 and 300 node networks. The N_f value leading to the maximum Rx_T is marked with a dot.

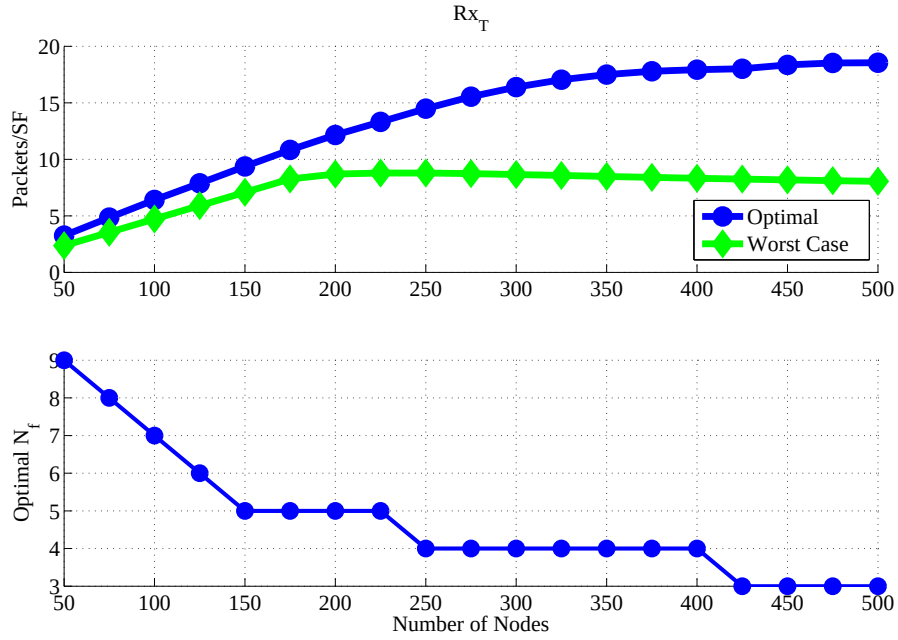


Figure 3.11: Maximum Rx_T for $N_f \epsilon \{3, 4, 5, 6, 7, 8, 9\}$ as the number of nodes in the network varies, together with the N_f leading to maximum Rx_T.

lower value, leads to higher Tx_T and tends to increase Rx_T. On the other hand, decreasing N_f also decreases the co-frame CH separation as can be observed from (3.9). As the separation between co-frame CHs is decreased, n_{coll}, increases and tends to decrease Rx_T.

Theoretically, the maximum throughput should be realizable at the N_f value that minimizes the combined effects of collisions and dropped packets at the given node density. The analytical model presented in this chapter can be used to resolve the trade-off between collisions and dropped packets. For a given network size and node density, Rx_T can be optimized over any finite set of realizable N_f values by evaluating the model for each value of N_f in the set and choosing the one that yields the highest Rx_T.

The variation in Rx_T as N_f is varied for 75 node and 300 node networks is presented in Fig. 3.10. The figure clearly demonstrates that N_f values of 8 and 4 maximize Rx_T for 75 node and 300 node networks, respectively. When N_f is increased beyond the optimal point, the loss in throughput due to dropped packets overwhelms savings due to the reduction in the number of collisions and hence Rx_T decreases. Conversely, as N_f is decreased below the optimal point, the effect of collisions dominate the effect of dropped packets and Rx_T again decreases.

The argument for 75 node and 300 node networks can be generalized for any node density. Fig. 3.11 presents the maximum and minimum Rx_T values together with the optimizing (maximizing) N_f values over the set $N_f \epsilon \{3, 4, 5, 6, 7, 8, 9\}$ as the number of nodes in the network is varied from 50 to 500.

As the number of nodes in the network is increased, the optimal N_f should be decreased in order to compensate for increased *Load* in the cluster. This behavior can be seen directly from Fig. 3.11.

The decision on which N_f value to use is relatively less important for sparse networks, as can be observed in Fig. 3.11 by the decreasing performance difference between the minimum and maximum Rx_T values as node density is decreased. On

the other hand, the decision about which N_f value to use is quite important for dense networks. For 500 node networks, choosing an incorrect N_f from the set ϵ {3, 4, 5, 6, 7, 8, 9} may lead to 56% loss in Rx_T.

3.4.2 Optimization Seeking Maximum Energy Efficiency

Another important metric for a MAC protocol is the number of received packets per unit energy consumption. A larger number of received packets per energy consumption indicates a more energy efficient protocol. Therefore, the energy efficiency can be defined as the number of packets received per unit energy consumption.

N_f determines the energy efficiency as it affects both the number of received packets and the energy consumption. The variation in energy efficiency for 75 node and 300 node networks as N_f is varied is presented in Fig. 3.12.

Fig. 3.13 presents the maximum and minimum energy efficiency values together with the optimizing (maximizing) N_f values over the set N_f ϵ {3, 4, 5, 6, 7, 8, 9} as the number of nodes in the network is varied from 50 to 500.

The deviation between maximum and minimum energy efficiency values are around 16% of the maximum energy efficiency values for all node densities. For 500 node networks, choosing an incorrect N_f from the set ϵ {3, 4, 5, 6, 7, 8, 9} may lead to 17% fewer packet receptions per unit energy consumption.

Comparing Fig. 3.11 and Fig. 3.13, although for some node densities the optimal N_f values for maximum energy efficiency and for maximum Rx_T are identical, this cannot be generalized for all node densities. For instance, when the number of nodes in the network is reduced below 125 nodes, optimization for maximum Rx_T suggests increasing N_f above 6 while optimization for maximum energy efficiency suggests keeping it at the value of 6. For such node densities, the $MaxTraffic$

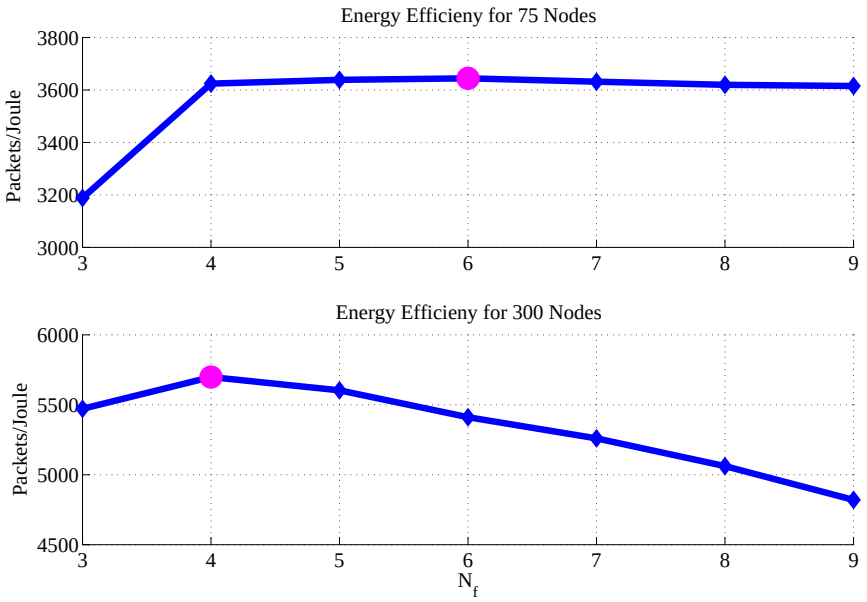


Figure 3.12: Energy Efficiency vs. N_f for 75 and 300 node networks. The N_f value leading to the maximum energy efficiency is marked with a dot.

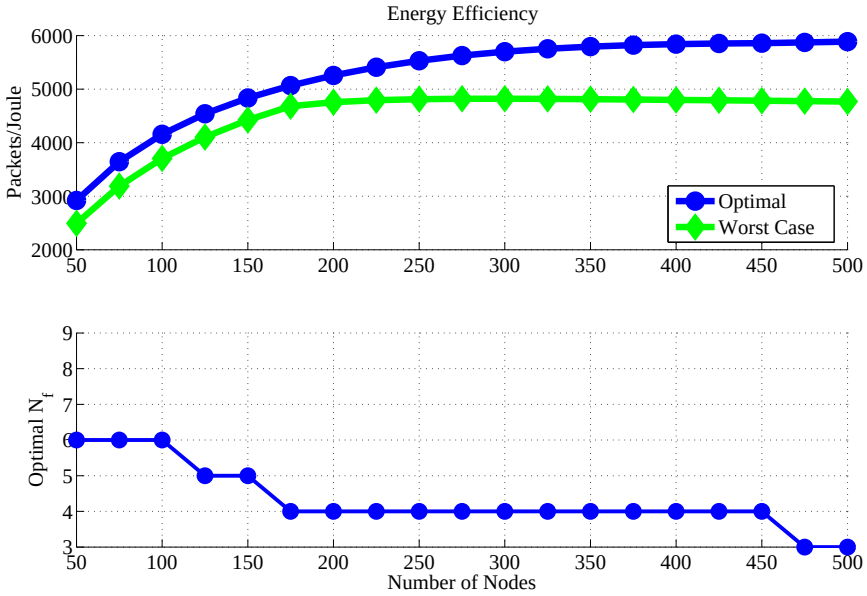


Figure 3.13: Maximum energy efficiency for $N_f \epsilon \{3, 4, 5, 6, 7, 8, 9\}$ as the number of nodes in the network varies, together with the N_f leading to maximum energy efficiency.

in each cluster is sufficient to meet the demand even for higher N_f values (note that P_{dp} is close to 0 even for 125 nodes in Fig 3.5). Thus, increasing N_f takes advantage of reducing $n_{coll_{pN}}$. Reducing the number of collisions and increasing Rx_T are also desirable in optimizing the energy efficiency. However, increasing N_f also comes with the cost of increased overhead messages per superframe. There would be more beacon, CA, contention and header slots in a superframe. Both listening and transmitting the additional overhead increases energy consumption and reduces energy efficiency. The increase in energy consumption overwhelms incremental increases in throughput.

3.4.3 Validation of Optimization Methods Through Analytical Model

In this subsection, the validity of using the analytical model to optimize the N_f parameter is tested using the available simulation results as an evaluation basis. In Figs. 3.11 and 3.13, the optimal values of N_f predicted by the model were presented for various node densities for throughput and energy efficiency. In this section, the simulation averages using these N_f values are compared to the averages of simulation results that use other N_f values.

Validation of Optimization Seeking Maximum Throughput

For throughput optimization, the performance of each N_f value can be defined as the average simulated throughput for that N_f value as a percentage of the maximum simulated throughput among all N_f values for a given node density. Therefore, the N_f value leading to the highest average throughput has 100% performance. For a range of node densities, Fig. 3.14 depicts the simulated performance of the optimal N_f values predicted by the model together with the

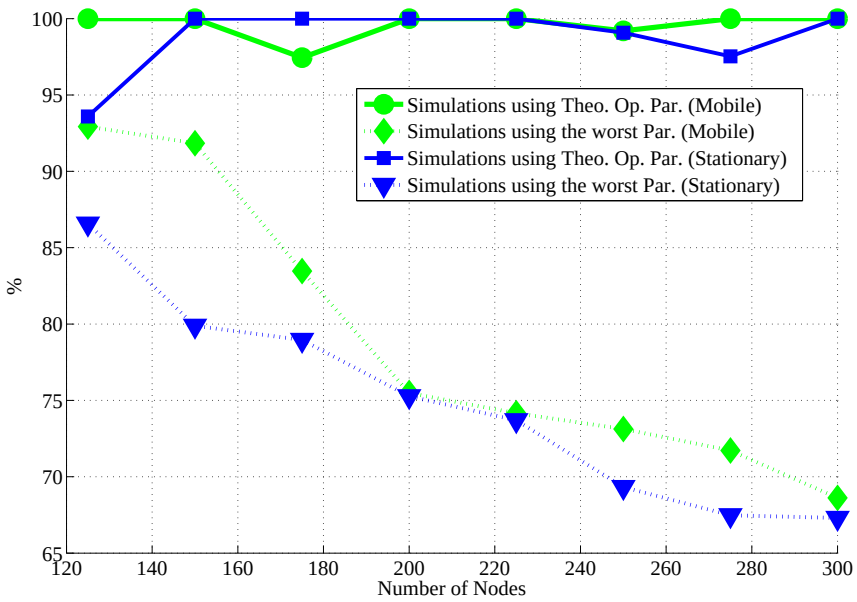


Figure 3.14: Simulated average Rx_T of the N_f values decided through the analytical model together with the worst average Rx_T for various node densities. All Rx_T values are indicated as a percentage of the maximum average simulated Rx_T at a given node density.

simulated performance of the N_f value leading to the lowest average simulated throughput among N_f values ranging from 4 to 8.

All decisions obtained through analysis lie within 6% and 3% of the maximum realizable throughput for the stationary and mobile cases, respectively. Analysis is successful in determining the optimal N_f value for dense networks, in both the mobile and stationary cases. However, in the stationary case, the model estimation converges to the optimal decision faster as the node density is increased.

For low node densities, the demand for channel access is fairly low and the $MaxTraffic$ is sufficient. The dropped packets are merely the results of non uniformities in node distribution. Thus, increasing $MaxTraffic$ per cluster (i.e., decreasing N_f) has relatively weaker impact on throughput. Also the impact of

collisions on throughput for sparse networks is not as strong as it is for dense networks since the number of collisions is proportional to the square of the node density. Hence, the decision on which N_f value to use is relatively less important for sparse networks, as can be observed in Fig. 3.14 by the decreasing performance difference between the worst and best decisions as node density is decreased. On the other hand, the decision about which N_f value to use is quite important for dense networks. For 300 node stationary and mobile networks, choosing the worst N_f value achieves only 65% and 64% of the optimum throughput, respectively. Thus, appropriate setting of the N_f parameter is extremely important, and the analytical model provided in this chapter can be used to optimally set this value.

Validation of Optimization Seeking Maximum Energy Efficiency

For energy efficiency optimization, the performance of each N_f value can be expressed as the average simulated energy efficiency for that N_f value as a percentage of the maximum simulated energy efficiency among all N_f values for a given node density. The performance of the analytically selected optimal N_f value is compared to the worst performance among N_f values ranging from 4 to 8 for each node density. Fig. 3.15 depicts this comparison together with optimal N_f values obtained through analysis for both mobile and stationary cases.

For the stationary case, the optimal N_f value estimated by the model is also the optimal among the simulation results that leads to the highest energy efficiency among $N_f \epsilon \{4, 5, 6, 7, 8\}$ and for all node densities, validating the use of the model in the optimization procedure of N_f for maximum energy efficiency.

For the mobile case, the N_f value suggested by the analytical model leads to the highest average energy efficiency of the simulation set for all node densities with the exception of the network with 125 nodes. Even for the relatively sparse network of 125 nodes, the energy efficiency of the simulation results using the N_f

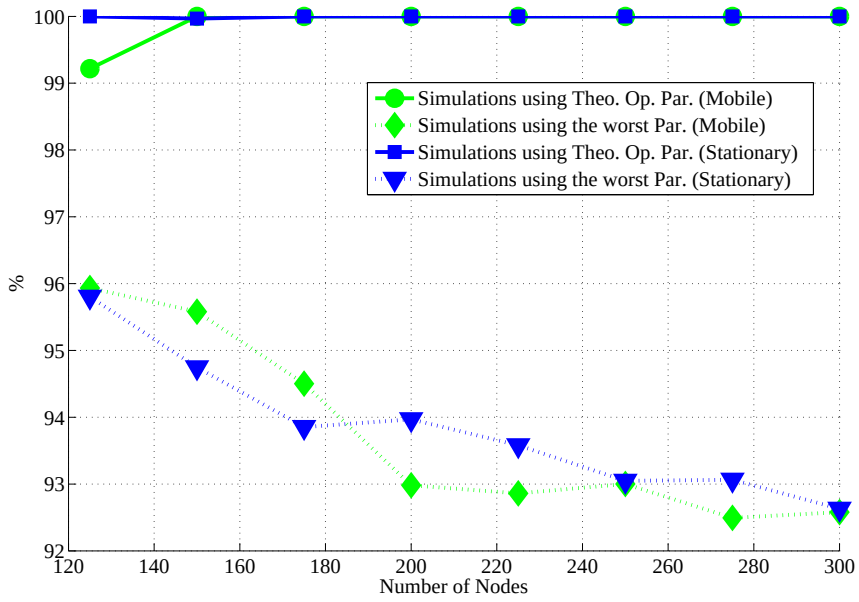


Figure 3.15: Simulated average energy efficiency of the N_f values decided through the analytical model together with the worst average energy efficiency for various node densities. All energy efficiency values are indicated as a percentage of the maximum average simulated energy efficiency at a given node density.

value suggested by the analytical model deviates less than 1% of the maximum average energy efficiency obtained through simulations using all possible Nf values.

Due to the fixed energy consumption for cluster maintenance, as the number of packets received per unit time increases, the energy efficiency tends to go up. Therefore dropped packets and collisions both decrease the energy efficiency. Collisions further decrease energy efficiency, as the transmission energy for collided packets are wasted. Since the impact of collisions and dropped packets are stronger for denser networks, the decision on which N_f value to use is relatively more important for denser networks, as can be observed in Fig. 3.15 by the increasing performance difference between the worst and best decisions as node density is increased. For 300 node stationary and mobile networks, choosing the worst N_f value leads to 7.5% loss in energy efficiency. Thus, appropriate setting of the N_f parameter ensures energy efficiency, and the analytical model provided in this chapter can be used to optimally set this value.

3.5 Varying the Data Generation Rate

In the previous section, the variation in the performance of the protocol is observed as N_f, which is an internal parameter of the protocol, is varied. The model can also be used to estimate the performance of the protocol as parameters external to the protocol, such as data generation rate, are varied.

Data generation rate will be increased by changing the mean spurt and gap durations while keeping the voice coding rate and the length of each data packet constant. The mean spurt and gap durations are used in calculating the p_s value as described in section 3.2.1. This section investigates the parameters such as P_{dp}, $n_{coll_{pN}}$, Tx_T, Rx_T and E_c as p_s is varied from 0 to 1. In the $p_s = 0$ case, nodes do not generate any data packets, whereas in the $p_s = 1$ case, the input application generates data packets for each node all the time.

Fig. 3.16 presents P_{dp} for various p_s values. As p_s is increased, *Load* and P_{dp} increase. Increasing p_s and increasing the number of nodes in the network has the same effect on P_{dp}. Therefore, increasing p_s just compresses the curve along the x-axis in Fig. 3.16. For instance, when p_s is 0.2, P_{dp} reaches 0.2 for an 800 node network, while it reaches to the same value for a 400 node network when p_s is 0.4.

Fig. 3.17 presents $n_{coll_{pN}}$ for various p_s values. As p_s is increased, the expected value of *Load* increases, which in turn increases $E_{coll}(x, y)$ for some (x, y) pair, as can be observed from (3.13).

For low node densities, $E[Load]$ is the determining factor in the last term in (3.13), $min(E[Load], MaxTraffic)$. As p_s is increased, $E[Load]$ increases, which in turn increase $n_{coll_{pN}}$. On the other hand, for large node densities, the bounding factor is the $MaxTraffic$, which is independent of both node density and p_s. Therefore, increasing p_s has negligible effect on $n_{coll_{pN}}$ when the number of nodes in the network is high.

Fig. 3.18 presents Tx_T for various p_s values. Examining (3.19), it can be observed that p_s effects Tx_T both through P_{dp} and through a multiplicative factor. For low node densities, P_{dp} is negligible. For a 50 node network, Tx_T values are almost identical to p_s in Fig. 3.18. On the other hand, for high node densities the effect of P_{dp} dominates in Tx_T as can be observed by the converging curves.

Fig. 3.19 presents Rx_T for various p_s values. For low node densities, the addition of new load through increasing either the number of nodes or p_s increases Rx_T as there is unutilized data slots. However, as load is increased, Rx_T converges to a fixed value where all the available data slots in the network is exhausted.

As can be observed from (3.17), the average total load injected to the network can be increased either by increasing the number of nodes in the network (since each node is also a data source) or increasing the amount of data generated by each node. However, increasing the number of nodes in the network also increases

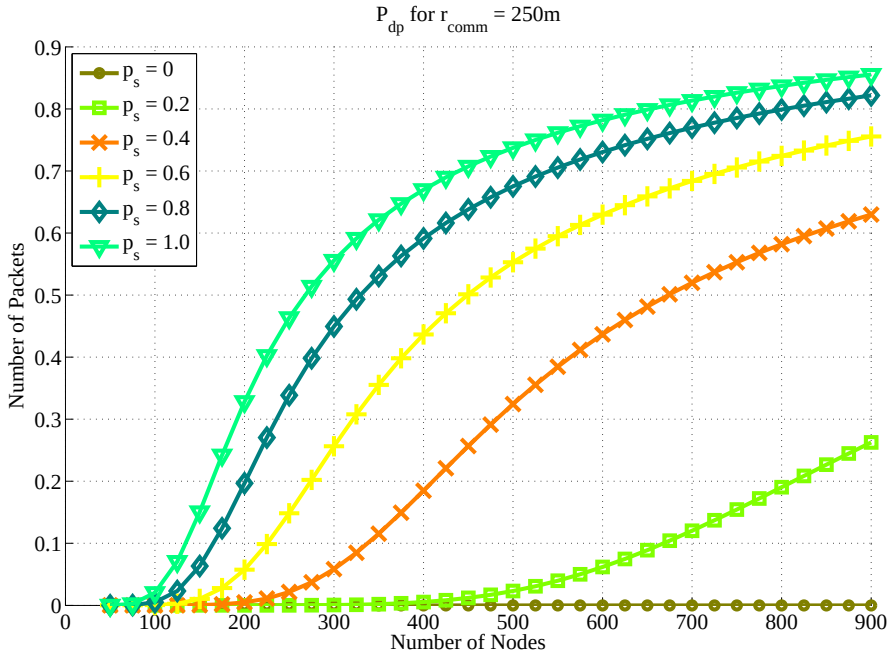


Figure 3.16: P_{dp} vs. number of nodes in the network for various p_s values with N_f set to 4.

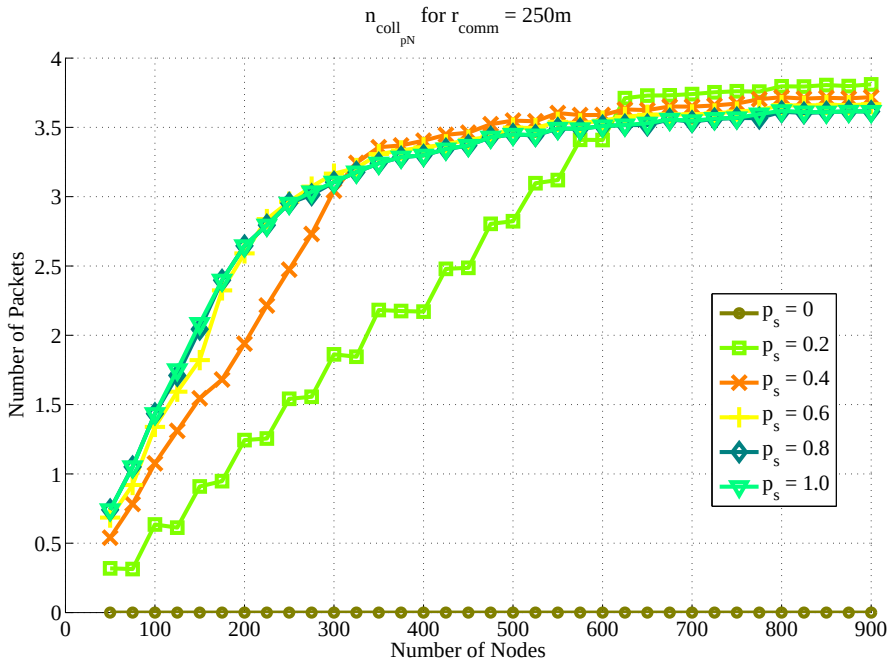


Figure 3.17: $n_{coll_{pN}}$ vs. number of nodes in the network for various p_s values with N_f set to 4.

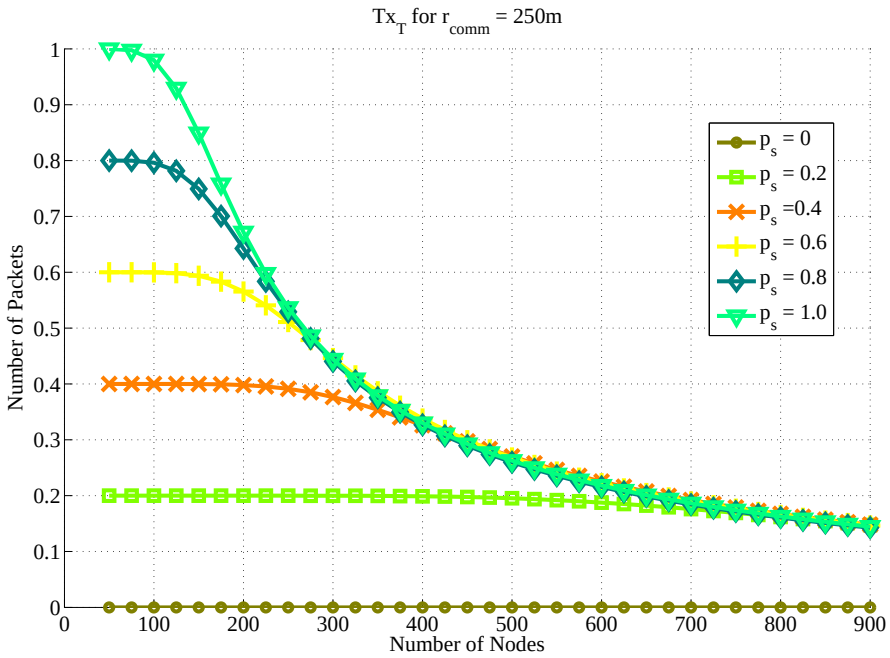


Figure 3.18: Tx_T vs. number of nodes in the network for various p_s values with N_f set to 4.

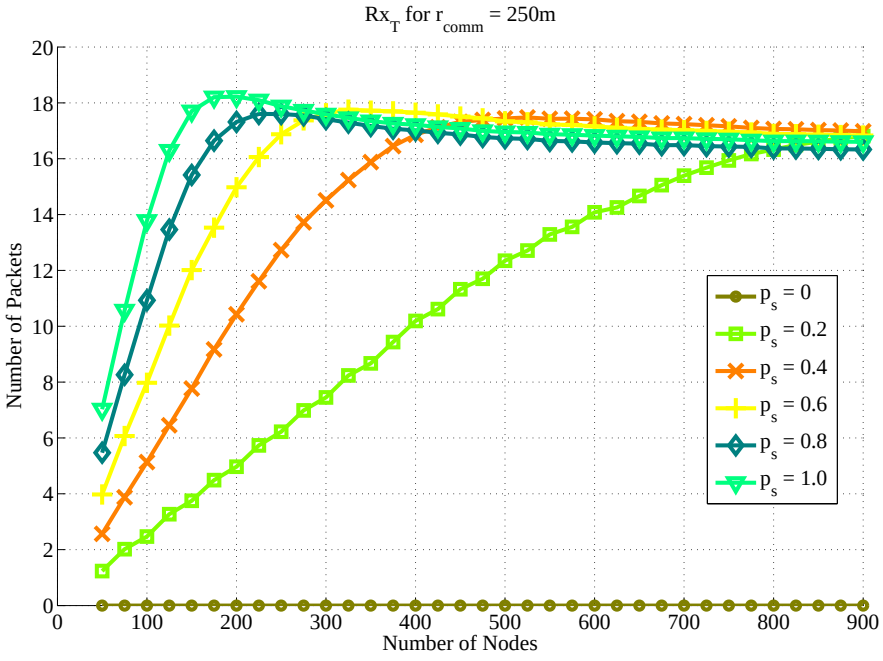


Figure 3.19: Rx_T vs. number of nodes in the network for various p_s values with N_f set to 4.

Table 3.3: Rx_T for a network with a total $N_{P_{gen}}$ of 100 Packets/SF.

Number of Nodes	p_s	$N_{P_{gen}}$	Rx_T
100	1.0	100	13.9
125	0.8	100	13.6
167	0.6	100	13.2
250	0.4	100	13.0
500	0.2	100	12.4

the variation in *Load* distribution, which in turn decreases the performance. Table 3.3 shows the Rx_T values corresponding to a network with total $N_{P_{gen}}$ of 100 packets/SF. It can be observed that although $N_{P_{gen}}$ is the same for all 5 cases, the network with the least number of nodes (or highest p_s) performs the best.

Fig. 3.20 presents E_c for various p_s values. When the network is idle (no load on the network), the energy is spent only to maintain the CH structure and is independent of node density. As load is introduced to the network, additional energy is spent for data transmissions and receptions. The increase in E_c is similar to the increase in Rx_T as p_s is increased since energy consumption is mostly due to reception.

3.6 Summary

In this chapter, we introduced an analytical model that estimates the performance of MAC protocols employing a soft-clustering structure and a TDMA channel access scheme, specifically the MH-TRACE protocol. The model eliminates the need for cumbersome and lengthy simulations to evaluate the performance of the

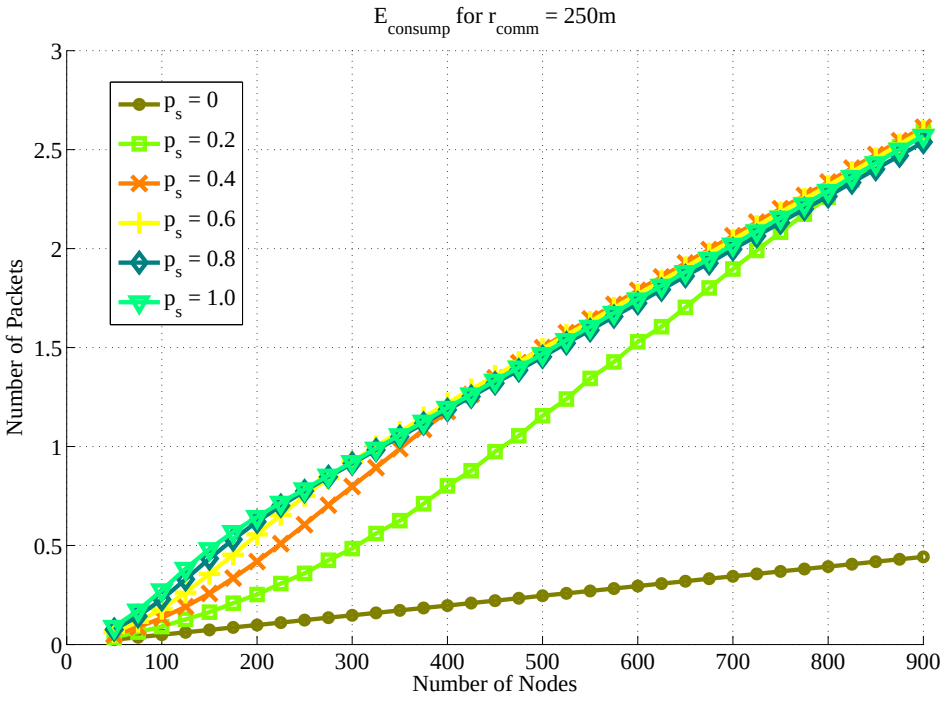


Figure 3.20: E_c vs. number of nodes in the network for various p_s values with N_f set to 4.

protocol. The analytical model presented in this chapter is simple enough to be run in seconds for a large number of parameter sets yet powerful enough to be used in optimization studies. By using simple probability distributions, the model accurately estimates the loss in throughput due to dropped packets and collisions, and thus can determine the expected throughput, the time spent at each energy state by the CHs and ordinary nodes, and the average energy consumption of each node in the network.

The analysis made in this chapter also reveals the importance of proper adjustment of the parameters according to the network conditions. It has been shown that improper selection of the parameters may result in serious throughput losses (up to 33% for stationary and 32% for mobile scenarios) and increased energy consumption per data reception (up to 7.5% for both stationary and mobile scenarios) in dense networks. The primary determining factor in the decision is the

node density.

The model presented in this chapter not only makes the analysis possible for very dense networks, where the simulation of the network is impractical, but the model also reveals the impact of the parameters on system performance. It should be emphasized that although the analytical model developed in this chapter was tested on MH-TRACE, the throughput model can be adapted to any MAC protocol employing a soft-clustering structure and a TDMA frame structure. Furthermore, the model can be modified for hard clustering protocols as well. The soft clustering assumption is used in calculating the number of collisions as well as in the throughput and energy models. To apply this model to a hard clustering protocol, these sections need to be modified.

The model helps to find the protocol parameters that make best use of resources in the long run using general probability distributions for node mobility and packet generation. It can neither calculate the protocol performance nor optimize the parameters for transient conditions. In the next chapter, we show that the resource utilization can be enhanced significantly by constructing dynamic mechanisms in the model that can accommodate for transient behaviors.

4 Cooperative Load Balancing and Dynamic Channel Allocation for Mobile Ad Hoc Networks

4.1 Introduction

The key challenges in effective MAC protocol design are the maximization of spatial reuse and providing support for non-uniform load distributions. Spatial reuse is tightly linked to the bandwidth efficiency. Due to the lossy nature of the propagation medium, the same channel resources can be used in spatially remote locations simultaneously without affecting each other. Incorporating spatial reuse into the MAC protocol drastically increases bandwidth efficiency. On the other hand, due to the dynamic behavior in MANETs, the traffic load may be highly non-uniform over the network area. Thus, it is crucial that the MAC protocol be able to efficiently handle spatially non-uniform traffic loads. Uncoordinated protocols intrinsically incorporate spatial reuse and adapt to the changes in load distribution through the carrier sensing mechanism. However, coordinated protocols require careful design at the MAC layer allowing the channel controllers to

utilize spatial reuse and accommodate any changes in the traffic distribution.

In Chapter 3, we presented a mathematical model that can be used to optimize the protocol parameters for any given condition. It is shown that properly selected parameters decreases the energy consumption and increases the throughput significantly.

However, this optimization can be used only where a priori models about the behavior of the network exist. Oftentimes node distribution patterns and packet generation patterns are not known in the design phase and also can change significantly after deployment and throughout the operation of the network. In such cases, having a model that can optimize the parameters for any given conditions does not suffice. The network has to be designed flexibly to accommodate changes over time.

Changes in the node distribution and packet generation patterns result in a non-uniform load distribution. Similar to cellular systems, coordinated MANET MAC protocols need specialized spatial reuse and channel borrowing mechanisms that address the unique characteristics of MANETs in order to provide as high bandwidth efficiency as their uncoordinated counterparts. Due to node mobility and the dynamic nature of the sources in a MANET, the network load oftentimes is not uniformly distributed. In this chapter, we propose two algorithms to cope with the non-uniform load distributions in MANETs:

- a light weight distributed dynamic channel allocation algorithm based on spectrum sensing, and
- a cooperative load balancing algorithm in which nodes select their channel access providers based on the availability of the resources.

We incorporate these two algorithms for managing non-uniform load distribution in MANETs into the MH-TRACE framework. Although MH-TRACE incorporates spatial reuse, it does not provide any channel borrowing or load balancing

mechanisms and thus does not provide optimal support to dynamically changing conditions and non-uniform loads. Hence, we apply the dynamic channel allocation and cooperative load balancing algorithms to MH-TRACE, creating the new protocols of DCA-TRACE, CMH-TRACE and the combined CDCA-TRACE.

CDCA-TRACE is a novel MAC protocol that maintains the same energy efficiency and channel regulation principles of MH-TRACE while enabling dynamic and scalable channel assignment in addition to cooperative load balancing. Instead of message exchanges between the channel regulators (CHs), CDCA-TRACE utilizes spectrum sensing to keep track of channel usage in nearby clusters. This feature minimizes the overhead found in dynamic channel allocation schemes for cellular networks and makes CDCA-TRACE suitable for MANETs. CDCA-TRACE also incorporates cooperation among the member nodes to improve the distribution of the load among the CHs and complements dynamic channel allocation to enhance the service rate.

The contributions of the chapter are: i) we propose a light weight dynamic channel allocation scheme for cluster-based mobile ad hoc networks; ii) we propose a cooperative load balancing algorithm; iii) we incorporate these two algorithms into our the TRACE framework leading to DCA-TRACE and CMH-TRACE; and iv) we combine both algorithms leading to CDCA-TRACE that provides better support for non-uniform load distributions. We compare the performance of these algorithms for varying network loads.

The rest of this chapter is organized as follows. Section 4.2 presents the dynamic channel allocation through spectrum sensing and cooperative load balancing algorithms in detail. Section 4.3 discusses the adaptation of these algorithms in the TRACE framework. In Section 4.4, the performance of CDCA-TRACE, DCA-TRACE, CMH-TRACE, MH-TRACE and IEEE 802.11 are compared for various network topologies. Finally, we summarize the results of the chapter in Section 4.5.

4.2 Bandwidth Efficiency Techniques for Coordinated MAC Protocols

In this section we describe the lightweight dynamic channel allocation mechanisms based on channel sensing and the cooperative load balancing algorithms. We begin with a discussion of our assumptions:

- Single transceiver: The nodes in the network are equipped with a transceiver that can operate in one of two modes: transmission or reception. Nodes cannot simultaneously transmit and receive.
- Channel Sensing: The receiver node is able to detect the presence of a carrier signal and measure its power even for messages that cannot be decoded into a valid packet.
- Collisions: In the case of simultaneous transmissions in the system, neither of the packets can be received unless one of the transmissions captures the receiver. The receiver can be captured if the power level of one of the transmissions is significantly larger than the power level of all other simultaneous transmissions. Such a capturing mechanism is the driving factor of the advantages gained through channel reuse.
- Channel Coordinators: The channel resources are managed and distributed by channel coordinators. These coordinators can be ordinary nodes that are selected to perform the duty, or they can be specialized nodes. The channel is provided to the nodes in the network for their transmission needs by these channel coordinators. The system is also assumed to be a closed system where all the nodes comply with the channel access rules.

Networks operating under these assumptions and incorporating a channel reuse scheme can achieve relatively higher bandwidth efficiency under uniform network

loads. However, the system needs additional mechanisms to tackle the problem of non uniform distribution of the network load.

4.2.1 Dynamic Channel Allocation Algorithm

The first mechanism that we propose is a dynamic channel allocation (DCA) algorithm similar to the ones that exist in cellular systems. Under non-uniform loads, it is crucial for the MAC protocol to be flexible enough to let the unused bandwidth be allocated to the controllers in the heavily loaded region(s).

Cellular systems usually handle channel allocation through message exchanges between the cell towers. However, these messages would be too costly for a MANET system due to the highly dynamic behavior of the network. Instead, we adopt a dynamic channel borrowing scheme that utilizes spectrum sensing.

In this algorithm, the channel controllers continuously monitor the power level in all the available channels in the network and assess the availability of the channels by comparing the measured power levels with a threshold. If local load increases beyond local capacity, provided that the measured power level is low enough, the channel coordinator starts using the channel with the lowest power level measurement. Once the channel coordinator starts using the channel, its transmission increases the power level measurement of that channel for nearby controllers, which in turn prevents them from accessing the same channel. Similarly, as the local network load decreases, controllers that do not need some channels stop the transmissions in that channel, making it available for other controllers.

In this dynamic channel allocation algorithm, channel coordinators react to the increasing local network load by increasing their share of bandwidth. Although being effective in providing support for non-uniform network loads, the reactive response taken by the channel coordinators increases the interference in the entire

system.

4.2.2 Cooperative Load Balancing

The DCA algorithm approaches the problem of non-uniform load distribution from the perspective of the channel coordinators. The same problem can also be approached from the perspective of ordinary nodes in the network. This cooperative behavior smooths out mild non-uniformities in the load distribution without the need for the adjustments at the channel coordinator side.

The load on the channel coordinators originate from the demands of the ordinary nodes. Many nodes in a network have access to more than one channel coordinator. The underlying idea of the cooperative load balancing algorithm is that the active nodes can continuously monitor the channel usage and switch from heavily loaded coordinators to the ones with available resources. These nodes can detect that the channels available at the channel coordinator are depleted and shift their load to the channel coordinators with more available resources. The resources vacated by the nodes that switch can be used for other nodes that do not have access to any other channel coordinators. This increases the total number of nodes that access the channel and hence increases the throughput.

4.3 Applying Distributed Channel Allocation and Cooperative Load Balancing to TRACE

4.3.1 Dynamic Channel Allocation for TRACE

In MH-TRACE, each CH operates in one of the frames in the superframe. Since the number of data slots is fixed, the CH can only provide channel access to a limited number of nodes. Due to the dynamic structure of MANETs, one CH

may be overloaded while others may not be using their data slots. In that case, although there are unused data slots in the superframe, the overloaded CH would provide channel access only to a limited number of nodes, which is equal to the number of data slots per frame, and the CH would deny the channel access requests of the others. Thus, the system needs a dynamic channel allocation scheme to provide access to a larger number of nodes.

DCA-TRACE lets CHs operate in more than one frame per superframe, if they are overloaded. Instead of choosing and operating in the least noisy frame as in MH-TRACE, in DCA-TRACE, based on the load level, CHs decide on the number of frames they require and choose that many frames from the least noisy frames.

DCA-TRACE includes two additional mechanisms on top of MH-TRACE: i) a mechanism to keep track of the interference level from the other CHs in each frame; and ii) a mechanism to sense the interference level from the transmitting nodes in each data slot in each frame. These mechanisms make use of existing messages and do not add complexity other than slightly increasing memory requirements to store the interference levels.

The MH-TRACE structure provides CHs the ability to measure the interference from other CHs in their own frame and in other frames through listening to the medium in the CA slot of their own frame and the Beacon slots of other frames. In MH-TRACE, CHs use this mechanism to choose the minimum interference frame for themselves. DCA-TRACE makes use of the same structure. However, in order to accommodate temporary changes in the interference levels that may occur due to CH resignation or unexpected packet drops, an exponential moving average update mechanism is used to determine the current interference levels in each frame. At the end of each frame, the interference level of the Beacon and CA slots are updated with the measured values in that frame using

$$I_{k,t} = \begin{cases} M_{k,t} & \text{if } I_{k,t-1} < M_{k,t}; \\ (1-\alpha)\, I_{k,t-1} + \alpha M_{k,t} & \text{o.w.}, \end{cases} \tag{4.1}$$

where $I_{k,t}$ and $I_{k,t-1}$ are the interference levels of the k^{th} slot in the current and the previous superframe, respectively. $M_{k,t}$ is the measured interference level in that slot, and α is a smoothing factor, which is set to 0.2 in our simulations. The interference level of the frame is taken as the maximum interference level among the interference levels of the Beacon and CA slots.

In DCA-TRACE, CHs mark a frame as unavailable if there is another cluster that uses the frame and resides closer than a certain threshold, Tr_{int}, measured through the high interference value of that frame. Even under high local demand, CHs refrain from accessing these frames that have high interference measurements, in order to protect the stability of the clustering structure and the existing data transmissions. At the end of each superframe, CHs determine the number of frames that they need to access, m, based on the reservations in the previous frame. Depending on the interference level of each frame, they choose the least noisy m frames that have an interference value also below a common threshold, Th_{intf}. If the number of available frames is less than m, the CHs operate only in the available frames. In our simulations, Th_{intf} is set to a level that corresponds to the power of a packet $350m$ away from the transmitter at the given transmission power level and propagation model.

Another mechanism that DCA-TRACE adds on top of MH-TRACE is the dynamic assignment of data slots. In MH-TRACE, data slots are assigned in a sequential order. On the other hand, since DCA-TRACE introduces channel borrowing, the CH has to refrain from reallocating a data slot that has been borrowed by another CH and instead must allocate another data slot that has a lower interference value. In order to do this, CHs keep track of the interference levels of each IS slot of each frame in the superframe. In order to accommodate

temporary changes, the exponential moving average smoothing mechanism of (4.1) is also used for IS frames.

Knowing the interference values of all IS slots, the CH assigns the available data slots to the nodes that request channel access beginning with the slot that has the lowest interference value. In the case of a channel borrower, this mechanism allows the CHs that operate in a frame to assign slots that are not used by other CHs as much as possible.

DCA-TRACE is similar to cognitive radio systems. However, since we do not distinguish between the primary CH of the frame and the CH that borrows a channel, we treat them equally in having access to the available data slots in any frame.

4.3.2 Collaborative Load Balancing for TRACE

In the previous section, we described DCA-TRACE, which tackles non-uniform load distribution by allowing the CHs to access more than one frame in the superframe. The same problem can also be tackled from the member nodes' perspective.

In our previous work [101], we determined that the majority of the nodes in a TRACE network are in the vicinity of more than one CH (they are in the vicinity of two, three or four CHs with probabilities of 52%, 19% and 1%). The nodes that are in the vicinity of more than one CH can ask for channel access from any of these CHs. Using a cooperative approach and a clever CH selection algorithm on the nodes, the load can be migrated from heavily loaded CHs to the CHs with more available resources.

In the TRACE protocols, nodes contend for channel access from one of the CHs that have available data slots around themselves. After successful contention, they do not monitor the available data slots of the CHs around them. Due to the dynamic nature of the network load, a cluster with lots of available data slots may

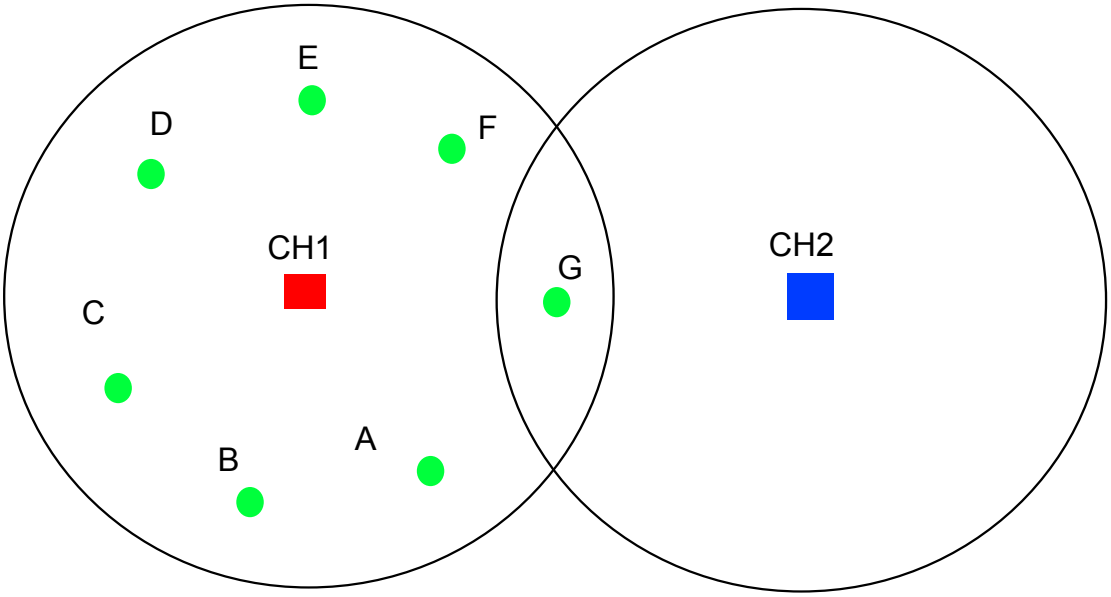


Figure 4.1: Demonstration of a scenario for the collaborative load balancing algorithm.

become heavily loaded during a data stream. In order to tackle this issue, nodes should consider the load of the CH not only when they are first contending for channel access but also after securing a reserved data slot and throughout their data stream.

In order to further elaborate this, consider Fig. 4.1. Nodes A-G are source nodes and need to contend for data slots from one of the CHs. Each CH has 6 available data slots. In MH-TRACE, if their contentions go through in alphabetical order, node G would mark $CH1$ as full and would ask for channel access from $CH2$. However, if node G secures a data slot from $CH1$ before any of the nodes A-F, one of the source nodes would not be able to access to the channel.

In DCA-TRACE, once $CH1$ allocates all of its available slots, it triggers the algorithm to select an additional frame. However, accessing one additional frame might not always be possible, if the interference levels on all the other frames are too high. Moreover, accessing additional frames increases the interference in

the Beacon and Header slots of these frames and may trigger CH resignations and reselections in the rest of the network that temporarily disturbs ongoing data streams on the resigned CHs. Finally, accessing additional frames increases interference on the IS and data slots of the new frame and decreases the potential extent these packets can reach.

In order to overcome these difficulties, we propose CMH-TRACE and CDCA-TRACE, which add cooperative CH monitoring and reselection on top of MH-TRACE and DCA-TRACE, respectively. In CMH-TRACE and CDCA-TRACE, nodes continuously monitor the available data slots at the CHs around themselves announced by the Beacon messages. When all the available data slots for a CH are allocated, with a probabilty p, the active nodes attempt to trigger the cooperative load balancing algorithm. When the cooperative load balancing is triggered, the node that is currently using a data slot from the heavily loaded CH contends for data slots from other nearby CHs while keeping and using its reserved data slot until it secures a new data slot from another CH.

Cooperative load balancing does not alter the clustering structure, and it is desirable over selecting an additional frame at the CH. However, cooperative balancing does not completely solve the hot spot problem. The source nodes may not be in the vicinity of another CH, and hence their load cannot be transferred to another CH. In that case, triggering the DCA algorithm is required. Thus, in CDCA-TRACE, we include the additional frame selection algorithm of DCA-TRACE with some delay. A fully loaded CH resets a counter, $N_{DCA} = 0$, and starts incrementing it at the beginning of each superframe while it remains fully loaded. The CH attempts to (subject to the interference levels in the frames) access an additional frame when $N_{DCA} >= T_{DCA}$. This provides time for the active member nodes to trigger the cooperative load balancing algorithm and transfer their load to nearby CHs. In our simulations we used a threshold value of $T_{DCA} = 3$.

4.4 Performance Evaluation

The size of the region over which the nodes are located, the number of nodes in the network, and their data generation patterns are all important in optimizing the design parameters [101]. However, due to the dynamic nature of MANETs this information might not be available a priori, and some of these parameters may change over the course of the network lifetime. Thus, it is necessary for the protocols to dynamically adjust to changing conditions.

In uncoordinated MAC protocols such as IEEE 802.11 [102], the common channel resource is shared among the nodes in the network based on carrier sensing. This simple behavior is well suited for handling any non-uniformities in the load distribution. However, these protocols do not scale well as the load in the network increases, due to the simple carrier sensing mechanism. On the other hand, coordinated MAC protocols such as the TRACE protocols are better suited for heavy load scenarios. Unlike MH-TRACE, the channel allocation for DCA-TRACE and CDCA-TRACE can be adjusted on the fly, making them more flexible protocols compared to their predecessor. By adjusting the channel access scheme, they are more capable of adapting to: i) shrinking network dimensions, and ii) non-uniformities in load distribution.

Due to the movement of the nodes in the network, the diameter of the network may shrink over the course of network operation. At one extreme, when the largest distance between any two nodes in the network is below the communication radius, nodes form a single hop connected network. The bandwidth efficiency of MH-TRACE sharply reduces for such an operation, as MH-TRACE cannot adjust the number of frames in each superframe dynamically, and each CH can only utilize a single frame per superframe. However, the dynamic channel allocation mechanism of DCA-TRACE enables adaptation of the protocol to this environment by letting the single CH access all the frames and all the data slots. We investigate this

scenario in Section 4.4.1. Cooperative load balancing is not effective in this simple scenario since there is only a single CH. Hence, CMH-TRACE and CDCA-TRACE perform similar to their predecessors, namely MH-TRACE and DCA-TRACE, respectively. Thus, we omit the CMH-TRACE and CDCA-TRACE results for this scenario.

Due to the dynamic environment, the network load might not be distributed uniformly among the clusters. In Section 4.4.2, we study a scenario in which the network load is localized in a limited portion of the network. We investigate the effects of cooperative load balancing and dynamic channel allocation and compare CMH-TRACE and DCA-TRACE with MH-TRACE. We also analyze the combined improvements of both algorithms through CDCA-TRACE. Finally, we compare the performance of all of these protocols with a typical uncoordinated protocol, IEEE 802.11.

We study random load distributions in Section 4.4.3. The performances of CDCA-TRACE, DCA-TRACE, CMH-TRACE, MH-TRACE and IEEE 802.11 are compared in a scenario with randomly selected source nodes in a multi-hop network with randomly distributed mobile nodes.

For comparison purposes, we conduct ns-2 simulations of all of the protocols utilizing simple network and transport layer protocols that provide local broadcasting. Packets are assumed to be destined to the local neighborhood. All the nodes in the vicinity of the transmitter receive the packet as long as the power levels permit successful decoding. Note that the IEEE 802.11 protocol must thus use the ad hoc DCF mode in which the RTS/CTS and ACK mechanisms are disabled.

The source application generates real-time traffic in constant bit-rate (CBR) mode, coded at 32 Kbps. 100 byte long packets are generated from this application every 25ms. Due to real-time communication constraints, packets become obsolete and are discarded at the source if they are not sent within 25 ms. The channel

rate is set to 2 Mbps. The superframe time is matched to the source packet generation period of 25 ms. Each superframe consists of 6 frames with 6 data slots each. Starting at $t_s = 2$ s (80th superframe), every 5 superframes one source node starts generating packets, thereby increasing the number of active sources and the load in the network.

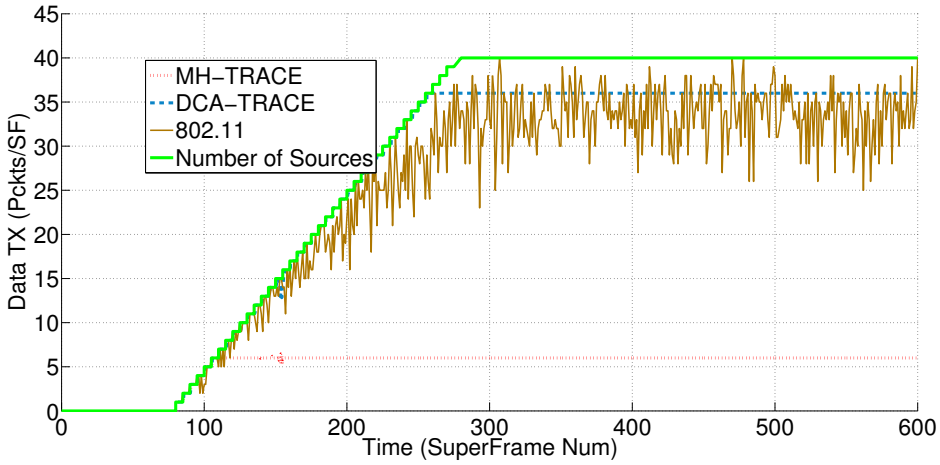
For node mobility, the random way-point mobility model [99] [100] is used, where the node speeds are chosen from a uniform random distribution between 0.0 m/s and 5.0 m/s with zero pause time. The energy model discussed in [101] and the default propagation model that is available in ns-2 [95] are used. We used a constant transmit power that results in a maximum receiving range of 250 m under zero interference. In the case of interference, all packets received during the interference period are dropped unless one of the packets captures the receiver with a power value at least 10 times larger than the power of any interfering packet.

Various simulation parameter settings are summarized in Table 4.1.

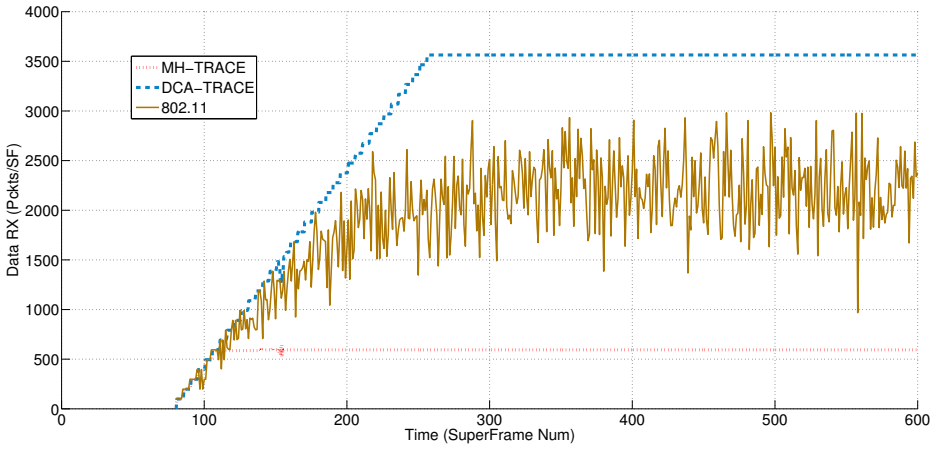
4.4.1 Single Hop Network

In this section, the performance of DCA-TRACE, MH-TRACE and IEEE 802.11 are compared for a single hop connected network in which 100 nodes, including 40 sources, are stationary and distributed over a 100 m x 100 m region with a uniform grid formation. Considering a receiving range of 250 m, the nodes form a single hop network. Fig. 4.2 presents the average number of transmitted packets per superframe and the average number of received packets per superframe, averaged over 80 iterations throughout the simulation duration of 20 seconds.

We omitted CDCA-TRACE and CMH-TRACE results for this scenario. Due to the small size of the network, the TRACE framework operates with a single



(a)



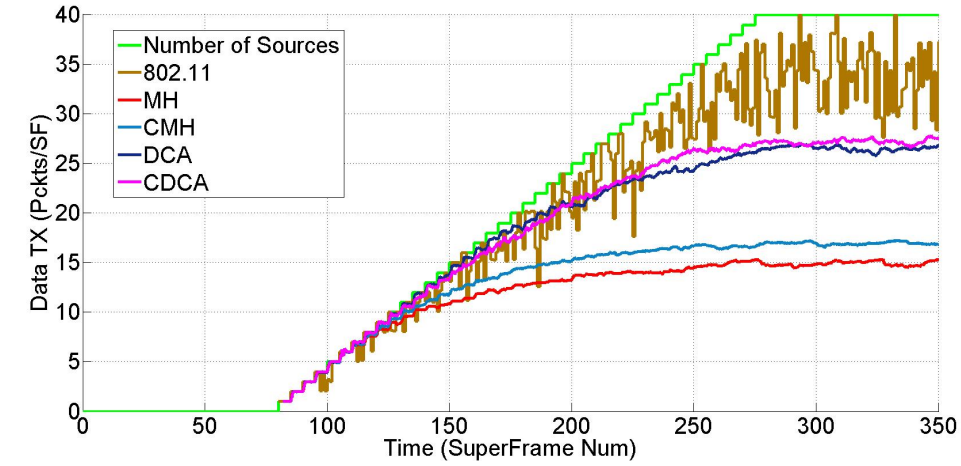
(b)

Figure 4.2: (a) Average number of data transmissions per superframe for a single hop network. (b) Average number of data receptions per superframe for a single hop network.

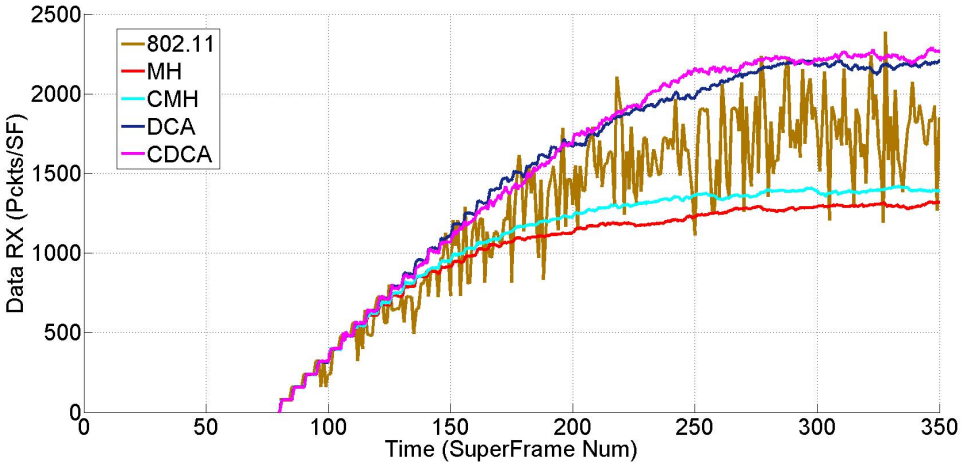
cluster. Thus, collaborative load balancing is ineffective for this scenerio, and CDCA-TRACE operates similar to DCA-TRACE with the exception of reaction time. In CDCA-TRACE, CHs wait 3 superframes before accessing to an additional frame. This duration is smaller than the rate of increase in network load and hence does not alter the results. Confirming this, we observed same performance in both protocols and omitted the CDCA-TRACE results. For similar reasons, the CMH-TRACE results are also omitted.

Due to the CH resignation mechanism in MH-TRACE, only a single CH can operate in such a scenario. Since each CH only accesses one of the frames in each superframe and hence has access to only 6 data slots, the number of transmissions per superframe (TX) saturates at a value of 6. The access time for all the remaining frames is not used and is thus wasted. On the other hand, DCA-TRACE adapts to this situation by letting the single CH access all 6 frames and all the data slots. Hence, DCA-TRACE saturates at a value of 35.6 providing channel access to 6 times more nodes. Since channel access is fully coordinated under a single CH, both MH-TRACE and DCA-TRACE eliminate all the collisions and a similar gain can be observed also in the number of receptions per superframe (RX) as shown in Fig. 4.2(b).

Similarly, as the number of source nodes increases, TX increases in IEEE 802.11. However, due to the lack of coordination, additional source nodes in IEEE 802.11 increase the collisions in the network. Hence, the number of receptions does not increase in the same proportion as the number of transmissions. Although the collision probability of 802.11 DCF is large as shown in [103], due to receiver capturing, many successful receptions are possible even under simultaneous transmissions. Still, at the maximum load, IEEE 802.11 yields around 30% fewer receptions compared to DCA-TRACE.



(a)



(b)

Figure 4.3: (a) Average number of data transmissions per superframe for localized load distribution. (b) Average number of data receptions per superframe for localized load distribution.

4.4.2 Localized Load Distribution

In this section, the performances of CDCA-TRACE, DCA-TRACE, CMH-TRACE, MH-TRACE and IEEE 802.11 are compared for a network in which 40 source nodes are stationary and distributed over a 100m x 100m square centered in the middle of the 1000m x 1000m region with a uniform grid formation. The remaining 200 nodes are mobile and deployed randomly. Fig. 4.3 presents the average number of transmitted packets per superframe, TX, and the average number of received packets per superframe, RX, averaged over 80 iterations throughout the simulation duration of 20 seconds.

In order to investigate the effect of dynamic channel allocation, we compare DCA-TRACE and MH-TRACE. In the beginning of the simulation, the number of active sources in the network is low and there are unused data slots in the frames of almost all the CHs. Hence, TX increases at the same pace in all four protocols as the number of sources increases. As the number of sources increases, in MH-TRACE, CHs allocate available data slots to the source nodes. After all available data slots are assigned, further channel access requests are denied and hence TX converges to around 15. This number is greater than the number of data slots in one frame as multiple CHs can provide access to the source nodes depending on random selection of the CHs. On the other hand, the average number of transmissions per superframe in DCA-TRACE converges around a value of 26. The dynamic channel allocation mechanism of DCA-TRACE adapts the channel allocation based on the load and enables the protocol to provide channel access to 73% more nodes compared to MH-TRACE at the highest load level of 40 source nodes.

Compared to MH-TRACE, DCA-TRACE also leads to a gain of similar magnitude in the number of receptions, as CHs choose the frames they access and the data slots they allocate based on the interference levels in the medium. The

average number of data receptions in MH-TRACE and DCA-TRACE are around 1300 and 2175, respectively. Thus, DCA-TRACE leads to a gain of 67% in the number of receptions compared to MH-TRACE.

Next, we focus on the cooperative load balancing algorithm by comparing MH-TRACE and CMH-TRACE. Both protocols converge as the load in the network increases. However, at the highest load, TX and RX in CMH-TRACE converge to values 10% higher than TX and RX in MH-TRACE.

The improvements of cooperative load balancing and dynamic channel allocation are combined in CDCA-TRACE. Under high load, CDCA-TRACE improves TX by 3% and 80% compared to DCA-TRACE and MH-TRACE, respectively. Similarly, RX is improved in CDCA-TRACE by 3% and 77% compared to DCA-TRACE and MH-TRACE, respectively.

Next, we compare the performance of CDCA-TRACE and IEEE 802.11.Unlike the TRACE protocols, the overhead for signaling between member nodes and the CHs, namely Beacon, CA, contention slots, and header, does not exist in IEEE 802.11. Moreover, IEEE 802.11 does not divide the channel spatially, and hence it is not affected by the larger region over which the passive nodes are distributed. The entire bandwidth is shared only among the active nodes in the smaller localized region through the channel sense mechanism. On the other hand, TRACE dynamically selects and maintains CHs in the entire network, including the passive part. Hence, at the maximum load, IEEE 802.11 can provide channel access to 33 nodes, which is 22% higher than the average number of nodes for which CDCA-TRACE provides channel access. However, some of the transmissions cannot be received at the receiver side due to collisions. The lack of coordination in IEEE 802.11 leads to a larger number of collisions compared to CDCA-TRACE. Thus, at the load level of 40 source nodes, RX in CDCA-TRACE is 29% higher compared to 802.11, which has an average RX value of 1750.

As mentioned previously, another key performance measure for the MAC pro-

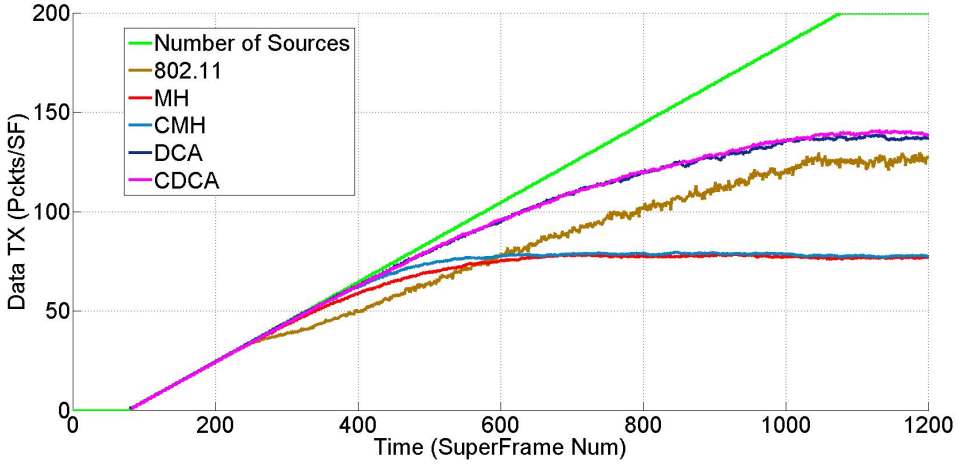
tocols serving MANETs is energy consumption. A desirable MANET MAC protocol should be not only bandwidth efficient but also energy efficient. CDCA-TRACE consumes 56% less energy compared to IEEE 802.11, as can be observed from the data in Table 4.2.

Inconsistency in packet delays is not desirable from the real time communication perspective, since the construction of the stream would be problematic. A high delay variation would require a large anti-jittering buffer, which would increase overall latency of a real-time application. In order to measure jitter, we consider inter packet delay variation (IPDV) for consecutive packets, as described in [104]. Thanks to the coordinated channel access, CDCA-TRACE provides smoother operation and leads to 4 orders of magnitude smaller average absolute IPDV compared to IEEE 802.11, as can be observed from the data in Table 4.3. The reduction in the IPDV makes CDCA-TRACE more suitable for real time applications compared to IEEE 802.11.

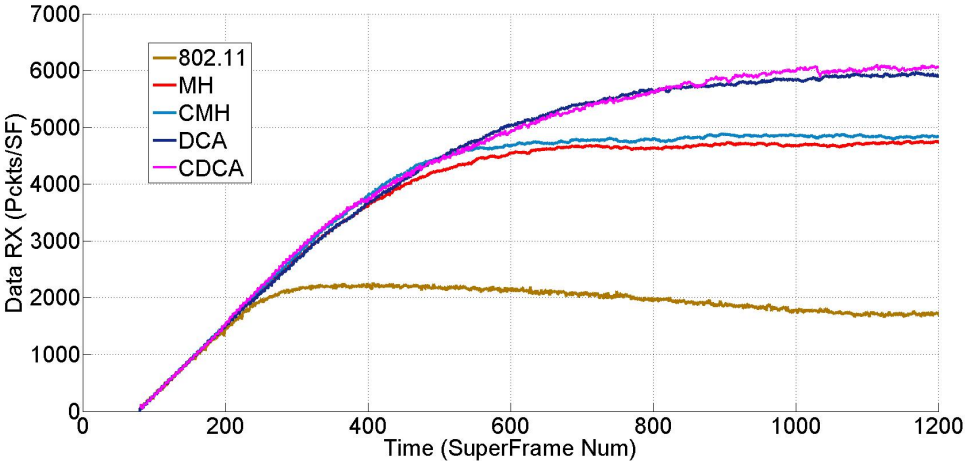
4.4.3 Random Load Distribution

In this section, the performance of CDCA-TRACE, DCA-TRACE, CMH-TRACE, MH-TRACE and IEEE 802.11 are compared for a network of 400 nodes including 200 source nodes. All the nodes are mobile with randomly distributed initial locations over a 1000 m x 1000 m region. Fig. 4.4 presents the average number of transmitted packets per superframe, TX, and the average number of received packets per superframe, RX, averaged over 80 iterations throughout the simulation duration of 60 seconds.

Similar to the previous scenario, as the network load increases, with a decreasing pace, all protocols provide channel access to more nodes, resulting in an



(a)



(b)

Figure 4.4: (a) Average number of data transmissions per superframe for random load distribution. (b) Average number of data receptions per superframe for random load distribution.

increase in TX up to a saturation point. Beyond this point, TX saturates and stays constant as the number of sources increases.

Thanks to the dynamic channel allocation mechanism, DCA-TRACE is not affected from the non-uniformities in the load distribution as much as MH-TRACE. Hence, the rate of increase of TX is higher for DCA-TRACE compared to MH-TRACE.

Dynamic channel allocation also helps dynamically adjust the spatial reuse ratio on the fly based on the channel interference measurements. Under low loads, it allows the protocol to operate with reduced interference by reducing the level of spatial reuse used by MH-TRACE. However, under high loads spatial reuse is increased up to the point limited by the frame availability interference threshold, Tr_{int} [1].

Thanks to the dynamic channel allocation, DCA-TRACE can provide channel access to a larger number of nodes compared to MH-TRACE, as seen by the higher saturation point in Fig. 4.4(a). At the highest simulated load level of 200 source nodes, DCA-TRACE provides channel access to an average of 139 nodes while MH-TRACE can only provide channel access to an average of 77.7 nodes. Hence, DCA-TRACE provides channel access to 79% more nodes under high load in a multi-hop scenario compared to MH-TRACE. However, due to the increased interference caused by the higher spatial reuse, the number of collisions also increases. Thus under high load, the improvement in RX is lower than the gain in TX. At the highest load, RX in DCA-TRACE is 19% higher than that in MH-TRACE. Due to the same phenomena, under low levels of traffic load, the number of receptions in DCA-TRACE is slightly lower than the number of receptions in MH-TRACE, although the numbers of transmissions are approximately equal. However, the maximum difference is around 1% and thus is negligible.

[1]Note that, this threshold setting can be arbitrated for a tradeoff between fewer collisions and higher TX

Next, we focus on the cooperative load balancing by comparing CMH-TRACE and MH-TRACE. TX increases faster for CMH-TRACE compared to MH-TRACE since non-uniformities in source distribution caused by the random source selection are smoothed out in CMH-TRACE. As observed in Fig. 4.4(a), CMH-TRACE improves TX by as much as 4%. However, for very large network loads, all the clusters in the network are fully occupied. Nodes cannot use cooperative load balancing as none of the clusters in their neighborhood have available resources. Thus, both protocols converge to the same value under very high network loads.

Looking at the combined performance of dynamic channel allocation and cooperative load balancing, we compare CDCA-TRACE and DCA-TRACE. In terms of TX, the effect of the addition of cooperative load balancing is only marginal. Both methods are effective in tackling the problem of non-uniform load distribution for medium load levels, however, cooperative load balancing is not effective when the network load is very high. Nonetheless, cooperative load balancing does not alter spatial reuse and hence does not increase the interference and the collisions. Thus, in Fig. 4.4(b), an improvement of 2% can be observed in RX for CDCA-TRACE compared to the RX for DCA-TRACE.

Furthermore, we also compare the performances of CDCA-TRACE and IEEE 802.11. Despite the clustering constraints and the signaling overhead of TRACE, CDCA-TRACE outperforms IEEE 802.11. In Fig. 4.4(a), under heavy load, TX in CDCA-TRACE is 14% higher than that of IEEE 802.11, which provides channel access to only 122 nodes at a load of 200 source nodes. In addition to this, IEEE 802.11 suffers from packet collisions due to the lack of coordination. Packet collisions increase with increasing network load, and an increased number of transmissions does not necessarily correspond to an increased number of receptions. As observed in Fig. 4.4(b), for IEEE 802.11, RX starts to decrease as TX increases above 50 (400th superframe) and reduces to 1700 at the maximum number of sources.

Furthermore, being a coordinated protocol, CDCA-TRACE keeps the advantages of low energy consumption and very low jitter. The average energy consumption per node per second for all three protocols are presented in Table 4.2. DCA-TRACE consumes only 54% of the energy consumed by IEEE 802.11, even though the number of receptions is significantly larger. The 16% increase in the average energy consumption in DCA-TRACE compared to MH-TRACE is the result of the increased number of transmissions and receptions.

Table 4.3 presents the average absolute IPDV for all three protocols averaged over all transmitter and receiver pairs and over the simulation set. DCA-TRACE leads to a 3 orders of magnitude smaller average absolute IPDV compared to 802.11, thanks to the channel reservation scheme in TRACE. Compared to MH-TRACE, DCA-TRACE has a larger average absolute IPDV due to the CHs actively monitoring IS slots for minimum interference and changing slot reservations accordingly with changing conditions. However, the trade-off between minimum interference point of operation and minimum packet delay variation can be resolved according to the requirements of the application by modifying the slot reservation mechanism at the CHs. For instance, CHs could preserve the transmission schedule unless the interference of the slot is above a threshold in order to decrease the variation in the packet delays.

To sum up, under heavy and randomly distributed loads, CDCA-TRACE not only increases the number of source nodes that can get channel access compared to an uncoordinated protocol, IEEE 802.11, but it also reduces the number of collisions, average energy consumption, and average absolute IPDV drastically, leading to a higher number of receptions and significant energy savings, thanks to the coordination mechanisms.

4.5 Summary

In this chapter, we studied the problem of non-uniform load distribution in mobile ad hoc networks. We proposed a light weight dynamic channel allocation algorithm and a cooperative load balancing algorithm. The dynamic channel allocation works through carrier sensing and does not increase the overhead. It has been shown to be very effective in increasing the service levels as well as the throughput in the system with minimal effect on energy consumption and packet delay variation. The cooperative load balancing algorithm has less impact on the performance compared to the dynamic channel allocation algorithm. We showed that these two algorithms can be used simultaneously, maximizing the improvements in the system. The combined system has been shown to perform at least as well as the systems with each algorithm alone and perform better for many scenarios.

We proposed a novel MAC protocol, CDCA-TRACE, that combines dynamic channel allocation and cooperative load balancing algorithms into the TRACE framework. CDCA-TRACE, which controls channel utilization through the dynamically selected distributed channel coordinators, is compared to IEEE 802.11, which controls channel utilization in a fully distributed manner. Having channel coordinators, CDCA-TRACE is shown to require about 60% less energy consumption and 3 orders of magnitude lower packet delay variation compared to IEEE 802.11. Moreover, thanks to the dynamic channel allocation algorithm and the cooperative load balancing algorithm, in a network with randomly distributed sources, CDCA-TRACE provides channel access to 14% more nodes and improves the number of receptions $3.5x$ compared to IEEE 802.11.

CDCA-TRACE does not implement any packet routing functionality. In the TRACE family of protocols, NB-TRACE and MC-TRACE implement network-wide broadcasting and multicasting, respectively. These protocols can be easily

adjusted to work with CDCA-TRACE since the basic MAC functionality of both of these protocols is based on MH-TRACE. Hence CDCA-TRACE adds the dynamic channel allocation and cooperative load balancing functionality to the entire TRACE family, resulting in a complete suite of protocols.

CDCA-TRACE ensures dynamic and efficient resource utilization in the MAC layer. Having studied dynamic and efficient resource utilization in the MAC layer, we turn our attention to the routing layer. Improper designs in data dissemination schemes can also cause inefficient resource utilization. In the next chapter, we focus on one-to-many group communications and investigate efficient data dissemination under varying conditions. Then, we combine various one-to-many group communications into a unified framework that picks the data dissemination method leading to efficient resource utilization.

Table 4.1: Simulation Parameters

Parameter	Value
Channel rate	2 Mbps
Transmission Power	282 mW
Propagation Model	Two way ground
Reception power threshold	3.652×10^{-10} ($\sim$ 250 m)
Carrier sensing power threshold	1.4×10^{-12} ($\sim$ 1000 m)
Receiver capture threshold (Th_{cap})	10
Traffic type	CBR
Data packet payload size	100 bytes
Inter packet generation period	25 ms
Mobility model	Random way-point
Range of node speeds	0.0 m/s - 5.0 m/s
Superframe length	25 ms
Number of frames per superframe	6
Number of data slots per frame	6
Number of contention slots per frame	9
Exponential smoothing factor (α)	0.1
Frame availability threshold (Th_{intf})	0.951×10^{-10} ($Tr_{int} \approx$ 350 m)

Table 4.2: Average energy consumption per node per second (J/s)

	802.11	MH-TRACE	DCA-TRACE	CDCA-TRACE
Random Load Distribution	0.235	0.109	0.127	0.124
Localized Load Distribution	0.205	0.069	0.090	0.091

Table 4.3: Average Absolute IPDV (s)

	802.11	MH-TRACE	DCA-TRACE	CDCA-TRACE
Random Load Dist.	9.82×10^{-3}	8.91×10^{-8}	4.85×10^{-6}	2.85×10^{-6}
Localized Load Dist.	6.29×10^{-3}	1.34×10^{-10}	7.98×10^{-7}	6.42×10^{-7}

5 Efficiency in Data Dissemination Schemes

5.1 Introduction

In Chapter 4, we presented a novel MAC protocol for MANETs that ensures efficient use of resources through a dynamic channel allocation mechanism. In this chapter, we focus on the routing layer and develop efficient data dissemination schemes.

Both network-wide broadcasting and multicasting are among the most frequently used data dissemination schemes in MANETs. Network-wide broadcasting protocols deliver the data to all the nodes in the network whereas in multicasting data is delivered to a selected subset of the nodes, called a multicast group. By limiting the data dissemination to the multicast group, multicast protocols prevent redundant retransmissions of the data. On the other hand, the overhead in multicasting protocols is more than the overhead in network-wide broadcasting protocols. In this chapter, our objective is to investigate the trade-offs between multicasting and broadcasting in order to determine the conditions that make one of them preferable over the other.

Towards the goal of investigating the trade-off between multicasting and broadcasting, we perform extensive simulation studies on a chosen protocol from each

class: Network-wide broadcasting through time reservation using adaptive control for energy efficiency (NB-TRACE) [72] for broadcasting; and Multicasting through time reservation using adaptive control for energy efficiency (MC-TRACE) [73] for multicasting. The first reason for choosing these protocols is that they have been shown to outperform many other protocols in their class. Moreover, those protocols are built on top of the same MAC structure, and their sensitivity to MAC layer issues such as mobility and link errors are similar. Finally, the data maintained by the protocols are very similar to each other, and any additional burden of multicasting can directly be observed. Consequently, the protocols can be combined into a unique framework and coexist simultaneously. Ultimately, this approach yields a unified protocol where the better approach (broadcasting or multicasting) can be used depending on the situation.

The rest of the chapter is organized as follows. We analyze the efficiency of the protocols for a sample scenario in Section 5.2. The effect of node density on the relative efficiency of the protocols is investigated in Section 5.3. Finally, we summarize the results of the chapter in Section 5.4.

5.2 Comparing Multicast and Broadcast

In general, multicasting protocols eliminate redundant retransmissions by confining the data dissemination to a limited area. However, this comes with the additional cost of overhead to keep the data distribution structure alive. Intuitively, while multicasting is expected to be a more efficient method of data distribution for small group sizes, broadcasting would be more efficient for large group sizes. In this section, we show that this is indeed the case by analyzing broadcasting and multicasting through extensive simulations of a select broadcasting protocol, NB-TRACE, and a select multicasting protocol, MC-TRACE.

In particular, the number of multicast group members beyond which NB-

TRACE becomes more efficient for data dissemination, called the cross-over point, is determined for various scenarios. By comparing the simulation results, we can analyze the effect of the total number of nodes in the network and the size of the region in which the nodes are distributed on the value of this cross-over point.

Two performance metrics, energy efficiency and spectrum efficiency, are considered. Specifically, the average energy spent per node per generated packet and the total number of transmissions per generated packet are measured for each scenario to compare the energy efficiency and the spectrum efficiency of the protocols, respectively.

We begin with describing the simulation environment and the parameters selected for the scenarios under concern. Then, for a network of 100 nodes distributed in a 1×1 km^2 area, the bandwidth efficiency of the protocols and their energy consumptions are compared. Finally, the analysis is extended by varying the number of nodes in the network and the size of the area in which nodes are distributed.

5.2.1 Simulation Environment

We conduct ns-2 simulations of NB-TRACE and MC-TRACE under different network scenarios. We used the default energy and propagation (two-ray ground) models in ns-2. Both path loss and interference are taken into account in determining a successful reception. The receiver can receive only those packets whose received power is above a certain threshold. For a successful reception, the receiver has to be within 250m of the transmitter with the given propagation model, reception threshold and transmission power. However, simultaneous transmissions interfere with each other and prevent successful reception. In the case of simultaneous transmissions, a successful reception is only possible if, at the receiver side, the power of one of the packets is 10 times larger than any other packet. The

transceivers are fixed at 2 Mbits/sec data rate.

The packet generation model assumes a 16-bit voice coder that generates 100 byte packets every 25ms. The length of an IS slot is 12 bytes long in NB-TRACE, while it is 15 bytes long in MC-TRACE due to the extra routing information requirements. The superframe period is fixed to the packet generation period, and the number of frames per superframe is fixed at 6 for both NB-TRACE and MC-TRACE. However, due to the extra bits in the IS slots, MC-TRACE has 6 data slots per frame whereas NB-TRACE has 7. Each node transmits or relays the data packets of the stream using one of the available data slots.

In order to have a fair comparison in terms of energy consumption, the concept of group members is introduced to NB-TRACE. Nodes that do not belong to the group do not listen to the data slots of the stream.

The power spent by each node varies according to the operation performed by the node. During successful reception, collision and carrier sensing periods, the node consumes power at the rate of the reception power level. There is also an idle state where only the power needed to run the circuitry is dissipated without any actual packet receptions. The nodes are assumed to turn off any circuitry when they go into the sleep state, where the power consumption is minimal.

All the nodes except the source node are initially distributed according to a uniform random distribution, and during the course of the simulation the nodes move following the random way-point mobility model [99] [100] with node speeds chosen from a uniform random distribution between 0.0 m/s and 5.0 m/s with zero pause time. The source node starts from the center of the region and follows the same random way-point mobility model.

100 repetitions are performed for each scenario, and the presented results show the averages and the standard deviations of the results.

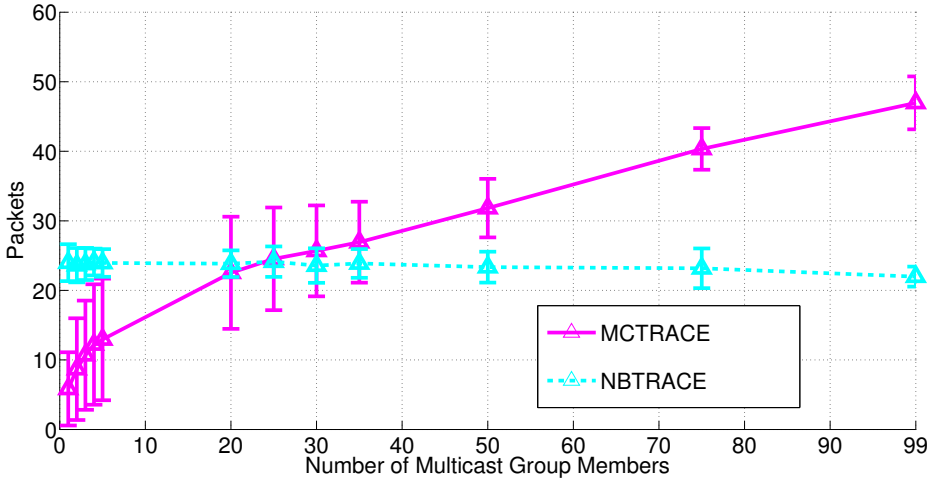


Figure 5.1: Number of transmitted packets per generated packet as the number of group members is varied from 1 to 99 for a network of 100 nodes distributed on a 1×1 km^2 area.

5.2.2 Bandwidth Efficiency

In the TRACE frame structure, there are fixed number of data slots per superframe. The data slots are used to transmit the payload of both the generated data and the relayed data. Any redundant use of the data slots wastes the available network resources and may prevent another stream's information from being disseminated over a region. Thus, efficient use of the data slots is one of the goals of both NB-TRACE and MC-TRACE.

The number of data packet transmissions per generated packet is depicted in Fig. 5.1 for both NB-TRACE and MC-TRACE. The change in the number of multicast members does not affect the number of data transmissions in NB-TRACE significantly since data is broadcasted to the entire network regardless of the locations and number of multicast members. The slight variation in NB-TRACE is due to the additional restriction we impose on the algorithm to make a fair comparison between NB-TRACE and MC-TRACE. We prevent non-group members from listening to the ongoing stream unless they are relaying the data

to save energy. However, this reduces the efficiency of the recovery of link failures through local branch repair mechanisms, since the neighbor nodes that are not in the multicast tree do not have any data in their transmission queue.

Unlike NB-TRACE, in MC-TRACE, the number of transmissions increases as the number of group members increases. Nodes are distributed with an independent identical random uniform distribution[1]. Hence, as the number of group members increases, the size of the multicasting tree increases. Hence, the number of transmissions per generated packet increases.

It can be observed from Fig. 5.1 that for small group sizes the data dissemination is more efficient in MC-TRACE compared to NB-TRACE, and vice versa. NB-TRACE requires 4 times more transmissions compared to MC-TRACE when there is only one group member. At the other extreme, when all the nodes are in the multicast group, the number of data transmissions required using NB-TRACE is half of the transmissions required when using MC-TRACE. The cross-over point of MC-TRACE and NB-TRACE occurs at a multicast group size of 24 nodes, above which NB-TRACE provides more efficient data dissemination and vice versa.

5.2.3 Energy Efficiency

In Section 5.2.2, we compared the number of data transmissions for NB-TRACE and MC-TRACE. The number of transmissions and receptions of other packet types, namely beacon, CA, contention, and header are expected to be comparable. Thus, a similar trade-off that exists in Section 5.2.2 is also expected for the energy consumption metric.

[1]It is well known that the random way point mobility model alters the uniform node distribution assumption as time elapses [105] [106]. However, this effect is negligible in our case since the simulation duration of 100 sec is short enough.

On the other hand, since each IS slot is matched with its corresponding data slot, the difference in the number of data transmissions between protocols is also expected to be observed in the number of IS transmissions. Furthermore, since the length of IS slots in MC-TRACE are longer compared to the ones in NB-TRACE, the energy consumption of MC-TRACE is expected to be even higher. This behavior is expected to shift the energy consumption curve of MC-TRACE in such a way that the cross-over point will occur with fewer multicast group members.

The energy consumption per node per generated packet is depicted in Fig. 5.2 as the number of group members is varied from 1 to 99. The energy consumption increases with an increase in the number of multicast members for both protocols. Although with an increasing number of multicast group members there was a slight decrease in the number of transmitted data messages for NB-TRACE, shown in Fig. 5.1, the energy consumption increases. This is due to the fact that the energy consumption in the reception and transmission states of a node are of the same order. As the number of multicast group members increases, the number of receptions increase, which in turn increases the energy consumption.

It can be observed from Fig. 5.2 that for small multicast group sizes MC-TRACE is more energy efficient while for large group sizes NB-TRACE performs a more energy efficient operation. NB-TRACE consumes 13% more energy compared to MC-TRACE when there is only one group member. On the other hand, when all 99 nodes are in the multicast group, the energy consumption of NB-TRACE is 21% lower than the energy consumption of MC-TRACE.

The energy consumption of MC-TRACE increases faster than the increase in NB-TRACE and goes above the energy consumption of NB-TRACE at the cross-over point of 11 multicast group members. The cross-over point in Fig. 5.2 is lower than the cross-over point observed in Fig. 5.1. This is expected since the

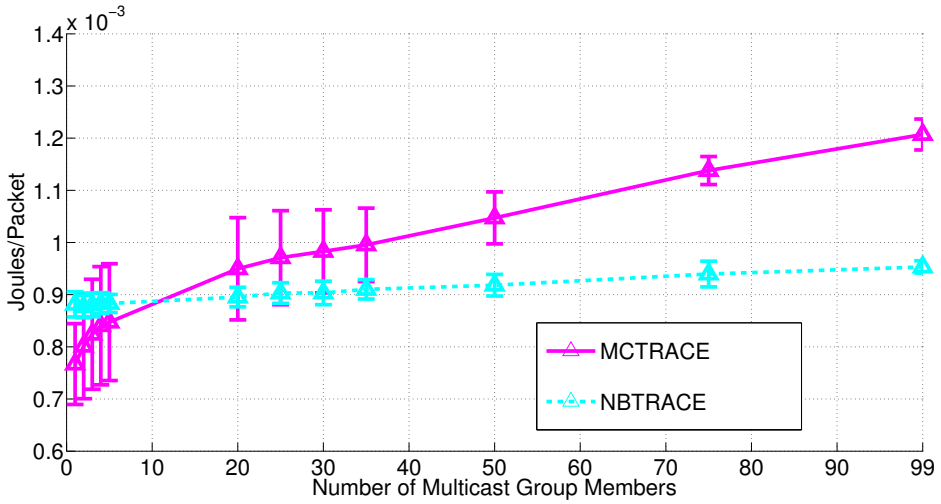


Figure 5.2: Energy consumption per generated packet per node as the number of group members is varied from 1 to 99 for a network of 100 nodes distributed on a 1×1 km^2 area.

length of the IS slots in MC-TRACE is higher than the length of the IS slots in NB-TRACE, which increases the energy consumption of MC-TRACE compared to NB-TRACE.

5.3 Effect of Node Density

In Section 5.2 we observed that, depending on the number of multicast group members, NB-TRACE and MC-TRACE can be advantageous over the other one in terms of bandwidth and energy efficiency. We have identified the cross-over points for a network consisting of 100 nodes distributed on a 1×1 km^2 area. In this section, we extend our analysis to other scenarios to see the effects of node density on the efficiency of the protocols.

In considering the density, both the number of nodes and the size of the area in which the nodes are distributed are of concern. The nodes close to the edge of the target area have fewer neighbors and data dissemination close to the edges follows

a pattern different than the data dissemination pattern of the nodes residing on the middle of the region. In order to investigate the edge effects accurately, variation in the number of nodes and the size of the area are considered separately.

Specifically, we investigate the cross-over points on the number of multicast group members above which NB-TRACE is more efficient compared to MC-TRACE and vice versa in terms of the number of data packet transmissions and energy consumption for a range of total number of nodes and area sizes. The observed cross-over points are presented in Table 5.1.

The first observation from Table 5.1 is that for a given number of nodes, the cross-over point for the number of data transmissions increases with an increase in the size of the area. For larger areas, the separation between the source and the multicast members is larger. This translates into a larger number of hops between the source and a destination, and hence a higher number of transmissions in both NB-TRACE and MC-TRACE. However, the increase in the number of transmissions in NB-TRACE is larger than it is in MC-TRACE since the number of data transmissions in NB-TRACE is effected not only by an increase in the expected distance between the source and the multicast members but also by an increase in the expected separation between the source and non-multicast members. As a result, the value of the cross-over point for the number of data transmissions metric increases with an increasing size of the network.

Table 5.1 also shows that, for a fixed network size, the network with more nodes has a higher cross-over point. Since the node locations are independent, the separation between the source and the multicast members is independent of the number of nodes in the network. Hence, the number of transmissions in MC-TRACE does not deviate significantly as the total number of nodes increases. On the other hand, as the total number of nodes in the network increases, they are expected to cover a larger region on the simulation area, and the number of CHs is expected to increase. As a result of this, the number of transmissions

Table 5.1: Cross-over points between MC-TRACE and NB-TRACE

Sim Area	**1000x1000**		**1500x1500**		**2000x2000**	
Number of Nodes	**100**	**200**	**200**	**400**	**200**	**400**
Number of Data TX	24	30	45	55	55	62
Energy Consumption	11	18	19	29	18	30

in NB-TRACE increases up to a limit where the entire area is covered as the total number of nodes increases. Thus, the cross-over point also increases for an increasing number of nodes.

The energy consumption of a node consists of a variable part that is incurred for data packet transmissions and receptions and a relatively constant part for the control messages. For a fixed network size and a fixed number of multicast members, the number of nodes that take part in the data dissemination tree created in MC-TRACE is independent of the total number of nodes in the network. Hence the variable energy consumption for data dissemination is also independent of the number of nodes. When the number of nodes in the network increases, the energy consumption per node in MC-TRACE decreases. On the other hand, for a fixed network size and a fixed multicast group size, the energy consumption in NB-TRACE is independent of the number of nodes in the network since the data dissemination tree covers the entire network. As a result, the cross-over point increases with increasing number of nodes in the network, as can be observed from Table 5.1.

It is interesting to note that, when considering energy-efficiency, the cross-over point is approximately constant for a fixed number of nodes as the network area increases. This is because the energy consumption of both NB-TRACE and MC-TRACE increases as the size of the area increases, mainly due to the increase in the number of clusters and in turn the additional control messages. Hence the

increase in the average energy consumption is of the same order in NB-TRACE and MC-TRACE and does not alter the value of the cross-over point significantly for a fixed number of nodes.

To sum up, for the goal of minimizing the number of transmissions, both an increase in the number of nodes and an increase in the size of the network favor multicasting over network-wide broadcasting, making multicasting the optimal choice for a larger set of multicast members. Similarly, considering the goal of minimizing the energy consumption per node, an increase in the number of nodes in the network makes multicasting a better choice up to a larger number of multicast members. On the other hand, the relative efficiency of multicasting and network-wide broadcasting in energy consumption is independent of the size of the network.

5.4 Summary

In this chapter, we examined the effect of the number of multicast members on the relative efficiency of multicasting and broadcasting. We consider two performance metrics: energy efficiency and bandwidth spectrum efficiency. We showed that for large multicast groups, using broadcasting instead of multicasting leads to 75% savings in the number of transmitted data packets and up to 21% savings in the average energy consumption. Similarly, for small multicast groups, multicasting reduces the number of data transmissions and the average energy consumption by 50% and 13%, respectively.

We also showed that an increase in the total number of nodes in the region decreases the relative efficiency of broadcasting compared to multicasting for both performance metrics, and hence the cross-over occurs at a larger number of multicast members. On the other hand, the increase in the size of the area does not affect the cross-over point significantly for the energy efficiency metric, while it

increases the cross-over point for the bandwidth efficiency metric.

Due to the similarity between the protocols, NB-TRACE and MC-TRACE, they can be combined into a new unified protocol, U-TRACE. Based on the a-priori information about the number of nodes and the size of the network, the source node can choose the appropriate type of operation considering the number of nodes in the multicasting group.

U-TRACE is a complete suite that incorporates the energy saving mechanisms of MH-TRACE discussed in Chapter 3, dynamic channel allocation capabilities of DCA-TRACE from Chapter 4, as well as one-to-many communication services that exist in NB-TRACE and MC-TRACE. Combining the protocols in TRACE and enhancing the resource utilization in them, U-TRACE becomes the most advanced protocol in the TRACE family.

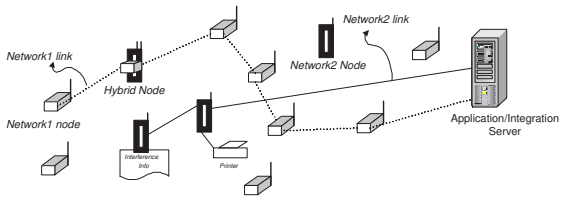
An internal optimization on resource utilization may not be sufficient for a protocol since the performance may also depend on the interactions of the network with other co-located networks. In the next chapter, we focus our attention on the interactions of MANETs with other co-located networks.

6 Network Symbiosis on Hybrid Nodes

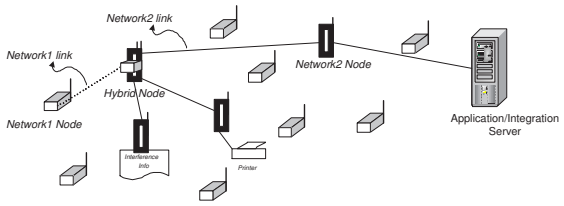
6.1 Introduction

Although being the most advanced in its family, the U-TRACE protocol presented in Chapter 5 has its own goal of energy efficient real time data communication in MANETs. Other networks with different purposes and applications may use different protocols. The interactions of a network with other networks in its environment have a significant impact on the resource utilization of a network. In this chapter, we study the cooperation of networks to optimize individual network performances by cross-network sharing of resources, information and services.

Nodes that support multiple networks are common and referred to as hybrid or multi-mode nodes. Such nodes are vital to achieve network symbiosis, i.e., the full cooperation of the networks. Currently, even if a relay node is a multi-mode node, it cannot optimally select the network to use for a received packet, e.g., a packet that originated in a MANET cannot be sent through a ZigBee network, due to the independent operation of the individual networks. Instead, networks can exchange information through applications running on application or integration servers as in [107]. Alternatively, the integration of different networks can be achieved at any network level by *symbiotic networking*, which enables the sharing



(a) Independent Networking



(b) Symbiotic Networking

Figure 6.1: Comparing independent and symbiotic networking in the presence of multi-mode (hybrid) nodes.

of network components for more efficient data communication or data processing.

Enabling such sharing through local cross-network interactions achieves much more efficient network operation in terms of bandwidth, delay and other important metrics such as energy consumption. Fig. 6.1 presents an illustrative scenario for the comparison of the two networking types, independent networking and symbiotic networking. In this scenario, two co-located networks are given where paths from a network-1 node to different destinations are illustrated. As seen in the figure, symbiotic networking enables routing paths with fewer hops by relaying the packet over different networks, and it enables the sharing of local information and resources by the co-located networks.

In this chapter, we investigate the potential gains achieved by symbiotic networking, concentrating on the routing performance. The contributions of the work presented are threefold. First, the advantages of symbiotic networks are presented quantitatively to motivate further study on symbiotic networks. Routing performance limits of symbiotic networks are also investigated for three different link

cost types and minimum cost routing. The results are compared with individual network performance limits where the routing decisions are done using only the individual network resources. Finally, a mathematical framework is derived for routing performance limits with and without network symbiosis that allows investigation of different network parameters on the routing performance.

The remainder of this chapter is organized as follows. Routing benefits of network symbiosis is investigated through simulations in Section 6.2. Mathematical models for the evaluation of routing costs for each networking type are presented and validated in Section 6.3. Section 6.4 discusses the effects of network parameters on network symbiosis performance gains, and Section 6.5 presents a cost analysis for the cost range of the hybrid nodes where symbiotic networking is beneficial. Finally, Section 6.6 concludes the chapter with an overall summary.

6.2 Routing Benefits of Symbiotic Networking

Routing paths of a network are chosen based on the objectives of the application and the network topology. Network symbiosis provides alternative paths for the communication of two nodes and hence may provide better routes than the ones that exist when only the nodes of an individual network are considered. In this section, we investigate the possible benefits of using network symbiosis in co-located networks.

We begin with a quantitative study of the benefits of network symbiosis via simulations. Specifically, we consider two co-located networks with different network parameter settings. The minimum cost paths are investigated for a range of link cost metrics for both independent and symbiotic networking, since minimum cost paths present a limit for routing protocol performances and can be used for

comparison of both networking types. Moreover, the minimum cost path can be found using algorithms such as Distributed Bellman-Ford [108] in a distributed environment.

Let network-i represent the network, where $i = 1, 2$. In network-i, N_i nodes are uniformly randomly distributed in a target area. This uniformly random distribution can correspond to a static random topology as in wireless sensor networks or an instance of dynamic topology of as in mobile ad hoc networks. Additionally, there are H hybrid nodes in the same target area, with network interfaces to both networks, allowing them to communicate with the nodes of both networks and to other hybrid nodes. In the independent network scenarios, hybrid nodes are only capable of forwarding packets using the interface on which they come, i.e., a network cannot relay the other network's packets. On the other hand, for the symbiotic network scenarios, switching interfaces on hybrid nodes is allowed.

Nodes of network-i are assumed to be capable of communicating in a loss-free manner with any nodes of network-i or with hybrid nodes within a fixed communication radius, r_i. A sample deployment is shown in Fig. 6.2 with the network parameter settings: N_1=80 nodes, r_1=1 unit, N_2=30 nodes, r_2=2 units, and H=5 nodes. A common destination node, e.g., sink, application server or internet access point, which can communicate with both node types, is located in the center of the network area. Hence, the destination node is also a hybrid node.

6.2.1 Evaluation of Minimum Cost Paths

In this section, we investigate a scenario of two co-located networks, where N_1=250 nodes, N_2=150 nodes, r_1=1 units, and r_2 = 2 units, distributed on a 8x8 units^2 region. The number of hybrid nodes (excluding the sink node) is varied from 0 to 200. For each parameter set, 100 iterations are performed with random

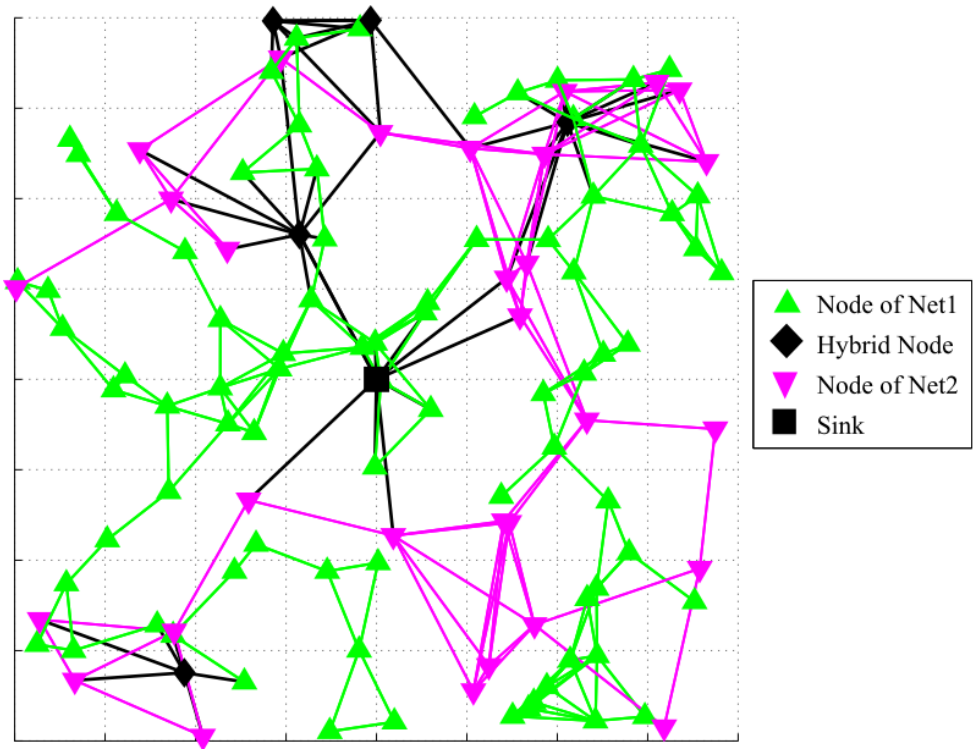


Figure 6.2: Two co-existing networks with 5 hybrid nodes.

deployments. The averages of the 100 runs are shown in the figures along with the 95% confidence intervals for the population mean. To not have nodes with infinite costs, each network is redeployed until a connected topology is achieved. The cases of disconnected networks are studied in Section 6.2.2.

We consider two possibilities for routing from the individual nodes to the sink: 1) routing in symbiotic networks, where data can travel from nodes in network-i to nodes in network-j; 2) routing in independent networks, where data that originates in network-i must remain in network-i.

For both types of routing schemes, we consider three link cost metrics. Let c_{kl} represent the link cost of the link between node k and node l. Then, the metrics and specific link cost values considered are

1. constant link cost ($c_{kl} = 1$),
2. network specific link cost ($c_{kl} = r_i^2$),
3. distance squared link cost ($c_{kl} = d_{kl}^2$, where d_{kl} is the distance between nodes k and l).

Minimum cost paths for constant link cost result in the shortest hop distance between the source and the destination. Especially in networks where the data load is low, there is a strong correlation between hop count and the delay per packet. Thus, constant link cost metric corresponds to minimizing the packet delays.

The second cost metric assumes different constants for each network's link cost. Specifically, we choose the cost of links in each network proportional to the square of the communication radius. Assuming fixed transmission power, the transmission energy spent by each node in network-i is proportional to the square of the communication radius, r_i, according to free space path loss propagation

model. Hence, the second cost metric aims to minimize the power consumption of the entire network.

There are protocols that allow nodes to adjust their transmission power so as to reach the intended receiver. Assuming perfect power adjustment, the power spent on each link is proportional to the square of the distance between the transmitter-receiver pair, which is the third link cost metric that we investigate.

For each network topology, the minimum routing path costs of each node-sink pair is calculated using the shortest path algorithm[1]. The minimum node-sink routing path costs are averaged over all nodes and over all different deployments for each parameter set. The averages and the corresponding 95% confidence intervals for the three cost metrics are shown in Fig. 6.3 to 6.5 for varying number of hybrid nodes. As the number of hybrid nodes in a network increases, the expected number of candidate relays at each hop increases and as a result, the average cost of the shortest path decreases for all three cost metrics. However, symbiotic routing enables more alternative relays for hybrid nodes preserving the ones that already exist for independent routing. The routing cost values using symbiotic routing are, therefore, lower than (or equal to) the routing cost values using independent routing for both networks and for all three cost metrics as seen in the figures.

The minimum cost paths for constant link cost metric corresponds to the minimum hop count paths. Network-2 has a larger communication radius and hence has much lower average hop count compared to network-1 for all node densities for the independent routing case, as can be observed in Fig. 6.3. For the same reason, network symbiosis improves minimum hop counts of network-1 nodes by allowing them to benefit from the links of network-2. Thus, even for a low number of hybrid nodes, the average minimum hop count of network-1 decreases

[1]In this context, shortest path refers to the path with minimum cost. Correspondingly, shortest path algorithm finds the path with minimum cost among all possible paths between two nodes.

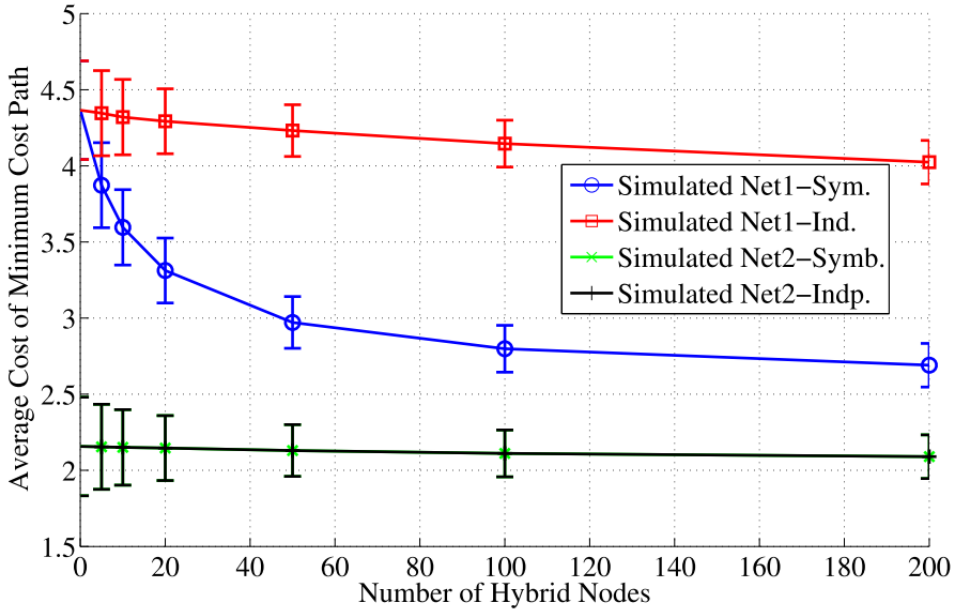


Figure 6.3: Average costs for the constant link cost metric for varying number of hybrid nodes.

significantly. The addition of 20 hybrid nodes in the area decreases the average minimum hop count of network-1 by 24% when using symbiotic routing, while the decrease when using independent routing is only 1.6%. On the other hand, for the minimum hop count metric, network-2 has no incentive to utilize network-1 for transporting its packets. Thus, there is no significant difference between the average cost of the minimum cost path for network-2 for independent routing and for symbiotic routing.

For the network specific link cost metric, the distance per unit cost $(\frac{r_i}{r_i^2})$ of network-1 and network-2 links are 1 and 0.5, respectively. Hence, network-1 links are potentially advantageous over network-2's links, and network-1 has a lower average path cost compared to network-2 for independent routing as shown in Fig. 6.4. Due to the same reason, network symbiosis helps network-2 by allowing it to use network-1's links while it does not help network-1 significantly. The addition

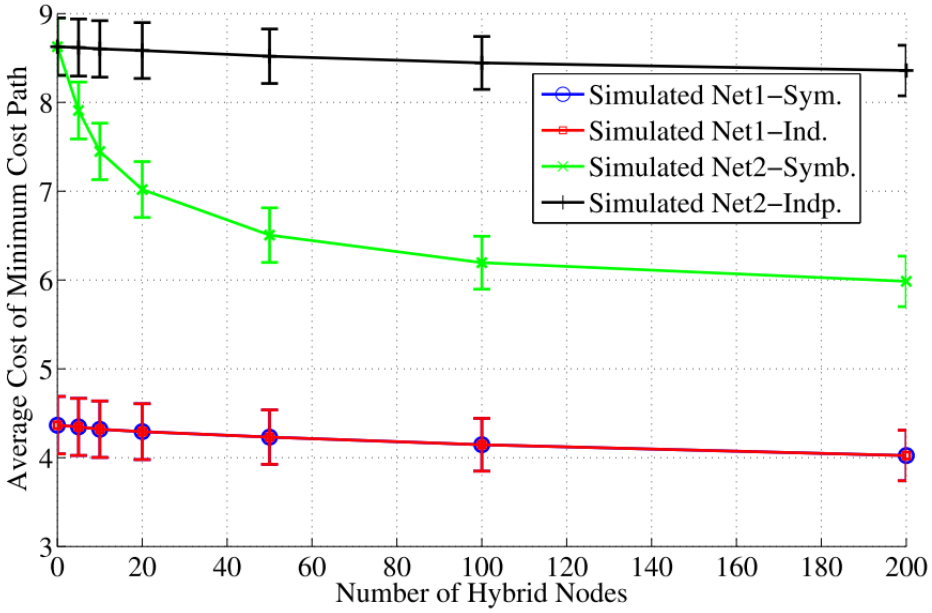


Figure 6.4: Average costs for the network specific cost metric for varying number of hybrid nodes.

of 20 hybrid nodes in the area decreases the cost of network-2 by 18.6% when symbiotic routing is used, whereas the decrease is only 0.5% when independent routing is used.

For both the constant link cost and the network specific link cost metrics, the cost of a link is independent of the distance of the node pair of that link. Thus, it is beneficial to choose the neighbor that makes the maximum possible advancement towards the sink in both of these cases. On the other hand, with the distance squared link cost, the cost of each link also depends on the node distances. Fig. 6.5 shows the average minimum path costs for distance squared link costs. Since the link cost values grow faster than the distance, and hence faster than the advancement towards the sink, paths with shorter links have lower costs. Having a short link is related with the availability of a close neighbor and is independent of the communication radius. Since, network-1 has a greater number

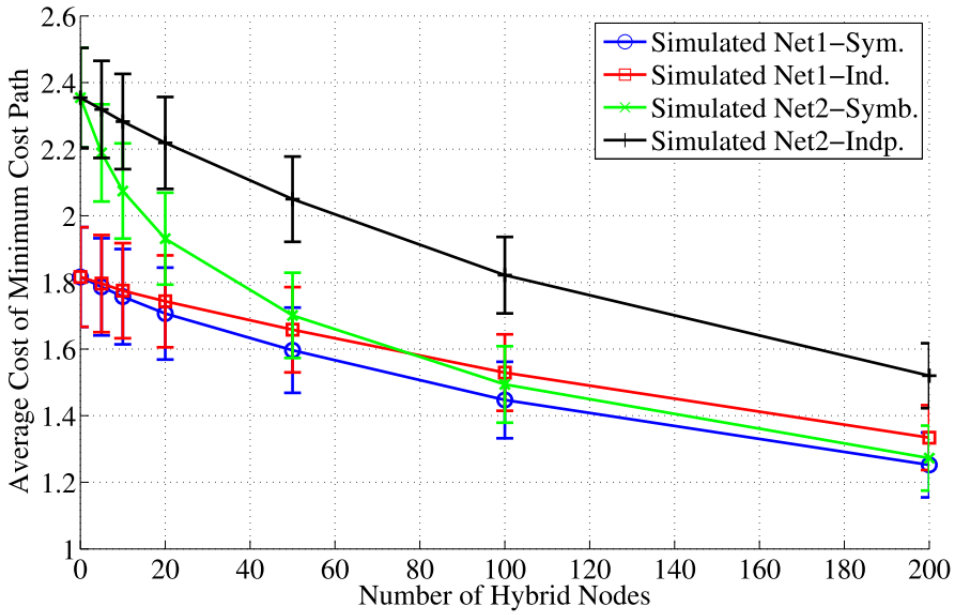


Figure 6.5: Average costs for the distance squared cost metric for varying number of hybrid nodes.

of nodes, it has lower average path costs, as can be observed in Fig. 6.5. Symbiotic routing allows utilization of the nodes of both networks for hybrid nodes. Thus, it increases the probability of having a path with shorter links and decreases the cost of the minimum cost paths of both networks.

To sum up, for all the three cost metrics, symbiotic routing can reduce the routing costs significantly compared to the independent routing. For a given scenario, symbiotic routing can improve the delay of one network's packets while reducing the other network's total energy consumption. As a result, both energy-efficient and delay-sensitive applications can be run on any of the two networks. Thus, symbiotic routing provides flexibility for the range of applications that could be run on a network and allows applications that require conflicting objectives to co-exist.

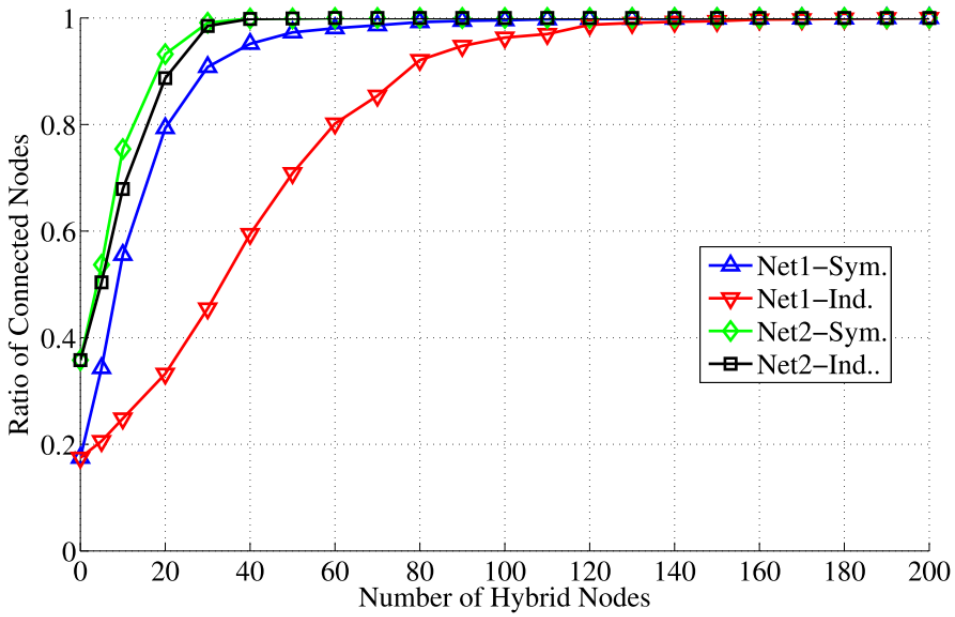


Figure 6.6: Ratio of Number of connected nodes to the number of nodes in each network for symbiotic routing and independent routing

6.2.2 Evaluation of Connectivity

Network symbiosis also plays a drastic role in increasing the connectivity for sparse networks. To illustrate this advantage, a sample scenario with two sparse networks with parameters $N_1 = 50$, $N_2 = 10$ nodes, $r_1 = 1$, and $r_2 = 2$ units, on an area of 8x8 units are evaluated. We investigate the number of connected nodes, i.e., the nodes that has a path to the sink, as number of hybrid nodes are varied from 0 to 200. The ratio of the number of connected nodes to the total number of nodes for both symbiotic routing and independent routing are shown in Fig. 6.6.

The increase in the number of hybrid nodes corresponds to an increase in the average number of neighbors in each network and thus the connectivity ratio increases for both independent routing and symbiotic routing. For symbiotic routing, however, the nodes of one network can be used in connecting the nodes

of the other network provided that there is at least one hybrid node within the communication range. Thus, the connectivity ratio using symbiotic routing is always greater than or equal to the connectivity ratio using independent routing for both networks. With independent routing, the ratio of connected nodes in network-2 is much higher compared to network-1. Since network symbiosis allows one network to use other's links, the gain achieved by network symbiosis is much greater for network-1.

As can be observed in Fig. 6.6, addition of small number of hybrid nodes increases the connectivity ratio of network-1 drastically in symbiotic routing compared to that of independent routing. Using network symbiosis, an addition of 10 hybrid nodes increases the number of connected nodes in network-1 by 38%, whereas the increase is only 7% using independent routing.

6.3 Mathematical Model of Routing Performance

In this section, we mathematically analyze the minimum cost path using independent and symbiotic routing. In particular, we develop a mathematical model that approximates the average cost of the minimum cost path for both constant link cost and network specific link cost metrics. We also present the lower bound on the cost of the minimum cost path that is approached when the node density goes to infinity. Then, the mathematical model is validated by comparing its results with the simulation results and the lower bounds.

6.3.1 Mathematical Model

We begin the analysis with constant link costs for independent routing. A least remaining distance (LRD) forwarding approach [109] [110] can be considered as

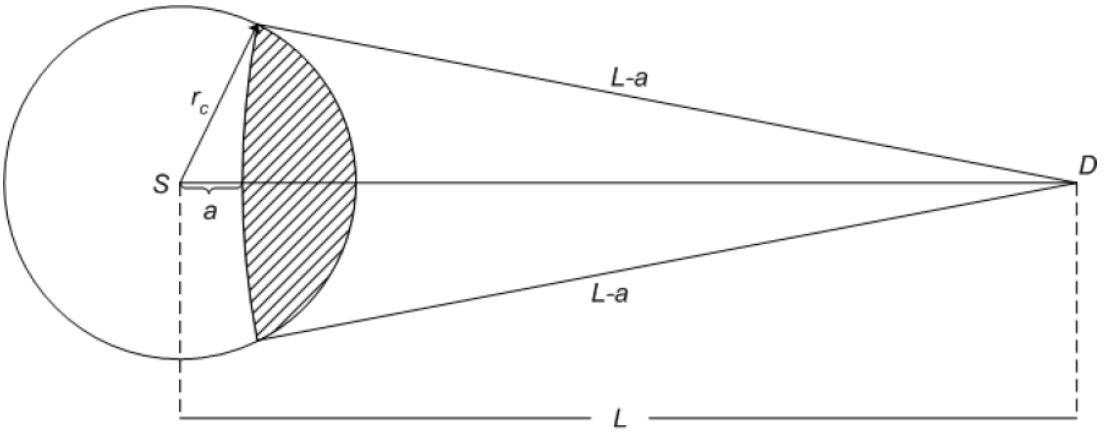


Figure 6.7: Graphical representation of advancement to destination.

an approximation for the shortest path for sufficiently dense networks. In LRD, the next hop is chosen such that the remaining distance to the destination is minimized. For the case given in Fig. 6.7, consider the neighbors of source node S, i.e., the nodes within the communication range r_c of S. Since, the node locations are statistically independent, the expected cost of the shortest path from any node at a given distance to the sink is equal. Thus, among the neighbors of S, the ones residing on the circle with radius $r = L - a$ and centered at the destination node D result in an advancement of a towards D [110]. Given that the cost of each link is constant, it is beneficial to choose the node with the maximum advancement as the next hop.

Let ζ represent the maximum possible advancement that can be achieved towards D from S in a single hop. For the trivial case of $L \leq r_c$, $Pr(\zeta = L) = 1$ and the expected value of the maximum advancement is

$$E[\zeta] = L. \tag{6.1}$$

For $L > r_c$, $Pr(\zeta \leq a)$ is given by the probability of having no nodes in the shaded area in Fig. 6.7. Assuming node locations are uniformly random and independent,

and there are N total nodes in the network,

$$Pr(\zeta \leq a) = \left(1 - \frac{A(L, r_c, L-a)}{TotalArea}\right)^N, \tag{6.2}$$

where $A(L, r_1, r_2)$ is the area of the intersection region of the two disks with radii r_1 and r_2, whose centers are separated by L. The formula of $A(L, r_1, r_2)$ can be found in [111]. The expected value of the maximum advancement for a given distance $L > r_c$ can be written as

$$\begin{aligned} E[\zeta(L, r_c, N)] = & \int_{a=0}^{a=r_c} a \frac{dPr(\zeta \leq a)}{da} da \\ = & (aPr(\zeta \leq a))\Big|_{a=0}^{a=r_c} \\ & - \int_{a=0}^{a=r_c} Pr(\zeta \leq a) da. \end{aligned} \tag{6.3}$$

The expected advancement at the next hop is a function of remaining distance to the destination, which is defined by the advancement at the current hop. Therefore, beginning from the source, an iterative algorithm can approximate the expected number of hops for a given distance L between source and destination, which is given in Algorithm 1.

Algorithm 1 Find the expected number of hops for network-i for independent routing with constant link cost metric

Require: L, r_i, N_i, H, $TotalArea$

$C_T = 0$

while L > 0 **do**

  Calculate $E[\zeta(L, r_i, N_i + H)]$ using (6.1), (6.2), and (6.3)

  L = L - $E[\zeta(L, r_i, N_i + H)]$

  $C_T = C_T + 1$

end while

Next, we extend this analysis to the case of symbiotic routing. Let the probability of being on a network-i node be p_i at a hop and the probability of being on a hybrid node be p_h, where $\sum_{i=1,2}(p_i) + p_h = 1$.

For two networks case, one of the networks' links are preferable since they provide greater advancement per unit cost. We refer this advantageous network as superior network, and the other network as inferior network[2]. For the hop count metric, network-j is superior if $E[\zeta(L, r_i, N_i + H)] < E[\zeta(L, r_j, N_j + H)]$, since the cost is 1. In that case, the tendency of the routing algorithm will be toward switching to network-2.

The forwarding algorithm is assumed to hop from the nodes of the inferior network to the hybrid nodes whenever there is a hybrid node that makes positive advancement. Thus, the probability of switching from the inferior (i.e., conditioned on being on an inferior node) to the superior network can be calculated as

$$p_{sw} = 1 - \left(1 - \frac{A(L, r_{inf}, L)}{TotalArea}\right)^H . \tag{6.4}$$

At each hop, the expected advancement toward the destination can have 3 different values corresponding to 3 different cases: the expected advancement from a node of the inferior network to a node of the inferior network, $E[\zeta_{inf}]$; from a node of the inferior network to a hybrid node, $E[\zeta_{sw}]$; from a node of the superior network or hybrid node to a hybrid node or to a superior network, $E[\zeta_{sup}]$. The probabilities of the three cases are $p_{inf}\,(1 - p_{sw})$, $p_{inf}\,p_{sw}$, $(1 - p_{inf})$ where inf and sup corresponds to the indices of inferior and superior networks, respectively. The expected value of the maximum advancement is the weighted average of $E[\zeta_{inf}]$, $E[\zeta_{sw}]$ and, $E[\zeta_{sup}]$, where the weights are the probabilities of the corresponding cases.

An iterative algorithm to calculate the expected minimum number of hops for

[2]For networks with significant difference in node densities, the definition of inferior and superior networks may not be fixed. The required minimum difference between the node densities of the two networks increase with increasing difference in the communication radii of the networks. Our evaluations show that for $TotalArea = 100$, r_i=1, $r_j = 2$, $N_i = 1000$, in order to have a change in between the superior and inferior networks, N_j should be less than 13.

a given distance L using symbiotic routing is provided in Algorithm 2.

Algorithm 2 Find the expected number of hops for network-i for symbiotic routing with constant cost metric

Require: L, r_i, r_j, N_i, N_j, H, $TotalArea$

1: $C_T = 0, p_i = 1, p_j = 0, p_h = 0$
2: **while** L > 0 **do**
3: **if** $E[\zeta(L, r_i, N_i + H)] < E[\zeta(L, r_j, N_j + H)]$ **then**
4: $inf \leftarrow i, sup \leftarrow j$
5: **else**
6: $inf \leftarrow j, sup \leftarrow i$
7: **end if**
8: $E[\zeta_{inf}] = E[\zeta(L, r_{inf}, N_{inf})]$
9: $E[\zeta_{sw}] = E[\zeta(L, r_{inf}, H)]$
10: $E[\zeta_{sup}] = E[\zeta(L, r_{sup}, N_{sup} + H)]$
11: Calculate p_{sw} using (6.4)
12: $L = L - E[\zeta_{inf}]\, p_{inf}\, (1 - p_{sw}) - E[\zeta_{sw}]\, p_{inf}\, p_{sw} - E[\zeta_{sup}]\, (1 - p_{inf})$
13: $p_{inf} = p_{inf}\, (1 - p_{sw})$
14: $p_h = p_{inf}\, p_{sw}$
15: $p_{sup} = 1 - p_{inf} - p_h$
16: $C_T = C_T + 1$
17: **end while**

Averaging results of Algorithms 1 and 2 numerically for all possible points on the target area yields an approximation to the average minimum hop count between a node and the sink for independent and symbiotic routing, respectively.

The same algorithms can be used with minor modifications also for the network specific link cost metric which considers different link cost constants for each network [3]. The only modification in Algorithm 1 is to increment C_T by C_i (the cost of the links in network-i) instead of by 1 in Line 5. Similarly, Line 16 of Algorithm 2 has to be $C_T = C_T + p_{inf}\ C_{inf} + p_{sup}\ C_{sup}$. Also, the inferior and superior networks have to be chosen for symbiotic routing with respect to the

[3]These algorithms are not applicable for distance based link cost metric since the node with maximum advancement may not result in minimum cost path in that case. The extension of the analysis for distance squared cost metric is left as a future work.

expected advancement per unit cost incurred. Thus, the condition to be checked in Line 3 of Algorithm 2 has to be $\frac{E[\zeta(L,r_i,N_i+H)]}{C_i} < \frac{E[\zeta(L,r_j,N_j+H)]}{C_j}$.

For comparison purposes, we present the lower bounds that can be achieved with infinite node density, where the probability of having a neighbor that makes an advancement of r_i goes to 1. Therefore, a lower bound for the cost of shortest path to send data to a distance L with independent network routing, $LB_i^{Ind}(L)$, can be found as

$$LB_i^{Ind}(L) = \left\lceil \frac{L}{r_i} \right\rceil C_i. \tag{6.5}$$

For the case of symbiotic networking, the first hop has to use the links of the network which source node belongs to. The remaining hops use the superior network's links designated

$$LB_i^{Symb}(L) = C_i + \left\lceil \frac{L - r_i}{r_{sup}} \right\rceil C_{sup}, \tag{6.6}$$

where *sup* corresponds to the index of the superior network.

For a given network, the lower bound for the average cost of a shortest path can be found by averaging lower bounds for all source-sink pairs. These lower bounds are only approached for very large numbers of nodes in the networks and are practically very hard to reach, as observed in the following section.

6.3.2 Validation of the Model

In this section, the results of the mathematical model for a sample scenario are compared to their simulated counterparts. For each deployment, Algorithms 1 and 2 are used to calculate the expected cost of the shortest path of each node. The same scenario that is evaluated in Section 6.2.1 ($N_1 = 250, N_2 = 150, r_1 = 1, r_2 = 2$, Area = 8x8) is used for validation of the model.

The average minimum hop counts between the nodes and the sink are plotted with respect to the number of hybrid nodes in Fig. 6.8. Mathematical model

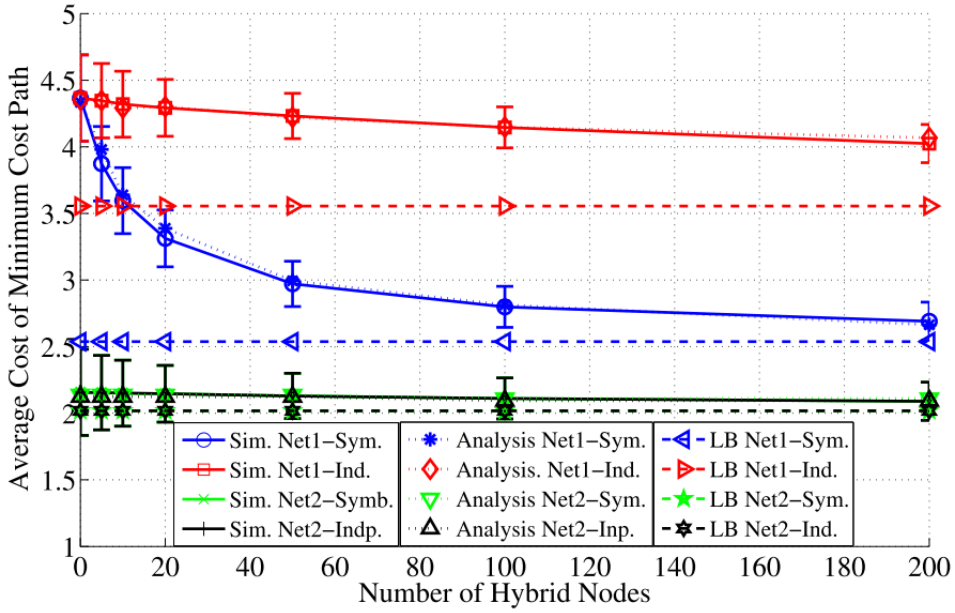


Figure 6.8: Average cost of the shortest path using constant link costs

results follow the simulation counterparts closely for both networks and for all data points. All the mathematical model results are within the 95% confidence intervals, validating the mathematical model presented in Section 6.3.1. Moreover, as seen in Fig. 6.8, symbiotic routing can achieve much lower hop counts than the lower bound of independent routing for network-1, which only can be achieved with infinite node density.

For the validation of analyses of average path costs for network specific fixed linked costs, Algorithms 1 and 2 are used with link costs of r_i^2. Simulation results are shown along with the results found by the analyses in Fig. 6.9. As seen in the figure, the mathematical models presented in Section 6.3.1 approximate the simulation results successfully for the network specific link cost metric as well.

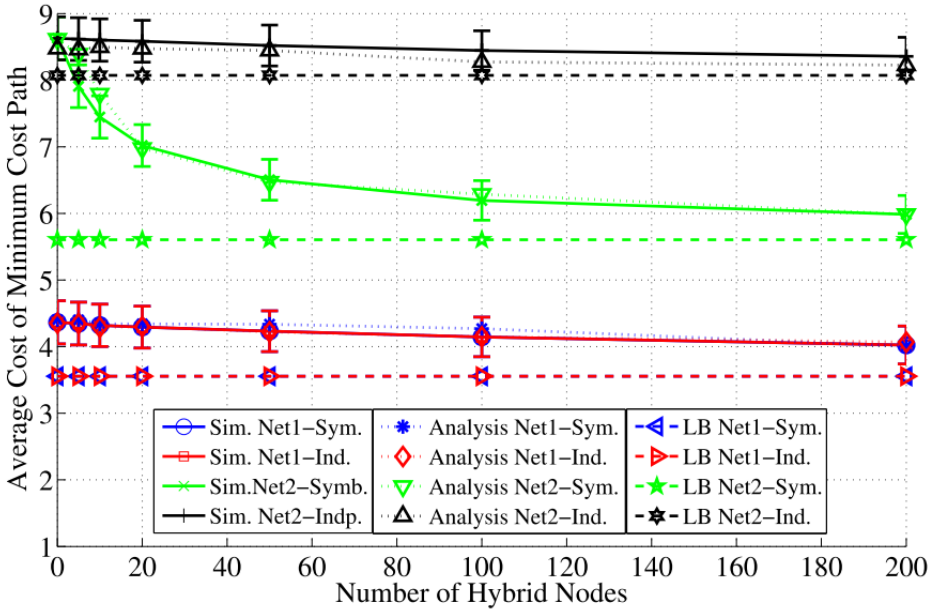


Figure 6.9: Average cost of the shortest path using network specific link costs

6.4 Effects of Network Parameters

In Sections 6.2 and 6.3, it is observed that for a given link cost metric, the cost improvement in one network achieved by the network symbiosis is much higher than the one in the other network. The parameters of the networks under concern determine the magnitude of the improvement. In this section, using the mathematical model developed, we investigate the variation in the cost improvements achieved by network symbiosis as the network parameters vary.

Specifically, we vary the settings of network-2 (i.e., r_2 and N_2) together with H, and observe the changes in the ratio of the average path cost of symbiotic routing to that of independent routing. The parameter settings of Section 6.2.1 are used as the default values (N_1=250, N_2=150, r_1=1, r_2=2).

The ratio of the average path costs using the constant link cost metric is presented in Fig. 6.10(a) for a range of r_2 and H values. Network symbiosis reduces the cost of network-1 when $r_2 > r_1 = 1$ and vice versa. As r_2 is increased

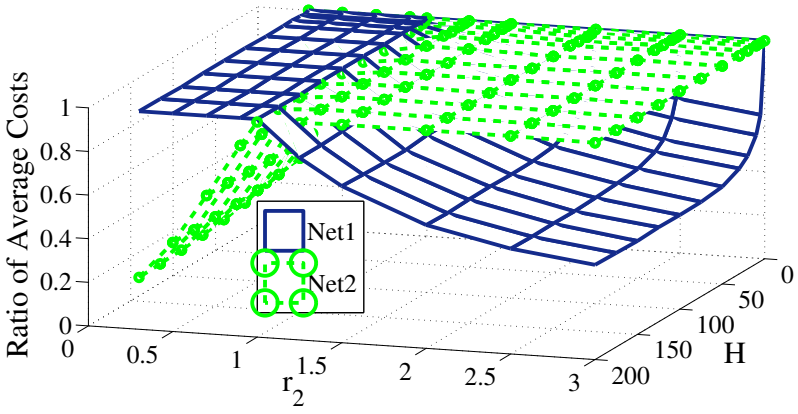
from 1 to 3, the expected advancement per hop on network-2's links increases and thus the path costs of network-1 with symbiotic routing decreases.

The ratio of costs for the network specific link cost metric is presented in Fig. 6.10(b) for different r_2 and H values. Network symbiosis reduces the path costs of network-2 when $r_2 > r_1$ and vice versa. As r_2 is increased from 1 to 3, the expected cost of the network-2's links increases faster than the increase in advancement per hop, since the cost of each network is taken as the square of the communication radius. Thus, using network-1 links becomes more advantageous for network-2 and the gain achieved by symbiosis increases.

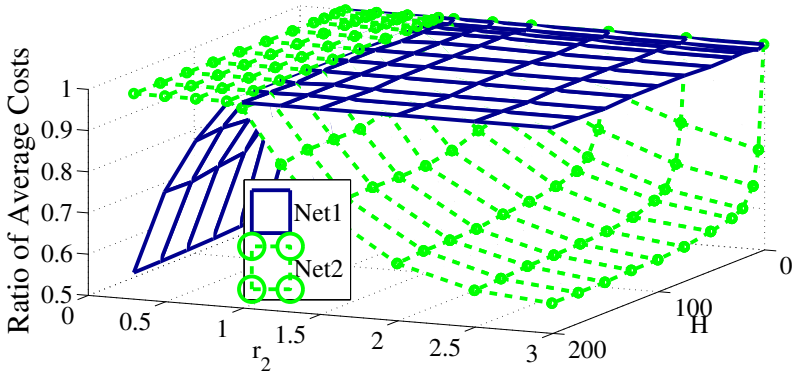
Next, we fix r_2 at 2 and vary N_2 from 100 to 400. The variation in the ratio of costs for constant link cost metric and network specific link cost metrics are presented in Fig. 6.11(a) and 6.11(b), respectively. The superior network is determined based on the expected advancement, which depends on both the communication radius and the node density in each network. As N_2 increases, the expected advancement in network-2's links increases and the gain by network symbiosis increases. However, the dependency on the communication radius is much larger since the networks under concern have similar density values. As a result, the effect of the node density on the ratio of costs is much weaker compared to the effect of the communication radius.

6.5 Cost Analysis

In Sections 6.2 and 6.4, we presented the contributions of symbiotic routing on various performance metrics. From an engineering point of view, that contribution can be translated into reduced costs of network deployment and operations. In this section, the network cost analysis is elaborated through a scenario in which

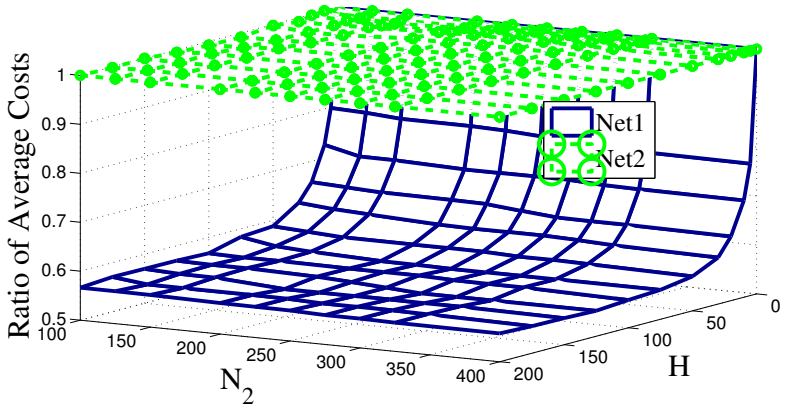


(a) Constant Link Cost Metric

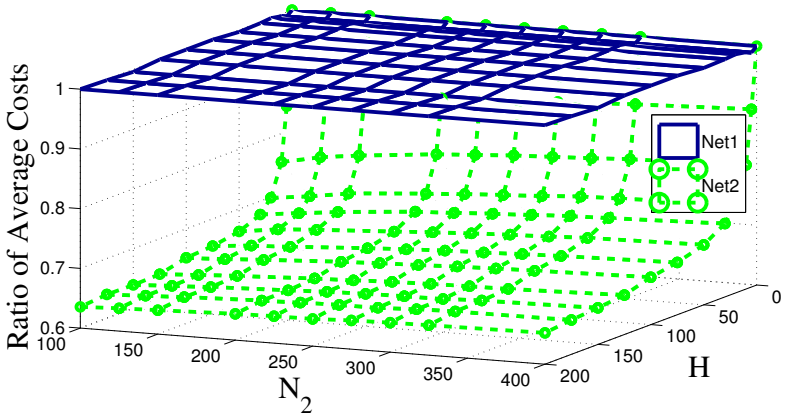


(b) Network Specific Link Cost Metric

Figure 6.10: Ratio of average cost of shortest path for symbiotic routing and independent routing as number of hybrid nodes and communication radius of the second network varies



(a) Constant Link Cost Metric



(b) Network Specific Link Cost Metric

Figure 6.11: Ratio of average cost of shortest path for symbiotic routing and independent routing as number of hybrid nodes and number of nodes in the second network varies

a performance metric of a given system is improved with the additional node deployments.

Specifically, we consider the same scenario of Section 6.2.1 ($N_1 = 250$, N_2=150,r_1=1, r_2=2) and investigate the costs of alternative methods to improve the performance of network-1. Originally, network-1 has an average hop count of 4.35 as can be observed from Fig. 6.3. The hop count value can be reduced in two ways: by increasing the number of nodes in network-1 or by adding hybrid nodes and utilize the network symbiosis.

Using the mathematical model presented in Section 6.3, the average routing cost of the network can be calculated easily for any given number of hybrid nodes. The number of network-1 nodes that has to be added to achieve a specific routing cost value can be calculated using a root-finding algorithm such as bisection method for the mathematical model.

Using that approach, one can find that, in order to reduce the average hop count by 10% (i.e., to 3.92 hops) for independent routing, at least 394 more network-1 nodes should be deployed to the area. On the other hand, network symbiosis makes use of larger communication radius in network-2 and reaches the same target average hop count value by only 8 additional hybrid nodes. Considering that the cost of a network-1 node is c_1 and the cost of a hybrid node is c_h, symbiotic routing achieves the performance requirement with a lower cost as long as $\frac{c_h}{c_1} < \frac{394}{8} = 49.2$.

The cost threshold analysis is extended for a range of percent improvement values in Fig. 6.12. As the required improvement on hop count increases, network symbiosis becomes desirable for an even larger cost of hybrid nodes. For a desired improvement of 18%, using hybrid nodes with network symbiosis is the cost-effective method of achieving this improvement as long as the cost of each hybrid node is less than or equal to 618 times the cost of an individual network node.

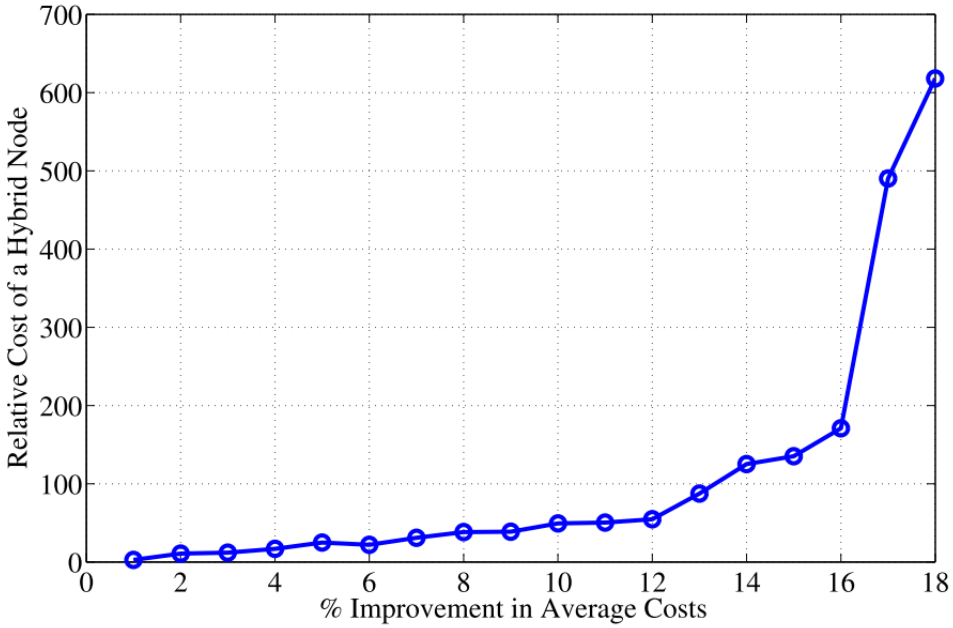


Figure 6.12: The highest cost ratio of hybrid and network-1 node that makes symbiotic routing the cost efficient routing method for a given hop count improvement requirement.

Another important observation is that, using independent routing, the average cost can only be reduced up to 18.3% considering the analytical lower bound of independent routing for network-1. Any improvement beyond that value can only be achieved using network symbiosis irrespective of the relative cost of a hybrid node.

To sum up, the cost of hybrid nodes should be below a certain value for network symbiosis to be a cost efficient method in improving system performance. Using the mathematical model presented, the maximum cost of a hybrid node for the cost efficiency can be found for any given scenario.

6.6 Summary

In this chapter, the advantages of symbiotic networking for co-located wireless networks are presented by simulations and through a mathematical model. To the best of our knowledge, this is the first study to analyze the benefits of symbiotic networking quantitatively. The results are promising: by introducing a small number of hybrid nodes, i.e., a node that can switch packets between networks, the average cost of the shortest path can be decreased significantly. Increasing the total number of nodes by only 5% by adding hybrid nodes into the network decreases the average cost of the shortest path by up to 24%, 18.6%, and 13% for the constant link cost, network specific link cost and distance squared link cost metrics, respectively.

We have also presented a mathematical framework that can be used to investigate the potential gains of symbiotic networking for different network parameters. The model has been shown to follow simulations for a varying number of hybrid nodes and for both the constant link cost and the network specific link cost metrics. The mathematical framework can be used to investigate the effects of different network parameters on the potential gains achieved by symbiotic rout-

ing. It is shown that the addition of a hybrid node, i.e., a node that can switch packets between networks, leads to a greater performance improvement for a low numbers of hybrid nodes.

Finally, a cost analysis is presented for analyzing the benefit of hybrid nodes and symbiotic networking. Even though additional hardware on hybrid nodes increases the production costs, the hybrid nodes make the overall deployment more cost efficient. For the scenario investigated, symbiotic routing is found to be beneficial if the cost ratio of hybrid nodes and network nodes is less than 49 to achieve 10% hop count improvement, which is a realizable threshold for many recent multi-network devices.

7 Implementation on a Software-defined Radio Platform and Practical Issues Encountered

7.1 Introduction

Throughout this thesis we have investigated node cooperation in various parts of communication systems. Through comparative simulation studies, we have shown that the TRACE family of protocols, which implement cooperation in both the MAC and network layers of the protocol stack, perform better in MANETs compared to uncoordinated protocols for heavily loaded networks for both uniform and non-uniform load distributions.

Although simulations are efficient tools to comparatively evaluate the efficiency of protocols, they cannot reflect many of the challenges for real implementation of these protocols, such as clock-drift, synchronization, imperfect physical layers, and interference from devices out of the system. Such issues may cripple a protocol that otherwise performs very well in software simulations. Thus, hardware implementation is essential for testing a protocol before any practical deployment.

In this chapter we use Microsoft Research's software radio (SORA) to implement the TRACE protocol and determine the challenges in implementing this protocol in a real world communication system.

In the original TRACE protocol design, node synchronization was assumed to be provided by external mechanisms such as global positioning system (GPS). We address the issue of synchronization by proposing two synchronization algorithms for TRACE that are based on packet receptions. We show that the algorithms work well in both single and multi-hop networks.

Finally, we address the issue of packet losses due to imperfect physical layers and interference. By adding missing packet compensation mechanisms, we increase the systems resilience against packet losses. We show that the TRACE system maintains its stability even with very low transmission power and cross-band interference.

7.2 Development Platform

The physical (PHY) and the medium access control (MAC) layers of conventional wireless communication systems are typically implemented in Application Specific Integrated Circuits (ASICs) due to their intensive computational requirements. The algorithms controlling these layers are embedded and thus cannot be changed or upgraded. On the other hand, Software Defined Radios (SDRs) implement these functions on special re-programmable hardware that allows flexibility for design changes and hence is more suitable for research. SORA (Software Radio) [112], developed by Microsoft Research Asia in Beijing, is an SDR platform that satisfies the throughput and timing requirements of modern wireless protocols while utilizing the rich general purpose processor development environment. Thus, we choose SORA radios to implement the TRACE protocol.

The SORA system consists of a radio controller board (RCB) that uses a PCI

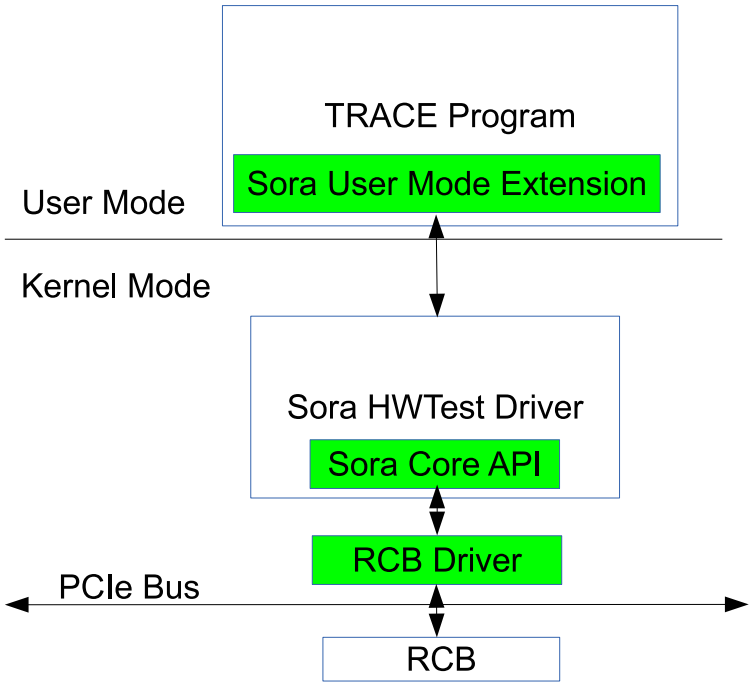


Figure 7.1: The interaction of TRACE software with the SORA architecture.

express slot to communicate with the PC's memory using direct memory access (DMA). The RCB connects to a SORA compatible RF front-end board. We utilize RF2450 type RF front-ends, designed by V3 Technology Ltd. in Beijing, attached to the SORA SDR. These front-ends work on the 2.4GHz band and are compatible with 802.11 PHY implementations.

Both the RCB and the RF front-end hardware resources are managed by the RCB driver that provides API's to send/receive digital waveform samples. SORA systems also include a *Sora User-Mode Extension API* (UMX API) that allows interaction of user-mode programs with the RCB/RF hardware. We choose the UMX Mode for our implementation since it eliminates the need for kernel level programming and greatly reduces the programming efforts. The interaction of the TRACE software with the SORA architecture is depicted in Fig. 7.1.

The processing time of the programs interacting with the radio hardware is of

extreme importance. The real-time behavior of SORA is supported via *exclusive threading*. An exclusive thread (ethread) is a non-interruptible thread running on a dedicated CPU core on a multi-core system. The SORA software development kit includes libraries that provide APIs to create and manage exclusive threads. We use SORA ethreads in the TRACE implementation. Details of the ethreads of the TRACE program are discussed in Section 7.4.

The TRACE protocol family covers MAC and network layer functionalities of the OSI protocol stack. In our TRACE implementation, we use 802.11b as the PHY layer since the UMX extension has a library, *UMXDot11* that implements a simple 802.11 decoder and encoder. At the receiver side, *UMXDot11* implements a simple software power detection algorithm that measures the channel power and returns a value that indicate whether a transmission is detected. Once a transmission is detected, *UMXDot11* goes into reception mode where the 802.11 frames are demodulated. Similarly at the transmitter side, *UMXDot11* extension provides a library that can generate (modulate) and send 802.11 PHY frames. The radio resources are initialized and the transfer buffers are allocated using the "Dot11BRxInit" and "Dot11BTxInit" functions of the *UMXDot11*. Packets are embedded into the 802.11b PLCP layer packet type using "Dot11BPreparePacket" function. The "BB11BPMDPacketGenSignal" function adds a preamble and embeds the PLCP layer packet into a PMD layer packet and modulates the frame with the chosen modulation method. The modulated complex samples are transferred to the RCB's memory using the "SoraURadioTransfer" function. The RCB memory resources are locked using "Dot11AcquireTxBufLock" function before the transfer, and the lock is released using the "Dot11ReleaseTxBufLock" function. These functions provide PHY layer transmitter functionality to the *UMXDot11*. Table 7.1 lists these functions and shows the average duration to complete each function, measured over 10,000 executions.

Table 7.1: PHY layer transmitter functions and average execution durations.

	Duration (μs)
Dot11BRxInit	3366.128
Dot11BTxInit	1336.667
Dot11BPreparePacket	39.333
Dot11AcquireTxBufLock	7.512
BB11BPMDPacketGenSignal	922.021
SoraURadioTransfer	138.521
Dot11ReleaseTxBufLock	8.514

7.3 Modules of the TRACE System

TRACE is implemented as an independent layer in the communication stack, and its operation is decoupled from system dependent functions. The interaction of TRACE with the other layers in the system is handled through adapter modules. Fig. 7.2 depicts the TRACE Manager and its interactions with other layers.

Being a TDMA based protocol, the operation of TRACE is tightly linked to time measurements. The time dependent services in the TRACE MAC Manager are handled by the TRACE Timer module. The TRACE Timer module is responsible for measuring the time using system counters and reporting the time in a format usable by the TRACE system.

The implementation of the TRACE Manager is independent of the underlying PHY layer. The information to be sent over the wireless channel is converted into PHY layer packets by the TRACE Packet Converter module.

The interaction of the TRACE Manager and the application layer is handled by the Data Manager module. The Data Manager module provides an interface between the TRACE Manager and the application layer. It is responsible

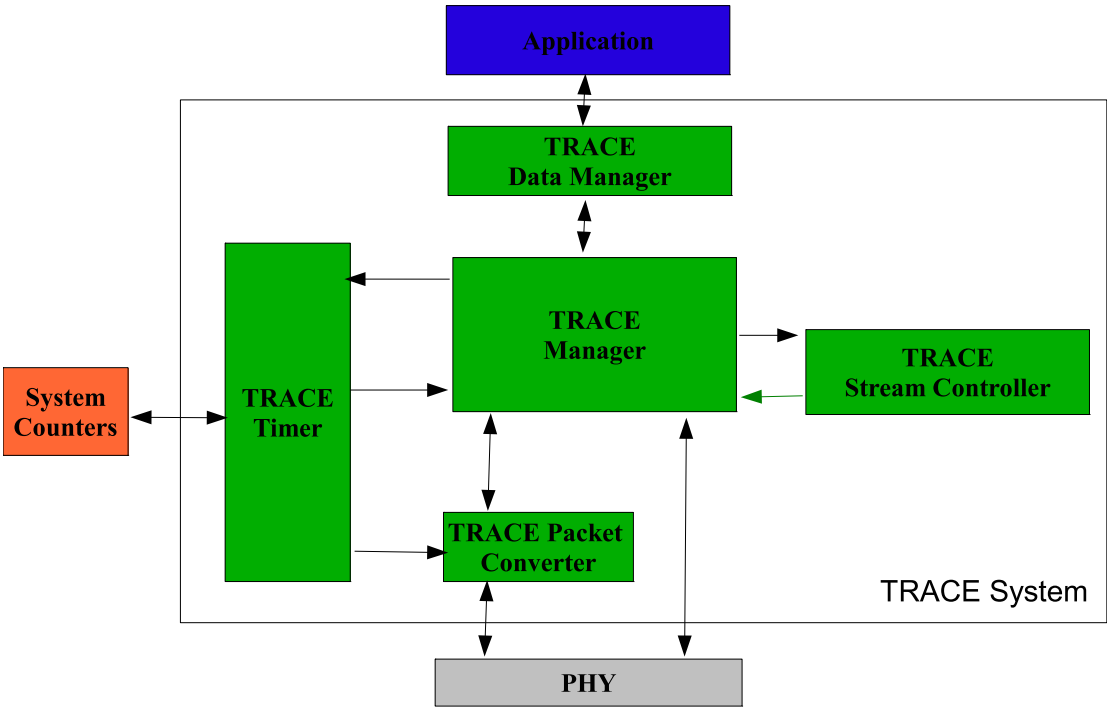


Figure 7.2: The interaction of the TRACE layers with other layers.

for organizing the generated data to be sent and storing the data from received streams.

The network layer services in the TRACE Manager are controlled by the Stream Controller. The Stream Controller maintains the information about the ongoing voice streams. The TRACE Manager decides whether to take part in relaying a particular stream or not based on the information stored by the Stream Controller.

7.3.1 TRACE Timer

The timing services within the TRACE system are handled by the *TRACETimer* implemented in "trace_timer.h" available in Appendix A.

For the proper operation of the TRACE protocol, an accurate representation of time is needed. Regular Windows Time services library only provides an accuracy

of 10ms. This accuracy is not sufficient for TRACE since the lengths of TRACE system slots are much smaller, as shown in Table 7.2.

In order to provide a higher accuracy, *TRACETimer* is implemented to be based on the lower level system counters. Windows APIs, namely *QueryPerformanceCounter* and *QueryPerformanceFrequency* [113], expose methods to obtain the number of CPU cycles that enable high resolution timing. On our testbed system that is equipped with an INTEL Core^{TM} i7-970 processor running at a maximum of 3.2GHz, these APIs provide a maximum resolution of $1/(3.2*10^9) \simeq 0.3\eta s$. Moreover, these APIs are only called in a single thread to prevent the frequency variations on threads running on different cores.

TRACETimer is responsible for converting the counter readings to time readings based on the frequency of the counter. *TRACETimer* keeps the counter value at the time of initialization as a reference point and reports time readings in sub micro second accuracy compared to this reference point.

For synchronization purposes, *TRACETimer* also maintains an offset value that is initialized to zero. The offset value is added to the measured time when reporting time readings to the TRACE Manager. By adjusting this offset value, the TRACE system is able to synchronize itself to other nodes in the network. The details of the synchronization process are available in Section 7.5.

7.3.2 TRACE Packet Converter

In addition to the data packets, TRACE uses various control packets, namely Beacon, CA, Contention, Header and IS packets. These packet carry a variety of information fields generated by the TRACE Manager. The information carried by these fields can be grouped under 4 types of variables:

- Flags: Embeds single bit information. Examples include start up flag, con-

Table 7.2: Packet sizes and slot lengths for the TRACE system.

Packet Type	Size (bytes)	Tx Duration (μs)	Slot Length (μs)
BEACON	4	102.545	232.5455
CA	4	102.545	232.5455
CONTENTION	4	102.545	232.5455
HEADER	10	106.909	236.9091
IS	10	106.909	236.9091
DATA	109	178.909	308.9091

tinuing stream flag, multi-cast member node flag, etc.

- Short Variables: Consists of multiple bits of information. These variables can take more than 2 values. Examples include packet type, slot number, packet priority, DataDisseminationMode.

- Long Variables: Consists of a few bytes of information. Examples include, node ID and data packet ID.

- Data Payload: Consists of multiple bytes of information.

In a wireless communication system, transmitting each additional bit consumes valuable resources such as energy and bandwidth. The variables in the transmitted packets should be represented as concisely as possible in order to reduce the duration of the transmission.

In order to efficiently set and retrieve data for each variable type, different operations should be performed depending on the data type. For instance, while bit-wise operations are required for flags, memory copy operations using multiple bytes long registers are preferred for faster operation on the data payload. In addition to this, many variables are represented by a few bits taking into account

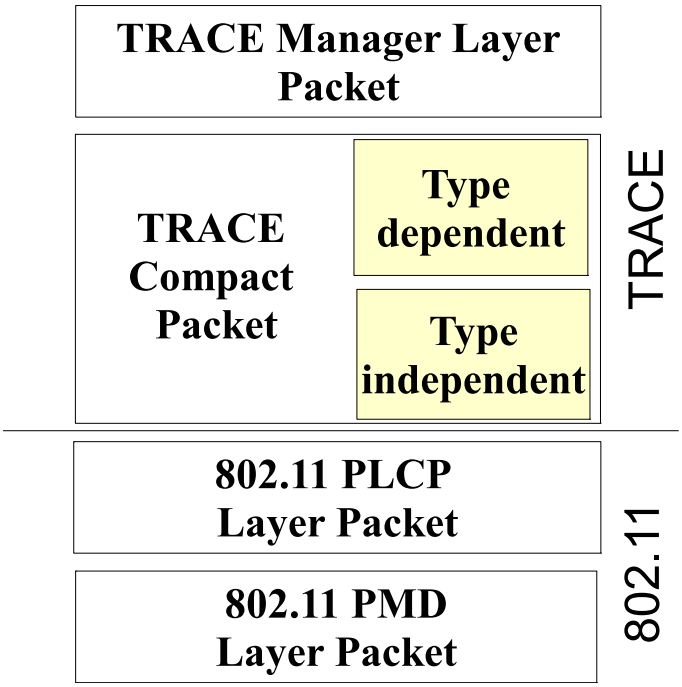


Figure 7.3: TRACE Packet Hierarchy.

the possible range of them. Variables shorter than one byte should be combined with each other using bitwise manipulators for efficient data representation.

On the other hand, the operation of the TRACE manager is independent of the representation of these variables in the transmitted packet and only depends on their value. For the purposes of faster access and easy processing of the variables at the TRACE Manager, it is preferred for these variables to be stored in more common data types such as Boolean, integers and integer arrays.

In order to simplify the retrieval and usage of the variable at each layer without sacrificing efficiency, the TRACE system uses a packet hierarchy as depicted in Fig. 7.3. The packet types above the PLCP layer packets are TRACE specific types and are implemented as C++ classes with conversion functions from and to the packet class above themselves. The details of the implementation of each packet type class are available in Appendix B.

For easier access to variables and compatibility issues, the TRACE layer operates using a single packet type *TRACE Manager Layer Packet* implemented in the *TRACEAllInOnePacketType* class. This is an inclusive class that has variables to store all the variables of all packet types along with additional fields for time of reception and received power level. At the receiver side, the TRACE Packet converter populates these additional fields after the conversion before passing the packet to the TRACE Manager. A single packet class in the TRACE Manager enables easier implementation of various functions.

The variables for each packet type are represented in the most compact form in the TRACE Compact Packet. There are 2 subclasses of the TRACE Compact Packet: type dependent and type independent. Each packet type in the TRACE system is implemented as a separate class in the type dependent TRACE Packet level together with a specific conversion process from *TRACEAllInOnePacketType* depending on the packet type. The variables are converted to bit-wise representation considering their range and combined with each other at this level. Depending on the packet type, packets in *TRACEAllInOnePacketType* are first converted into the corresponding type dependent TRACE Compact Packet type and then embedded into a type independent TRACE Compact Packet type. At this level, the contents of the variables are not accessible and the packet is represented with a stream of bits of a given length. Type independent TRACE Compact Packet type also adds a one byte encryption header in the beginning of the packet. Optionally, the packet contents of type dependent TRACE Compact packet type are encrypted in the type independent TRACE Compact packet type using the parameters in the encryption header.

The resulting packets of TRACE Compact Packet Type are further encapsulated into an 802.11 PLCP packet with a PLCP Header and a 4 byte long CRC field.

Finally, a preamble is added to the PLCP layer packet before it is modulated and converted into an 802.11 PMD layer packet type. At this level, the packet is represented by complex samples related to the in phase and quadrature components of the waveform to be transmitted by the RF front-end. The generated complex samples are transferred to the radio controller board (RCB) and stored in the RCB's memory. The actual transmission of the packets is controlled by the TRACE Manager and triggered using a separate mechanism.

Similarly, on the receiver side, after PHY layer headers are removed, the packet contents are saved into a type independent TRACE Compact Packet. At this layer, packet contents are decrypted using the encryption header, and the packet type is determined using the common packet type bits. The received packet is then converted into the corresponding type dependent TRACE Compact Packet and eventually to *TRACEAllInOnePacketType* before it is passed to the TRACE Manager.

7.3.3 TRACE Data Manager

The Data Manager module of the TRACE system provides an interface to the application layer. Its responsibilities include the responsibilities of the layers above and including the transport layer.

The target application of the TRACE system is real time voice communication. Voice communication has been shown to be relatively resilient to packet losses, and the main concern is the timely delivery of the data. Thus, a UDP like service is targeted at the Data Manager module. The Data Manager is responsible for data segmentation and reassembly. Unlike TCP, neither flow control nor congestion control is implemented since the main concern is the timely delivery of the data.

At the transmitter side, the Data Manager closely monitors the data generated by the application layer. The generated data is segmented into packets of fixed

length. Each packet is marked with a unique identifier that consists of a 2 byte data source node ID and a 2 byte data sequence number.

At the receiver side, the Data Manager sorts the packets according to the data source node identifier. Received data payload of each stream associated with each data source node identifier is saved to a separate file. The Data Manager reassembles the segmented packets. Segments corresponding to missing data packets detected by data sequence numbers are marked and potentially used for error correction services at the application layer.

For each stream identified by the source node ID, the last received packet sequence number is stored in the TRACE Stream Controller. The sequence number of each received packet is compared with the sequence number of the last received packet. For missing packets, to preserve the file size of the stream at the receiving end, the Data Manager adds a fixed payload with a length equal to the length of the missing data.

7.3.4 TRACE Stream Controller

The protocols in the TRACE family are mixed layer protocols covering the data link layer as well as network layer services. Our implementation combines the data link layer services of CDCA-TRACE with the network layer services of U-TRACE for the most comprehensive operations. The network layer in U-TRACE combines the network wide broadcasting and the multicasting services in a single framework and allows their coexistence.

The routing decisions of the TRACE Manager are combined under a separate module and are handled by the Stream Controller. The Stream Controller module stores the states of each detected ongoing stream in a dynamic list implemented in the "*StreamRegister*" class. Each entry in the dynamic list stores information about a stream and is stored as a "*Stream*" class object.

The information about the ongoing streams are received by the IS and Data packets. A new entry is created if the stream is not found in the *StreamRegister*. For each received IS and data packet, the corresponding entry is updated together with the last update time. Streams that are not updated for a given amount of time are marked as obsolete and are removed from the *StreamRegister*.

In our implementation, source nodes can transmit a single stream using a single routing strategy. Hence, streams are identified by the data source node ID in the *StreamRegister*. For each stream, in addition to the type of routing service requested by the stream and the time of the last update, a variety of variables are kept depending on the routing strategy requested by the stream.

For local broadcasting services, only the last received packet ID is kept. New packets are identified by comparing the packet ID in the IS packets with the packet ID stored in the Stream Controller. The receiver sleeps in the corresponding data slots for packets that have been received before. For streams requesting network-wide broadcasting services, the ID of the immediate upstream node and the last received downstream ACK time are stored in addition to the variables kept for local broadcast routing. For streams requesting multicasting services, the downstream node ID and the last upstream ACK time are also stored in addition to the variables stored for streams requesting network-wide broadcasting services. The relaying decisions are calculated based on the stored data following the rules of the NB-TRACE and the MC-TRACE protocols for network-wide broadcast and multicast routing, respectively. These rules are implemented in the Stream Register. The TRACE Manager decides whether to relay the messages of a particular stream by querying the Stream Register.

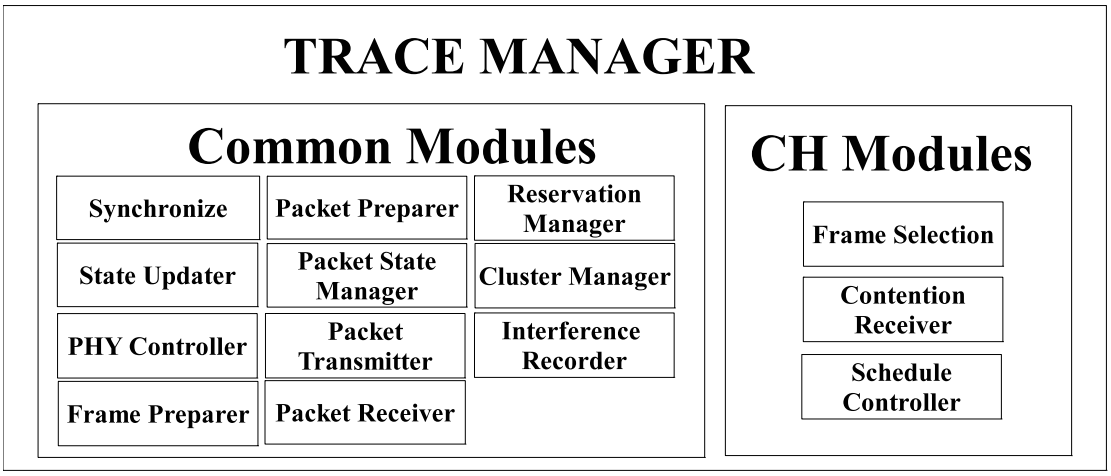


Figure 7.4: Sub-modules of the TRACE program.

7.3.5 TRACE Manager

The TRACE Manager Module is responsible for the data link layer services of TRACE. This is the main module that keeps a collection of variables determining the state of the TRACE system and a number of subroutines determining the processing methods of the available data. This module is implemented in "traceextension.h" and "traceextension.c" files as a C++ class, namely "*MAC_CDCAUTRACE*". The implementation is available in Appendix C.

The TRACE Manager is a collection of variables and the methods that calculate the behavior of the TRACE system based on these variables. The functionality of the TRACE manager can be grouped in sub-modules as shown in Fig. 7.4.

CH Modules

The submodules are grouped as CH specific modules that are used by the nodes that take the CH role in the network and common modules that are used by all the nodes. The submodules used by the CHs are as follows.

Frame Selection Implemented in the *SelectGTDMASlot* function. This module is used by the CHs to select the frame operation. If the dynamic channel allocation algorithm is enabled, instead of a single frame, a number of frames that is calculated based on the local load of the CH are selected. The frame selection is based on the interference of each frame that is measured by the interference levels in the Beacon and the CA slots. When a previously selected frame is not selected in a subsequent superframe, existing data slot reservations of that frame are moved to the other selected frames. This enables continuous operation for the nodes with ongoing reservations in the case of a change in the selected frames at the CH.

Contention Receiver Implemented in the *CheckTransmissionSchedule, InsertTransmissionSchedule* functions. CHs reserve the data slots for nodes requesting resources using this submodule. For successfully received contention packets, data slots are assigned opportunistically starting from the slot with the least interference based on the interference levels in the corresponding IS slots. When all data slots are already reserved, a priority scheme is used to selectively assign the slots. The priorities of the streams are announced and updated through contention and IS packets. If the priority of the contending stream is higher than the priority of one of the reserved streams, the reservation of the stream with the lowest priority is replaced with the contending stream.

Schedule Controller CHs cancel the reservations of a stream under two conditions: i) if "end of the stream" is indicated by the stream through IS packets, or ii) if the reserved IS slot is not used. The status of the stream obtained through the IS packets is recorded in a variable array in *ScheduleControlTable*. The transmission schedule is updated at the beginning of each frame based on the *ScheduleControlTable*.

Common Modules

The rest of the submodules of the TRACE Manager are used by all the nodes in the network regardless of the CH role and are described as follows.

Synchronizer Implemented in the *BeaconSyncronize* and *ContentionSyncronize* functions. The time reference of the transmitter node is calculated based on the received beacon and contention packets. The time reference calculated by the packet is compared with the time reference of the TRACE Manager for synchronization. The offset value of the TRACE Timer module is adjusted to synchronize the receiver.

State Updater Implemented in the *UpdateTraceTime* function. The current time reading obtained through the TRACE Timer module is used to update the state of the TRACE protocol. The time reading is further used to calculate and update the state of the TRACE Manager. The TRACE Manager states are based on the slot type within the frame. The state transitions are depicted in Fig. 7.5.

PHY Controller Implemented in the *DoIListenThisSlot* and *DoITXThisSlot* functions. The operation mode of the PHY layer is calculated based on the state of the TRACE Manager and time spent in that state, CH role and the time in the state of operation.

Frame Preparer Implemented in the *SetSST* function. In the beginning of each frame, this function is called to update a number of variables that will be used throughout the frame. For CHs, whether the upcoming frame is selected for the operation of the cluster is determined. The CHs operate only on selected frames and follow the role of a non-CH node in other frames. Finally, the contention and the SYNC packets used in the synchronization algorithm are scheduled in this submodule.

Packet Preparer Implemented in the *preparePacket* function. This function prepares a packet of a given type and embeds information required into the packet.

Packet State Manager The status of each packet type is kept in the corresponding variables, namely *BeaconPacketState, CAPacketState, ContentionPacketState, HeaderPacketState, ISPacketState, and DataPacketState*. The state of the packet indicates the existence of a copy of the given packet type in the RCB's memory and whether the copy needs to be updated.

Packet Transmitter Implemented in the *HandleOutgoing* function. The TRACE Manager is notified about transmissions at the time of transmission using this function. Depending on the type of the packet, the packet states and corresponding TRACE Manager variables are updated.

Packet Receiver Implemented in the *HandleIncoming* function. Provides the interface of the TRACE manager with the underlying PHY layer. Incoming packets are directed to corresponding subroutines based on the packet type.

Reservation Manager The transmission schedule of the entire superframe is stored in matrix form in the *TrSchSrcPrio* variable. The transmission schedule of each frame is prepared and announced by the CH in the Header packet, which in turn are used to update the transmission schedule of non-CH nodes.

Cluster Manager Implemented through the *CHTableStructure* type variable. The information about the existing CHs are recorded in this table and updated through the Beacon and Header packets. The entries that are not updated for a given amount of time are marked as obsolete and are deleted.

Interference Recorder Implemented in the *RecordChannelPower, ResetNumIntf, and ResetNumIntf2* functions. For Beacon, CA and IS slots, maximum

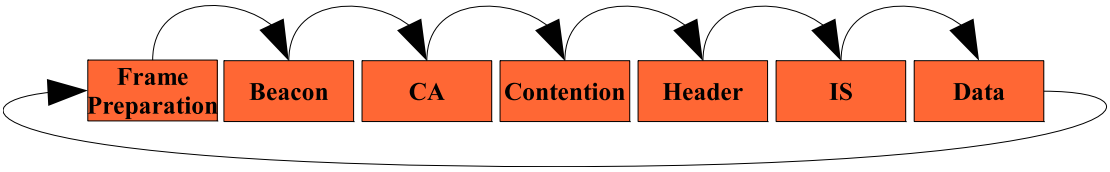


Figure 7.5: The states of the TRACE Manager.

channel power measurements are recorded in every frame. These measurements have a high variance due to noise and cross system interference. In order to smooth out the variance of the interference, we use an exponential smoothing algorithm to calculate perceived interference levels for each slot. The exponential smoothing algorithm combines the historical value of the interference level and the current measurement. We have a conservative implementation that has a very low smoothing factor for the cases where the measurement is higher than the previous perceived interference level.

The perceived interference levels for the IS slots are updated at the end of each frame. On the other hand, the perceived interference levels for the Beacon and CA slots are updated immediately at the end of the CA slot in every frame. By using the most recent values before the transmission of the Header, CHs prevent further collisions in the case of high co-frame interference. For non-CH nodes, global interference levels are used for selecting a CH among the accessible ones for channel access.

7.4 Multi-threaded TRACE Implementation

Our TRACE implementation uses the parallel processing capabilities of the Windows programming environment and the *exclusive threading* APIs provided by the SORA development libraries.

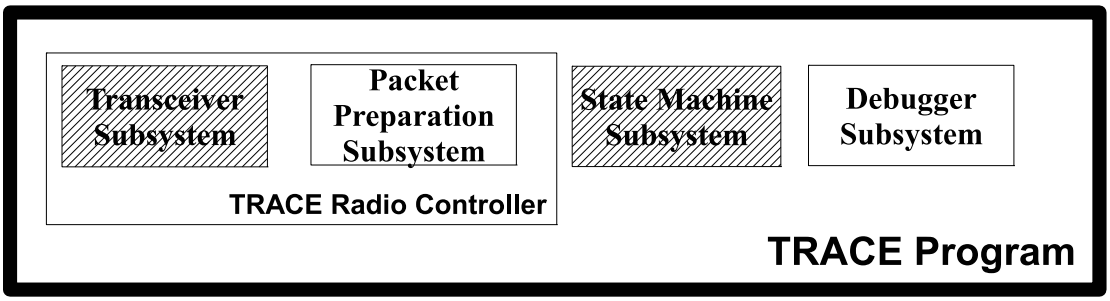


Figure 7.6: Functional subsystems of the TRACE Program.

The TRACE system is required to respond to other nodes in the network in a timely manner. This imposes strict timing requirements for various tasks. In order to meet these timing requirements, the implementation of the TRACE system is divided into functional programming subsystems that simultaneously run on parallel threads. As shown in Fig. 7.6, the functional subsystems of the TRACE system are:

- Transceiver Subsystem
- Packet Preparation Subsystem
- State Machine Subsystem
- Debugger Subsystem

The TRACE program drives the SORA Radio Controller Board using the *TRACE Radio Controller Subsystem*. The TRACE layer and the underlying PHY layer are decoupled and only interface with each other through the *TRACE Radio Controller Subsystem* that consists of two subsystems: the *Transceiver Subsystem* and the *Packet Preparation Subsystem*. The *Transceiver Subsystem* controls the PHY layer RX/TX operations, and the *Packet Preparation Subsystem* is responsible for managing the packets on the SORA Radio Controller Board's memory.

In our TRACE system, we have chosen to use the 802.11b PHY layer implemented through the UMXDot11 extension of the SORA SDR system as the main PHY layer. Our TRACE implementation can easily be adopted for different PHY layers by changing the instructions for packet preparation and reception modes in the *TRACE Radio Controller Subsystem*.

The operation of the TRACE protocol depends on the its state, which changes with time. The *State Machine Subsystem* is dedicated for polling the system time and changing the state of the TRACE protocol on time.

While the TRACE system is running, the user is periodically notified about the states of various variables. This task is performed by the *TRACE Debugger Subsystem*. This subsystem queries the system timer and periodically displays the system information on screen.

Each of the subsystems depicted in Fig. 7.6 runs on a separate exclusive thread. The SORA thread manager allocates dedicated CPU cores to each of these threads. Since *ethreads* are non-interruptible, the non critical subsystems, namely the *Packet Preparation Subsystem* and the *Debugger Subsystem*, periodically return back to the system to be reassigned to a proper core by the SORA core library for best core allocation. On the other hand, the *Transceiver Subsystem* and the *State Machine Subsystem* depicted with shaded boxes in Fig. 7.6 operate within an infinite loop since the performances of these subsystems are negatively affected by the dynamic *ethread* scheduling overhead. The TRACE system needs at least 2 dedicated and one shared CPU cores in addition to the computation resources used by the Operating System (OS). Hence, the TRACE system requires a minimum of quad-core system while a hexa-core system is recommended.

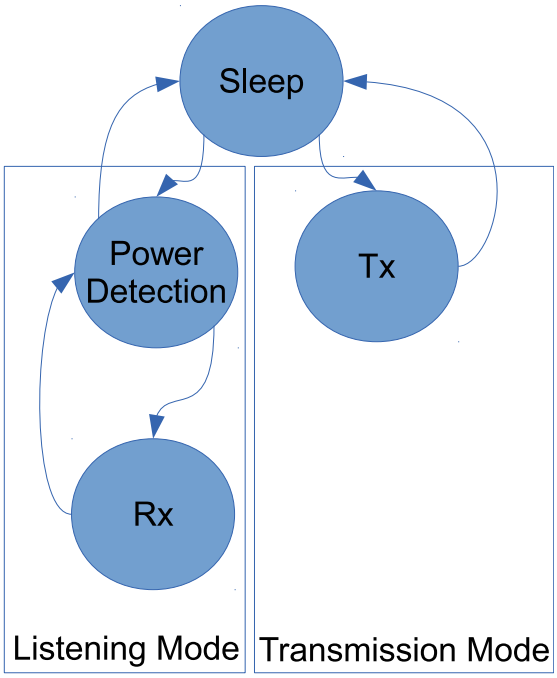


Figure 7.7: The states of the TRACE Transceiver Subsystem.

7.4.1 Transceiver Subsystem

The TRACE system runs the Transceiver subsystem in an *ethread* that is created by *TRACE11TxApp* at the beginning of the application. This *ethread* runs the *TRACE11BTRX* function that implements the transceiver subsystem's operations in an infinite while loop. Since the operation of the *ethread* is critical, this subsystem does not return the resources dedicated to the *ethread* unless a flag signaling the end of operation is set, namely *ISShutDown*.

Inside the loop, the *Transceiver Subsystem* constantly polls the TRACE Manager variables, *DoITransferinThisSlot* and *DoIListentoThisSlot*, to determine the type of operation. The positive value returned by the TRACE manager starts the transmission or listening modes of operation at the PHY layer, as shown in Fig. 7.7.

Transmission Mode

A positive *DoITransferinThisSlot* value switches the *Transceiver Subsystem* into transmission mode. In the transmission mode, *Transceiver Subsystem* starts a loop that constantly polls for the *TraceTimeInSlot* value reported by the TRACE Manager for the exact transmission time. For synchronization purposes, the physical transmission of a packet is delayed by a guard band period, namely *GBBeforeTX*.

Once the time in slot condition is met, the type of the packet to be sent is determined by querying the mode of operation from the TRACE manager. This value is stored in order to prevent multiple transmissions of the same packet within the same slot. In order to transmit the packet, the packet type to be sent should have been prepared and transferred to the SORA radio controller board in advance. The availability of the packet in the SORA Radio Controller Board is checked by querying the corresponding packet state flag from the TRACE Manager. If the packet on the radio controller board is outdated or does not exist, *Transceiver Subsystem* skips the transmission of the current packet and goes out of the transmission mode.

Provided that the packet to be sent is ready, the transmission is initiated using the pointer of the corresponding packet on the SORA Radio Controller Board. If the PHY transmission function returns a value indicating a successful transmission, the TRACE Manager is notified about the type of the packet that has just been transmitted. The required variables are updated in the TRACE manager and packet state is marked as *sent*. In addition to that, the debug monitor subsystem is also notified and the number of packets transmitted and the number of bytes transmitted statistics are updated.

An optional debugging mode is also implemented by the transmission mode of the *Transceiver Subsystem*. In this mode, *Transceiver Subsystem* saves the values of the appropriate TRACE variables and the TRACE Time reading in the

beginning and end of the transmission mode as well as the time reading at the beginning and end of the physical transmission.

Listening Mode

A positive *DoIListentoThisSlot* value takes the *Transceiver Subsystem* into the listening mode, and any change in *DoIListentoThisSlot* takes the *Transceiver Subsystem* out of the listening mode. In the listening mode, *Transceiver Subsystem* repeatedly calls the software power detection subsystem of the SORA UMXDot11 extension, namely *BB11BSpd*, in a loop. *BB11BSpd* not only returns a flag indicating the presence of a packet but also calculates and stores the measured channel power value in the *BlockEnergySum* variable of the *pSpdContext* object. The measured channel power is reported to the TRACE manager. If the TRACE Manager is in the Beacon, CA, or IS state, it compares the reported value with the *InFrameIntfLevels* corresponding to the slot and updates it if the measured value is higher. Later, *InFrameIntfLevels* are used by the *State Machine Subsystem* to update *InterFrameIntfLevels* following an exponential smoothing algorithm. Any change in *DoIListentoThisSlot* takes the *Transceiver Subsystem* out of the listening mode. Once the presence of the packet is detected, *Transceiver Subsystem* switches to the packet reception mode by calling the *BB11BRx* subsystem of the SORA UMXDot11 extension. *BB11BRx* returns a variable indicating the success or failure of packet reception. The time reading is recorded right after *BB11BRx* and used for synchronization purposes. In the case of a failure, the *Transceiver Subsystem* switches back to listening mode.

If the packet is successfully received, the first thing that *Transceiver Subsystem* does is to determine whether the received packet belongs to the TRACE network or not. Since we are operating on a commercial and commonly used PHY layer, there is a good chance that the received packet could belong to some other system transmitting with the same PHY parameters. These packets are eliminated from

the system using a TRACE specific encryption header. Each over the air packet includes a one byte long encryption header. Each received packet's first byte is compared to the predefined encryption header value, and the packet is discarded in the case of a mismatch. Next, the packet is converted into the *TRACEAllInOnePacketType* using the next 3 bit packet type field that is common to all over the air TRACE packet types. In the case of an unknown packet field type or any mismatch in the packet length with the length of the corresponding packet type, the packet is marked as an out of the system packet and is discarded. After a successful conversion, the recorded time stamps and packet power reading are added to the packet, and the converted packet of *TRACEAllInOnePacketType* is sent to the TRACE Manager.

Similar to the transmission mode, an optional debugging mode is also implemented by the reception mode of the *Transceiver Subsystem*. In this mode, *Transceiver Subsystem* saves the values of the appropriate TRACE variables and the TRACE Time reading at the beginning and end of successful packet receptions into a receiver debugging file.

7.4.2 Packet Preparation Subsystem

Another PHY layer dependent subsystem is the *Packet Preparation Subsystem* that is responsible for the preparation of packets. For synchronization purposes, the TRACE system must tightly control the time of the physical transmission of the packets. In order to accomplish this, packets have to be prepared and transferred to the SORA Radio controller board in advance. The *Packet Preparation Subsystem* performs this operation through the *DoTRACE11BPreparePacket* function running on a separate *ethread*.

In the initialization phase, the TRACE system allocates dedicated memory on the Radio Controller Board (RCB) large enough to store one TRACE packet of

each type: Beacon, CA, Contention, Header, IS and Data Packets.

Complex samples of packets are stored on the RCB's memory. At any time, the *Packet Preparation Subsystem* allows storing one packet of each TRACE packet type; namely beacon, CA, contention, header, IS, or DATA. A pointer variable is maintained to store the locations of the packets within the RCB's memory. When a packet is transferred to the RCB's memory, the corresponding pointer is updated to point to the packet.

The responsibility of the *Packet Preparation Subsystem* is to create these packets, transfer them to the RCB before their transmission and update them as necessary. For each packet type, a state variable in the TRACE Manager keeps the status of the packets in the RCB's memory. The state of the packet indicates: i) the existence of the packet in the RCB; and ii) whether it needs to be updated.

The state of the packets on the RCB is stored in the TRACE Manager. The *Packet Preparation Subsystem* checks these states one by one to find a packet type that needs to be prepared. In order to be reassigned to a proper core, *DoTRACE11BPreparePacket* returns a positive value after cycling through all packet types.

Once the *Packet Preparation Subsystem* finds a packet type that needs to be prepared, the previous instance of the same packet type is removed from the RCB memory if it already exists. Then, the packet is retrieved from the TRACE Manager, converted into complex samples to be transmitted and transferred into the RCB. If the transfer is successful, the *Packet Preparation Subsystem* updates the corresponding packet state variable of the TRACE Manager.

For easier access to packet contents and decoupled PHY layer operation, the TRACE Manager accepts and outputs packets in *TRACEAllInOnePacketType*. The packet is converted into an over the air packet type for the most compact representation of the data, as discussed in Section 7.3.2. A one byte encryption header is added to the packet before it is embedded into an 802.11b PHY layer

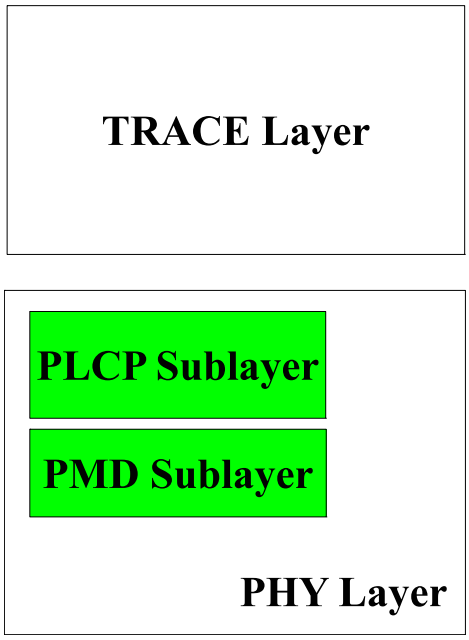


Figure 7.8: The OSI model representation of the layers under TRACE.

PSDU together with a 4 byte CRC for error detection.

Complex samples are prepared using the *BB11BPMDPacketGenSignal* function of the *UMXDot11* library. The operations of the 802.11 PHY layer are divided into Physical Layer Convergence Procedure (PLCP) layer and Physical Medium Dependent (PMD) layer as seen in Fig. 7.8. Within the transmission chain, *BB11BPMDPacketGenSignal* performs the operations of the PLCP sublayer and the PMD sublayer other than the actual data transmission. Based on the chosen PHY layer parameters such as the data rate, preamble type, *BB11BPMDPacketGenSignal* adds a PLCP Preamble and a PLCP Header to the PSDU forming the PLCP Layer Packet. *BB11BPMDPacketGenSignal* also calls the corresponding functions from the SORA library for scrambling, modulation, and spreading blocks of the 802.11 PMD sublayer transmission chain.

The complex samples then are transferred to the RCB, and the corresponding pointer is set at its location on the RCB's memory using the *SoraURadioTransfer*

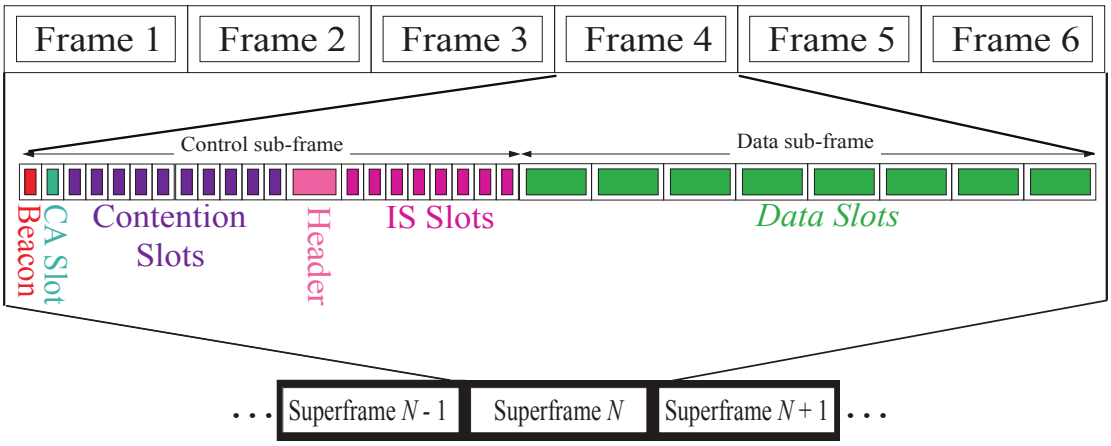


Figure 7.9: TRACE frame and slots.

function of the SORA UMXDot11 library. The *Packet Preparation Subsystem* locks the buffer at the RCB's memory before the transfer in order to prevent any memory access clashes and releases the lock as soon as the transfer is completed.

Finally, the corresponding packet state variable is updated if the transfer is successful.

7.4.3 State Machine Subsystem

Being a TDMA based protocol, in TRACE, time is divided into equal length *Frames*. As depicted in Fig. 7.9, each frame consists of a number of slots and a number of frames are grouped together forming a superframe (SF). There are different tasks that are required to be performed by the TRACE system in each slot. The responsibility of the *State Machine Subsystem* is to update the TRACE Clock based on the system counter, determine the slot that the system is in and update necessary variables that form the TRACE Manager's state.

The operation of the TRACE protocol depends on its state, which changes with time. The *State Machine Subsystem* is dedicated for polling the system time and changing the state of the TRACE protocol on time.

It is critical for the TRACE system to detect the beginning of slots and react in a timely manner. This makes the operation of the *State Machine Subsystem* an essential task for the operation of the TRACE system. Our implementation dedicates a CPU core for the operations of the *State Machine Subsystem* through an *ethread* that starts at the beginning of operation and does not return the resources until the system is shut down.

After initialization, *TRACE11TxApp* creates a thread with a handle of *hThreadTraceManager* and running the *DoTRACEManager* function. *DoTRACEManager* implements the operations of the *State Machine Subsystem*.

DoTRACEManager updates the TRACE Manager by calling its *UpdateTraceTime* method in a while loop. *DoTRACEManager* also implements an optional debugging routine that records a debug line if the time difference between two consecutive updates is larger than a certain threshold.

The operation of the *UpdateTraceTime* function begins with updating the TRACE time based on the system counters. The frame number is updated along with the *TimeinFrame* variable that indicates the time since the beginning of the frame. The mode of operation and time in slot is calculated based on *TimeinFrame* and updated in the TRACE Manager. In the case of a change in the mode of operation, depending on the previous state and the upcoming state, additional operations are performed. These additional operations are:

Beginning of a New Frame The TRACE Manager prepares the internal variables for the upcoming frame through the *setSST* function. The main operations of *setSST* are: updating the transmission schedule and the CH table; updating perceived interference levels of the IS slots of the previous frame; resetting the schedule control table, interference recording registers and data slot source list; updating SF number and the cyclic frame number; selecting the frames of operation (FOO) based on updated interference

measurements and scheduling CA for CHs; for non-CH nodes checking for restart-up conditions in the case of no access to CHs.

End of CA Slot The perceived channel power variables are updated based on the channel power measurements during the Beacon Slot and the CA Slot. This update enables the CHs to immediately stop the header transmission in the case of large interference.

End of Beacon Slot The packet states of contention and header packets are updated. These packets are marked as obsolete if they exist in the RCB memory in order to trigger their update by the *Packet Preparation Subsystem*. The *Packet Preparation Subsystem* is also triggered to update the IS packet, if the node has new data to be sent.

End of the last IS slot In order to reflect the up-to-date information of the current frame in the Beacon and CA packets, these packets are marked as obsolete, triggering their update by the *Packet Preparation Subsystem*.

End of the last data slot If the node has new data to be sent, the data packet status is updated to trigger the update of the data packet by the *Packet Preparation Subsystem*.

Finally, based on the calculated slot of operation and time in slot, the intended operation of the PHY layer is determined and *Transceiver Subsystem* is triggered by setting the appropriate flags.

7.4.4 Debugging Subsystem

Although each subsystem implements recording of optional debugging messages at critical points, during the course of normal operation, the state of the TRACE Manager is hidden from the user. In order to monitor the operation of the TRACE

Manager, a separate thread is devoted to periodically notify the user about the status of the operation.

Debugging Subsystem is implemented in the *DoTRACEQuery* function and is executed in a SORA thread with the *hThreadTraceQuery* handle. There are 2 types of queries performed by the *DoTRACEQuery* function, namely TRACE Manager Query and PHY Layer Query.

TRACE Manager Query is responsible for periodically reporting the state of the TRACE Manager. A reporting period of 1s is chosen. This query stores the last reporting time and compares it with the TRACE Manager's time. If the difference between the TRACE Manager's time and the last reporting time is larger than the reporting period, a new report is generated by calling the *PrintTRACEState* function. The report includes variables that indicate:

- TRACE Manager's mode of operation,
- the states of the packets within the RCB's memory,
- the number of tx/rx packet counters of the TRACE Manager,
- the IDs of the last received data packets from each source,
- whether the TRACE Manager is in source mode,
- whether the node is a CH and its chosen frame(s) of operation,
- transmission schedule of the superframe as observed by the node,
- interference level measurements,
- the CH Table that stores the existing CHs as observed by the node, and
- the list of entries in the Stream Controller that stores the states of the ongoing streams as observed by the node.

The PHY layer query is responsible for periodically reporting variables related to the operation of the PHY layer. These variables are updated by the Transceiver Subsystem. The printed variables indicate:

- the number of times the power detection algorithm detects the presence of a packet
- the number of successful PHY layer receptions and the reception rate, and
- the number of PHY layer packets sent, and the tx rate.

The operation of the *Debugging Subsystem* has a lower priority compared to the operation of the *Transceiver Subsystem* and the *State Machine Subsystem*. Hence, after calling each of TRACE Manager Query and PHY layer query, it returns the resources back to the SORA thread manager with a positive return value. The SORA thread manager recalculates the optimal core assignment and reassigns the *Debugging Subsystem* to a proper core.

7.5 Synchronization

Since TRACE is a TDMA based system, the nodes in the system must be synchronized. In the original TRACE protocol, the synchronization was not considered, as synchronization was assumed to be provided by external mechanisms such as GPS. In our TRACE prototype implementation, we added an algorithm to the TRACE protocol that provides synchronization using message exchanges between the nodes in the network.

The TRACE Manager is responsible for the MAC layer operations and tightly controls the radio resources. The TRACE Manager sets the radio in listening, transmission and sleep states based on the state of operation, which in turn is based on the TRACE Timer module.

In TRACE, the nodes should be synchronized up to one frame duration. In other words, although absolute frame number or the cyclic prefix of the frame in a superframe may differ between nodes, the start time of each slot should be synchronized. To achieve this level of synchronization, we have created a synchronization subsystem based on packet receptions.

The latency of a packet is caused by:

- Sender Side Delay: This delay includes the creation of the PHY layer packet, the creation of the complex samples, transferring the samples to the radio controller, and the time to transmit the packet.
- Propagation Delay: This delay includes the travel time of the electromagnetic waves over the air.
- Reception Delay: This delay is associated with the processing time of the packet at the receiver side. It includes the time between the reception of the complex samples at the RCB and the reception of the packet by the UMXDot11 API.

At the sender side, the information coming from upper layers are embedded into a PHY layer packet that includes a PHY layer header. Up to that point, the information in the packet is stored as bits. Then complex samples are created following the chosen modulation scheme. In SORA systems, these operations take place on the GPP processor, and the samples are stored in the PC's memory, which drives the SORA card. This data is transferred to the onboard memory on the SORA card before over the air transmission. Although the transfer takes place on the relatively fast PCI x8 bus, due to the transfer initiation latency, the transfer operation takes a considerable amount of time, as shown Table 7.1. Moreover, the measured latency values to transfer the packet to the RCB at the sender side are found to have a high variance. These variations alter the physical

transmission time of the packet and disrupt the synchronization at the receiver side.

In order to overcome this, we created a packet preparation subsystem that runs on a separate thread and is responsible for preparing the packets in advance. The packet preparation subsystem creates over the air packets, prepares complex samples representing the packets, transfers them to the SORA RCB and stores a pointer for each packet type on the RCB's memory. The packets stay on the memory of the RCB until they are transmitted by the RadioTransiever thread. The RadioTransiever subsystem is controlled by the TRACE Manager and initiates the transmissions at the corresponding slot based on the TRACE timer.

For debugging purposes, the transceiver thread records the value of the TRACE timer right before the transmission API is initiated and right after the transmission API returns. For various packet sizes, the averages and the standard deviations of transmission time calculated over 10,000 sample transmissions together with the theoretical time to transmit are listed in Table 7.3. It has been observed that the duration of the transmission measured by the difference in time recordings follow the theoretical time to transmit the packet. This verifies two facts: i) the transmission function is an inline function that returns after the operation is complete, and ii) the time to initiate the transmission for samples stored in the RCB's memory is independent of the packet size.

Propagation delay is related to the physical traveling of the communication carrier over the communication medium. For certain environments this delay is quite large and plays a significant role. For instance, in acoustic communication, propagation delay constitutes a significant portion of the packet delay due to the slow propagating properties of acoustic waves compared to electromagnetic waves. Another example is satellite communication in which the electromagnetic waves take a long time to reach the destination due to the large distance between the

Table 7.3: Comparison of the theoretical and observed time to transmit packets of various sizes.

Packet Size (bytes)	Theoretical (μs)	Measured Mean (μs)	Measured Std (μs)	(Mean - Theoretical) (μs)
150	298.534	310.380	1.650	11.845
100	263.023	275.507	1.708	12.485
50	227.511	237.824	1.760	10.313
20	206.205	217.367	1.647	11.163
10	199.102	208.812	1.700	9.709

transmitter and the receiver. On the other hand, practical MANET applications operate over relatively shorter ranges varying from 250m to 2km. In the 802.11 systems, the typical operational range of the 802.11b packets, d_{range}, is 250m for line of sight transmissions. Considering the propagation speed of electromagnetic waves in air, $SProp$, the maximum propagation time, $DProp_{max}$, is calculated by

$$DProp_{max} = \frac{d_{range}}{SProp} = 0.83\mu s. \tag{7.1}$$

Since the $DProp_{max}$ is small, the dependence of the propagation delay on the receiver's location is negligible.

A sample 802.11 frame is shown in Fig. 7.10. At the receiver side, the presence of the packet is detected using the preamble. However, the moment at which the packet is detected cannot be used for synchronization purposes since the receiver does not necessarily have to start at the beginning of the preamble for a successful packet reception. The purpose of the preamble is to lock the phase of the internal oscillator at the receiver side to the phase of the transmitter. After that, the receiver waits for the start of the frame delimiter (SFD), which marks the beginning of the PHY header of the 802.11 packet. This is another useful point that can be used for synchronization purposes. However, the internal clock is neither recorded

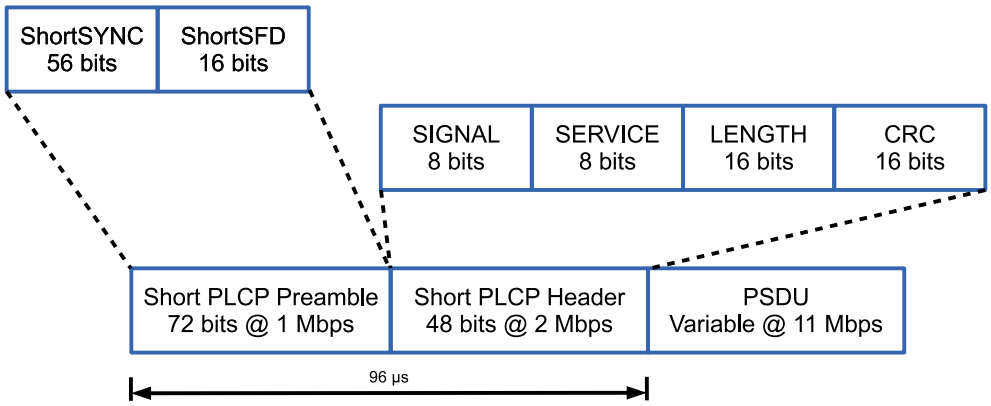


Figure 7.10: 802.11b PHY frame format used in the TRACE system.

nor reported by the 802.11 PHY block and hence cannot be used by the TRACE layer.

Our aim is to have a generic implementation of the TRACE layer, which would allow interoperability with other PHY layers. TRACE, being an upper layer, only is notified at the end of the packet reception. There is no packet buffer between the PHY layer and the TRACE layer. The TRACE layer is notified of the reception of the packet as soon as it is demodulated. The small delay between the physical reception of the last constellation and the notification of the TRACE layer consists of the demodulation, despreading and the data transfer delay. The time for these operations depends on the packet size. Since the size of a given TRACE packet is constant, the delay between physical reception of the packet and the notification of the TRACE layer is also approximately constant.

The reception time of the packet at the receiver depends on the transmission time on the transmitter side, $\tilde{T}_{tx}$, the difference between the receiver and the transmitter clocks, $\tilde{C}_{rx-tx}$, the propagation delay, $\tilde{D_{prop}}$, and the reception delay, $\tilde{D}_{rx}$, as in Eq. (7.2).

$$\tilde{T}_{rx} = \tilde{T}_{tx} + \tilde{C}_{rx-tx} + \tilde{D}_{prop} + \tilde{D}_{rx}. \tag{7.2}$$

$\tilde{T}_{tx}$ can further divided into an intended start time, T_{intent}, delay on the start of transmission, $\tilde{D}_{tx}$, and transmission duration, T_{TxTime}.

$$\tilde{T}_{tx} = T_{intent} + T_{TxTime} + \tilde{D}_{tx} \tag{7.3}$$

T_{intent} and T_{TxTime} are known at the receiver, and the rest of the delays can be modeled with a random variable $\tilde{D}$ which can be divided into its mean μ_D and another random variable $\tilde{Z} = \tilde{D}$ - μ_D as in Eq. (7.4).

$$\tilde{T}_{rx} = T_{intent} + T_{TxTime} + \tilde{D}. \tag{7.4}$$

7.5.1 Packet Based Synchronization

In order to explore the variance of $\tilde{D}$, we perform an experiment with two asynchronous systems. We set one system to periodically send fixed length packets with increasing sequence numbers. The intended period in between two transmissions is set to $SFT = 25\ ms$. We set the other system to always be in listen mode. Both the transmitter and the receiver systems keep a log of packet transmissions/receptions. Each log entry records the time stamps marking the end of transmission/reception, packet type, packet source ID, and packet sequence number. We analyzed the time difference between consecutive packets within each log containing 10,000 entries. Note that the consecutive packets are detected via packet sequence numbers.

We first investigate the time difference between consecutive packets at the transmitter side. We expect a packet to be sent at regular intervals equal to the super frame duration, SFT. Hence, using Eq. (7.3),

$$\Delta\tilde{T}_{tx} = T_{tx_2} - T_{tx_1} \tag{7.5a}$$

$$\Delta\tilde{T}_{tx} = (T_{intent_2} - T_{intent_1}) + (T_{TxTime2} - T_{TxTime1}) + (\tilde{D}_{tx_2} - \tilde{D}_{tx_1}) \tag{7.5b}$$

$$\Delta\tilde{T}_{tx} = SFT + \tilde{X} \tag{7.5c}$$

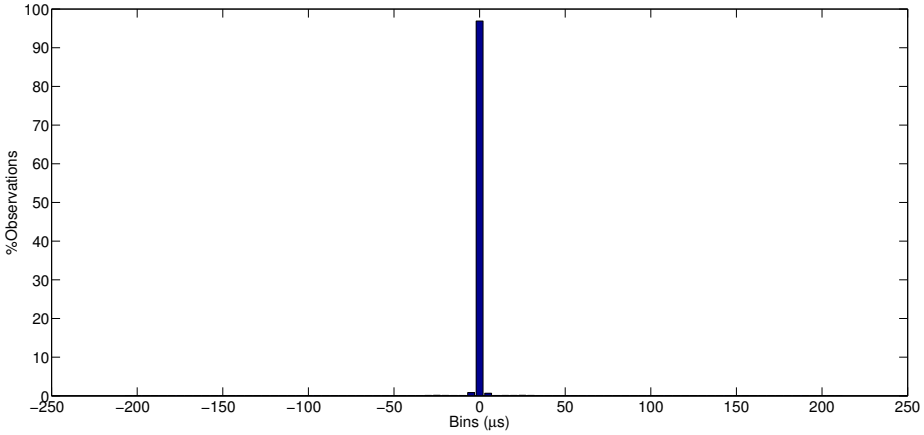


Figure 7.11: Histogram of inter packet transmission duration recorded at the transmitter.

where $\tilde{X}$ is a random variable with $Var(\tilde{X}) = 2 * Var(\tilde{D}_{tx})$ assuming the delays on two consecutive transmissions are independent. From the data analysis, we find that the mean, $\mu_X < 10^{-3}\mu s$ and the standard deviation, $\sigma_X = 5.873\ \mu s$. A histogram of the data $\Delta T_{tx} - SFT$ is shown in Fig. 7.11. The data is centered around $\mu_X = 0$ and 97% of the data lies within 2.5 μs of the mean.

Next, we investigate the time difference between consecutive packets at the receiver side. We expect packets to be received at regular intervals equal to the super frame duration, SFT. Hence, using Eq. (7.4),

$$\Delta T_{rx} = T_{rx_2} - T_{rx_1} \tag{7.6a}$$

$$\Delta T_{rx} = \Delta T_{tx} + \Delta C_{rx-tx} + \Delta D_{prop} + \Delta D_{rx} \tag{7.6b}$$

The experiments are conducted in a closed office environment, and the nodes

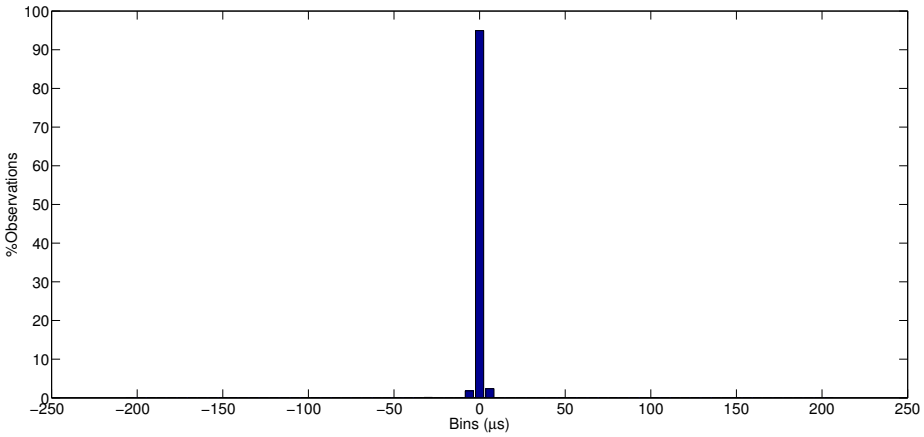


Figure 7.12: Histogram of $\Delta T_{rx} - SFT$ recorded at the receiver for a transmitter transmitting periodic packets with a period of SFT.

were stationary, so we expect ΔD_{prop} to be zero. Hence,

$$\Delta T_{rx} = \Delta T_{tx} + \Delta C_{rx-tx} + \Delta D_{rx} \tag{7.7a}$$

$$\Delta T_{rx} = SFT + \tilde{X} + \tilde{Y} + \tilde{Z} \tag{7.7b}$$

$$\Delta T_{rx} = SFT + \Delta D = SFT + \tilde{\theta} \tag{7.7c}$$

where $\tilde{Y}$ and $\tilde{Z}$ represent random variables associated with the clock drift in one SFT and the difference in the reception delay of two consecutive receptions, respectively. $\tilde{\theta}$ is associated with the combined effect of $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$. From the measured data, we find that the mean, $\mu_\theta = 0.225\ \mu s$ and the standard deviation, $\sigma_\theta = 5.594\ \mu s$. A histogram of the data $\Delta T_{rx} - SFT$ is shown in Fig. 7.12. The data is centered around $\mu_X = 0$ and 95% of the data lies within 3 μs of the mean.

The main cause of D_{rx} is the load on the receiver system. We can assume the time difference between two receptions ($\simeq SFT$) is large, making the samples of D_{rx} independent, and, in turn, the mean value of $\tilde{Z}$, $\mu_Z = 0$. Since μ_X is small,

the mean value of the clock drift in a SFT, μ_θ , is calculated as

$$\mu_\theta = \mu_X + \mu_Y + \mu_Z \tag{7.8a}$$

$$\mu_\theta \simeq \mu_Y \tag{7.8b}$$

Assuming the delays on two consecutive receptions are independent, $Var(\tilde{\theta}) = 2 * Var(\tilde{D})$ or $\sigma_D = \sigma_\theta/2 = 2.739\mu s$.

These observations show the possibility of creating a synchronization subsystem at the TRACE layer that is independent of the underlying PHY layer. The internal clock reading is recorded immediately at the end of reception by the TRACE Convergence Sublayer. This value is stored along with the packet and later used in the synchronizer subsystem in the TRACE Manager. The TRACE Convergence Sublayer then determines the packet type and converts the over the air received packets to TRACE layer packets. For each received packet, the next operation is to determine the packet type and to convert the received over the air packet to *TRACEAllInOnePacketType*. *TRACEAllInOnePacketType* is embedded into *TRACEMACPacketType* together with a timestamp of recorded time and packet energy. At the input point of the TRACE layer, packets of chosen types are sent to the synchronization subsystem.

7.5.2 Single-Hop Synchronization

In this section, we study synchronization in the TRACE Systems focusing on a single-hop network where the nodes in the network are close to one another such that each node can directly communicate with any other node. In order to address the synchronization problem, we present a new algorithm, namely *Single Side Synchronization Algorithm*. First we are going to present the algorithm and then test its performance in a two-terminal system in a single hop arrangement.

In a single hop connected network, every node can directly receive packets transmitted by every other node unless there is a collision. For such networks,

TRACE creates a single cluster that is managed by a single dynamically selected clusterhead (CH). In the beginning of each frame, the CH transmits a beacon packet signaling the start of a new frame. We use the Beacon packet as the primary source of synchronization. The nodes in the network synchronize their internal clocks to the Beacon packet and hence synchronize to the CH.

The initial synchronization is handled by the StartUpRoutine. The nodes start their operation in this mode in which they listen to the channel for a random duration longer than the duration of a superframe before transmitting a beacon packet and assuming the role of the CH. This mechanism provides proper operation for nodes starting at different time instances and reduces the collision probability of the first beacon packets. The collision of the first beacon packets creates a potential synchronization problem for the transmitters of the collided packets that assume a CH role and further assume that all the other nodes would be synchronized to themselves. However, since their packets are not received by other nodes in the network, another CH will be selected and all the other nodes would be following its schedule. Potentially, the beacon packet of the selected CH will not align with the listening periods of the transmitters of the collided packets and they will not be synchronized with the rest of the network. In order to solve this problem, the CHs alter their sleeping schedule and instead listen to the channel for the duration of an entire superframe with a probability, p_l. This mechanism ensures the reception of the beacon of the selected CH by the out of synch CHs and that, in turn, triggers the resignation mechanism.

For received Beacon packets, TRACE sends the reception time in *TRACEAllInOnePacketType* to the synchronization subsystem. The synchronization subsystem calculates the transmission duration of a packet using the packet type along with the selected PHY layer parameters such as the Data transmission rate of the PSDU and the size of the physical header, H_{PHY}, as in Eq. (7.9).

$$T_{TxTime}(PacketType) = H_{PHY} + \frac{\text{Packet Size(Packet Type)}}{\text{Data Rate}} \tag{7.9}$$

Then the start time of the TRACE frame based on the packet, FST_P, is calculated using the recorded reception time, T_{rx}, the guard band before sending packets, GB_{TX}, and the mean processing delay, T_{proc} using Eq. (7.10). T_{proc} is used to compensate for the mean value of the packet delay, $\tilde{D}$.

$$FST_P = T_{rx} - T_{TxTime}(PacketType) - GB_{TX} - T_{proc} \tag{7.10}$$

Finally, the FST_P is compared with the frame start time calculated based on the internal clock, FST. The internal clock is advanced by increasing the offset value of the TRACE timer by Ta, which is calculated as in Eq. (7.11).

$$Ta = FST - FST_P \tag{7.11}$$

Under ideal conditions, the subsequent calculations of Ta are expected to be zero once the node is synchronized to a CH. However, due to clock drifts and the delays at the transmitter and the receiver side, Ta is modeled as a random variable with mean $\mu_\Phi = 0$ as shown in Eq. (7.12).

$$Ta = \tilde{\Phi} \tag{7.12}$$

In order to verify this algorithm, we ran experiments using a single hop connected network of 2 nodes. A debugging option that prints out the adjustment amount and the time of adjustment in a debugging file, *TRACETimerDebugger.txt*, is implemented in TRACETimer and enabled. We analyze a record of 10,000 adjustments on the member node.

Although the data is mostly centered around $\mu_\Phi = 0$, there are a small number of outlier points with large deviation from the mean. These outlier points

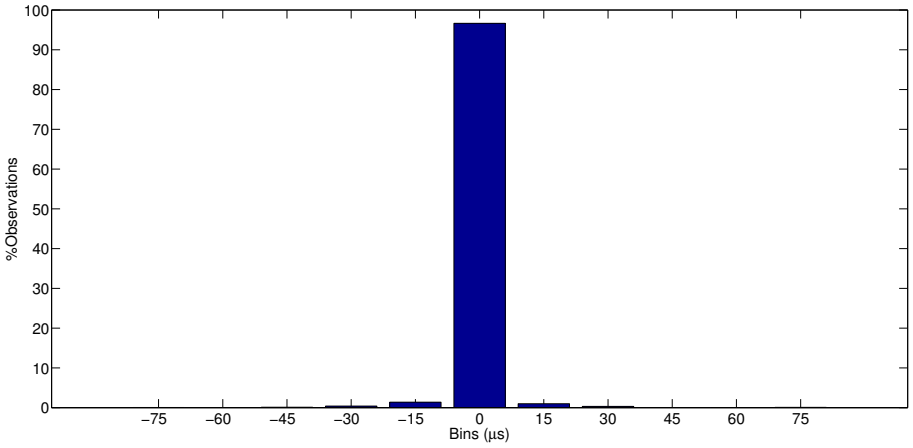


Figure 7.13: Histogram of timer adjustments data for a node pair running the single side synchronization algorithm.

correspond to the time instances in which the receiver system cannot get the required CPU resources due to operating system restrictions. The experiment systems have an IFS allowance of $100\mu s$ evenly distributed before and after the slot times. Hence the packets with $|Ta| > 100\mu s$ leak on an adjacent slot, making their receptions dependent on the state of the TRACE Protocol. Hence, the outliers are defined as the values $|Ta| > 100\mu s$. We eliminate the outlier samples with $Ta_i > 100\ \mu s$ that constitute a negligible 0.08% of the data set. The resulting histogram of the time adjustments with the eliminated data is available in Fig. 7.13.

The data in Fig. 7.13 has a sample mean of $-0.021\mu s = \mu_\Phi \simeq 0\mu s$ and 97% of the data lies within 7.5 μs of μ_Φ. Estimating standard deviation of $\tilde{\Phi}$, σ_Φ, with the sample standard deviation leads to $\sigma_\Phi = 4.730\mu s$. This result is in line with our previous results of Section 7.5.1. The slight increase in the standard deviation, $\sigma_\Phi > \sigma_D/2 = 2.739\mu s$, is due to the additional processing stages of the TRACE system that take a random amount of time to be executed and the frame selection algorithm of the CH of the TRACE system.

A high majority (99%) of the observations lie in a tight interval around the median, Cov_{Φ}, calculated as in Eq. (7.13).

$$Cov_{\Phi} = [0.05\% \text{ percentile of } \tilde{\Phi}, 99.5\% \text{ percentile of } \tilde{\Phi}] = [-26.155\mu s, 19.664\mu s] \tag{7.13}$$

This verifies that the synchronization algorithm works up to a constant value, T_{proc}. The proper value of T_{proc} is optimized in Section 7.5.3.

7.5.3 Multi Hop Synchronization

For a multi-hop network, the TRACE algorithm creates clusters such that the CHs form a dominating set. The nodes that are out of the reach of a CH form a new cluster through the CH arbitration phase of the TRACE algorithm. However, the selected CHs are out of direct communication range of one another, and the beacon packet of a CH can only be received by its immediate neighbors and cannot reach all the nodes in the network. As a result, the *Single Side Synchronization Algorithm* presented in Section 7.5.2 is ineffective in a multi-hop setting.

In order to address this problem, in this section, we present a new algorithm, namely *Dual Side Synchronization Algorithm*, which is an extension of the *Single Side Synchronization Algorithm*. First we will present the algorithm and test its performance in a two-terminal system in a single hop arrangement and then present the experimental results with a network consisting of multiple terminals in a single cluster followed by multiple terminals with multiple clusters.

Since the Beacon packet cannot reach other CHs, the packets transmitted by member nodes should be used for synchronization purposes. The contention packets are selected as dual purpose multi-hop synchronization messages. The utilization of the contention slots is independent of the network load and is very low, as observed in Chapter 3. Hence extra transmissions in the contention slots

will not alter the proper operation of the system and will have minimum impact on the system performance.

At the end of the Beacon slot, nodes pick a random contention slot and determine whether to send a contention packet based on the CH table and the existence of data packets. The decision to send a contention packet is stored in the boolean variable *DoINeedToSendContention*, and the chosen random contention slot is stored in the integer variable *MyContentionSlotNumber*. We have added a new decision for nodes that decide not to send a contention in the current frame. With a probability of p_{dc}, these nodes set a boolean variable *DoINeedToSendDummyContention* and send a dummy contention packet in the chosen *MyContentionSlotNumber*.

In order to prevent dummy contentions making CHs reserve a data packet and to facilitate the synchronization algorithm, *DoINeedToSendDummyContention* and *MyContentionSlotNumber* are embedded into the over the air contention packet. In order to have the correct variables in the contention packet saved on the SORA radio controller board's memory, the packet preparation subsystem is triggered to update the packets by resetting the packet status flags.

Both dummy and regular contention receptions go through the synchronization subsystem. The synchronization subsystem calculates the time to transmit the packet using the packet type along with the selected PHY layer parameters using Eq. (7.9). Using the recorded reception time of the contention packet and *MyContentionSlotNumber* value in the packet, the frame start time based on the packet, FST_P, is calculated as in Eq. (7.14).

$$\begin{aligned} FST_P = & T_{rx} - T_{TxTime}(PacketType) - GB_{TX} - T_{proc} \\ & - (\text{MyContentionSlotNumber} - 1) * ContentionSlotLength \\ & - BeaconSlotLength - CASlotLength \end{aligned} \tag{7.14}$$

Finally, similar to the *Single Side Synchronization Algorithm*, the FST_P is compared with the frame start time calculated based on the internal clock, FST. The internal clock is advanced by increasing the offset value of the TRACE timer by Ta, which is calculated as in Eq. (7.11).

Terminal Pair in Single Hop

In a system with a single transmitter, the selection of T_{proc} is independent of the timer adjustments. On the other hand, in a system with multiple transmitters, T_{proc} must be optimized. In order to set the optimum value of T_{proc}, we perform an experiment with 2 TRACE systems following the dual synchronization algorithm. The debugging option of the TRACETimer that prints out the adjustment amount and the time of adjustment in a debugging file, *TRACETimerDebugger.txt*, is enabled. In order to test the dual mode of synchronization, we set $p_{dc} = 1$ to ensure the node sends SYNC packets for each Beacon reception. We collect 1000 samples of time adjustments, Ta, for each T_{proc} value and observe the sample mean of absolute value of adjustments, $\mu_{|Ta|}$, as T_{proc} is varied in the set $\{-20\mu s, 0\mu s, 20\mu s, 30\mu s, 35\mu s, 40\mu s, 45\mu s\}$. The smallest $\mu_{|Ta|}$ is observed with $T_{proc} = 40\mu s$. Thus, we set $T_{proc} = 40\mu s$ and repeat the experiment, collecting 10,000 sample points of Ta on each device (i.e., the CH and the regular node) for the rest of the analysis.

The experiment systems have an IFS allowance of $100\mu s$ evenly distributed before and after the slot times. Hence, the packets with $|Ta| > 100\mu s$ leak on an adjacent slot, making their receptions dependent on the state of the TRACE Protocol. Hence, the outliers are defined as the values $|Ta| > 100\mu s$ and constitute a negligible 0.10% and 0.20% of the sample set for the node and the CH, respectively. The histogram of the collected Ta samples after the elimination of the outliers is depicted in Fig. 7.15.

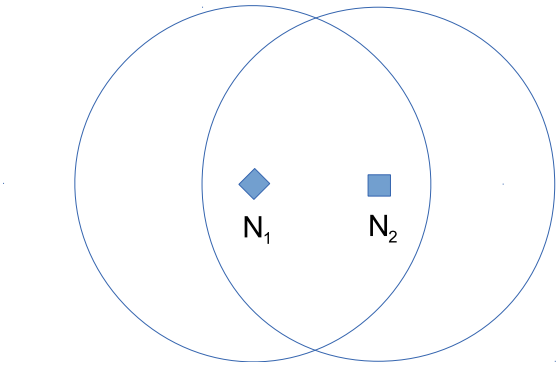


Figure 7.14: The layout of the nodes for the experiment with a pair of nodes in single cluster formation. The node that takes the CH role is represented with a filled diamond, and the member node is represented with a filled square.

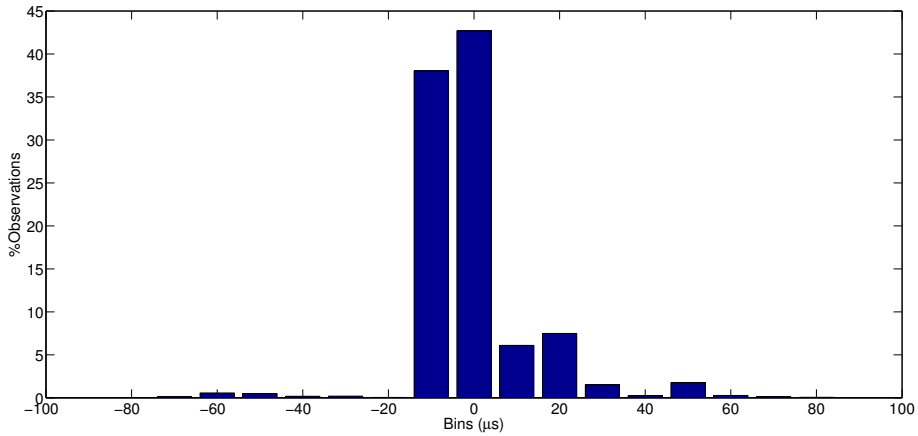
Although the sample sets at both the CH and the member node have similar variations, they are slightly biased with sample means at $\mu_{Ta_n} = -0.594$ and $\mu_{Ta_{CH}} = -0.635$ for the member node and the CH, respectively. This bias is due to a slight mismatch of the T_{proc} value from its optimum. The sample standard deviations are also very similar, $\sigma_{Ta_n} = 9.668\mu s$, $\sigma_{Ta_{CH}} = 11.255\mu s$, for the member node and the CH, respectively.

The standard deviation of Ta for the dual synchronization algorithm is larger than the single synchronization algorithm. This is due to:

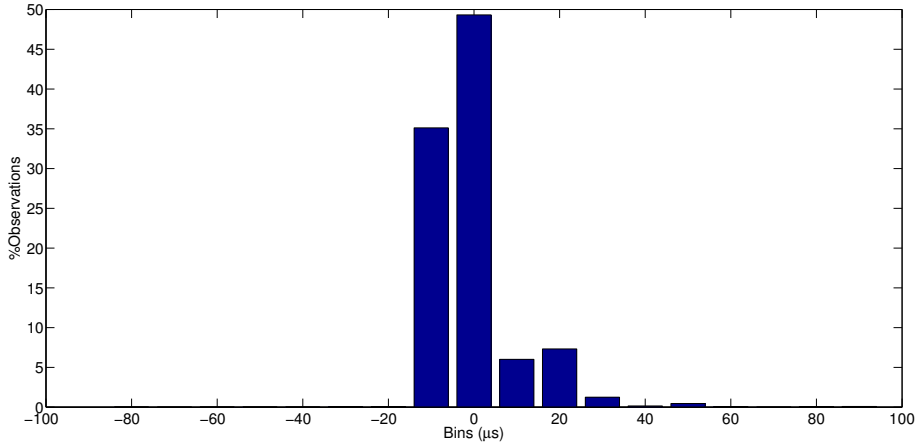
- the increased complexity of the reception and the transmission;
- the variability of the relative clock drifts on the SYNC packets due to the random slot selection; and
- the synchronization loop that increases the variance, making the consecutive samples dependent.

Still, a high majority (99%) of the observations lie in a tight interval around the median, Cov, calculated as in Eq. (7.15).

$$Cov_{Ta} = [0.05\% \text{ percentile of } Ta, 99.5\% \text{ percentile of } Ta] \tag{7.15}$$



(a)



(b)

Figure 7.15: Histogram of timer adjustments data for the dual side synchronization algorithm in an experiment with a node pair in a single hop network (a) for the CH and (b) for the member node.

The calculated intervals for the member node and the CH are $Cov_{Ta_n} = [-8.123\mu s\ ,\ 48.443\mu s]$, and $Cov_{Ta_{CH}} = [-57.821\mu s, 55\mu s]$. The rest of the packets could be considered as lost packets along with other packet losses due to channel noise and cross system interference. Thus, the synchronization system with a properly set IFS value works with reasonable accuracy.

Multiple Terminals with a Single Cluster

In this section, the performance of the dual synchronization algorithm is studied with multiple nodes. In a multi node environment, the nodes are synchronized to the CH, and with a probability $p_s = 0.5$, they send a SYNC packet in a randomly selected contention slot. The CH is synchronized to each sync packet. In the next superframe, the CH sends a new Beacon that in turn is used to synchronize the member nodes.

The experiment setup is depicted in Fig. 7.16. The member nodes on the left and on the right are depicted with filled squares, and the node that takes the CH role is depicted with a filled diamond. The nodes' transmission and reception gains are adjusted so that the CH is able to receive the packets transmitted by the nodes and vice versa. However, the member nodes cannot receive the packets that are transmitted by each other. All the nodes follow the dual synchronization algorithm.

The synchronization algorithm records the amount it is adjusting its internal clock, Ta with time stamps. We take 10,000 sample points for each node. The histogram of Ta samples for the CH (N_2) and node 3 (N_3) are depicted in Fig. 7.17. Note that, since node 1 (N_1) and node 3 (N_3) have similar conditions, we only collect statistics on N_3. The majority of the samples are centered around the origin, and the sample means and standard deviations are: $\mu_{Ta_{CH}} = -1.636\mu s$ and $\sigma_{Ta_{CH}} = 12.793\mu s$ for the CH; and $\mu_{Ta_{node}} = -1.809\mu s$ and $\sigma_{Ta_{node}} = 9.792\mu s$

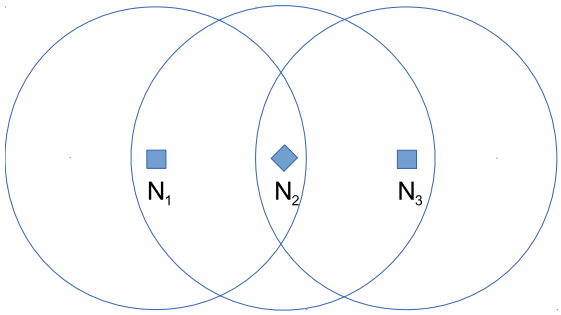


Figure 7.16: The layout of the nodes for the experiment with multiple nodes in single cluster formation. The node that takes the CH role is represented with a filled diamond, and the member nodes are represented with filled squares.

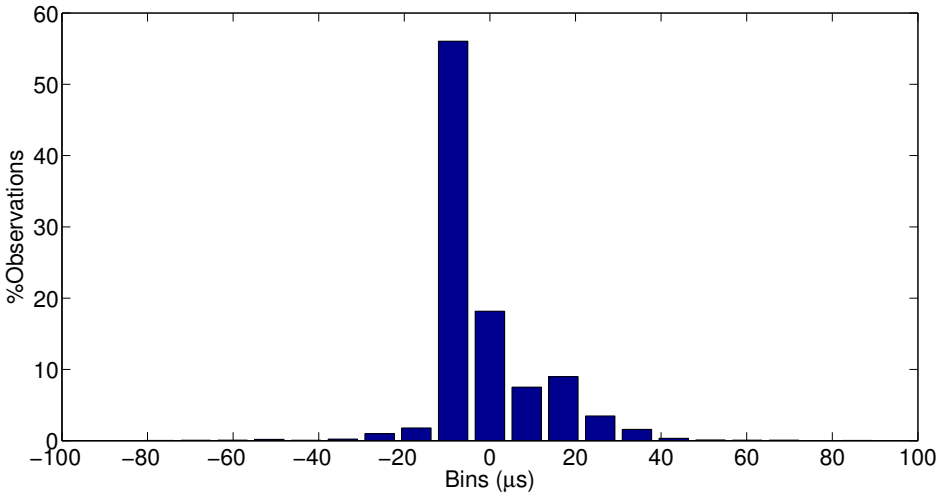
for the member nodes.

These values are similar to each other and the previous results for the experiment with a node pair. Hence, the dual synchronization algorithm works equally well for both the CH and the nodes. Moreover, the number of nodes in a cluster does not decrease the accuracy of the dual synchronization algorithm.

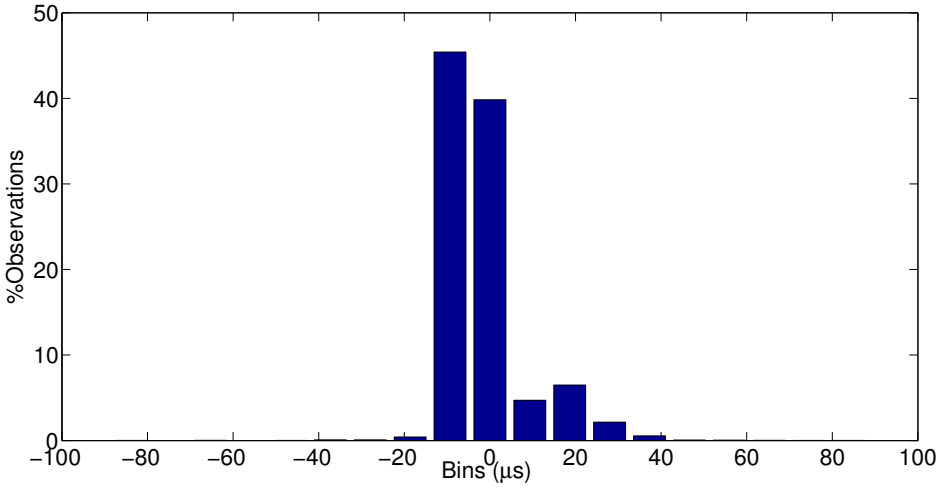
A high majority (99%) of the observations lie in a tight interval around the median $Cov_{Ta} = [-45.221\mu s, 50.223\mu s]$ and $Cov_{node} = [-16.511\mu s, 36.332\mu s]$ for the CH and the member nodes. Following a conservative approach, we keep the interframe spacing of $100\mu s$.

Multiple Terminals with Multiple Clusters

In this section, the performance of the dual synchronization algorithm is studied in a multi cluster setting. In a multi cluster setting, the member nodes are listening to the Beacon and the contention slots of all the frames in the superframe. The CHs listen to the Beacon slots of the frames other than their chosen frames of operation and listen to all the contention slots of all the frames. Any received Beacon and SYNC packet provides synchronization for all the nodes regardless of their role



(a)



(b)

Figure 7.17: Histogram of timer adjustments data for the dual side synchronization algorithm in the experiment with a multiple nodes in a single cluster (a) for the CH (b) for the member node.

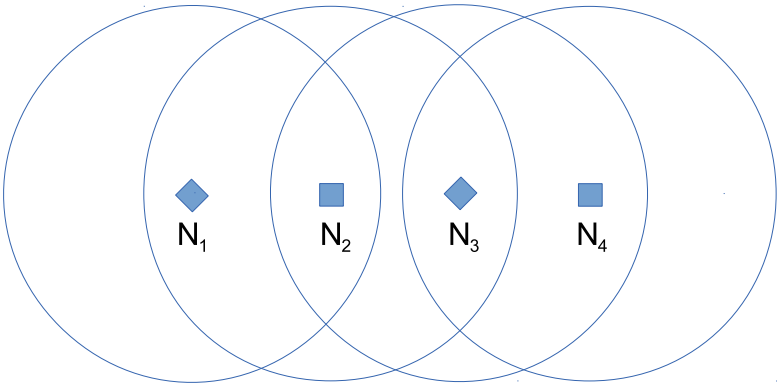


Figure 7.18: The layout of the nodes for the experiment with multiple nodes in multi-cluster formation. The CHs are represented by the diamonds and the member nodes are represented by the squares. The circles around the nodes represent the reception ranges.

in the network. However, since CHs cannot be in the range of one another, they only use SYNC packets for synchronization that distribute the synchronization information from one cluster to another.

The experiment setup is depicted in Fig. 7.18. The member nodes are depicted with filled squares and the CHs are depicted with filled diamonds. The nodes' transmission and reception gains are adjusted so that each node is able to receive only the packets transmitted by the adjacent nodes. We investigate the performance of the dual synchronization algorithm on the node that is in between the two clusters (N_2) and the CH with 2 member nodes (N_3).

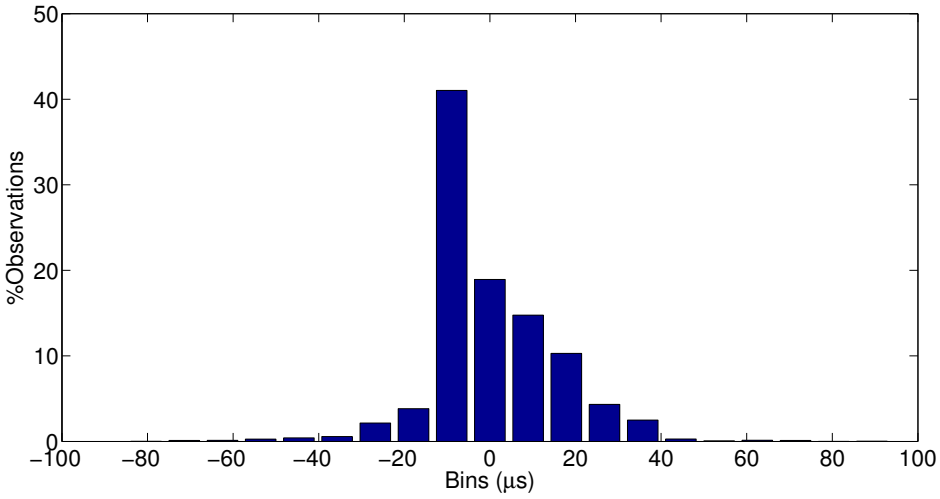
The synchronization algorithm records the adjustments, Ta, with time stamps. We take 10,000 sample points for each node. The histograms of Ta samples for the member node (N_2), and the CH (N_3) are depicted in Fig. 7.19. The majority of the samples are centered around the origin, and the sample means

and standard deviations are: $\mu_{Ta_{CH}} = 0.0094\mu s$ and $\sigma_{Ta_{CH}} = 14.831\mu s$ for the CH; and $\mu_{Ta_{node}} = -0.495\mu s$ and $\sigma_{Ta_{node}} = 15.844\mu s$ for the member node. These values are similar to each other and the previous results for the single cluster experiments. Hence, we conclude that the dual synchronization algorithm works equally well: (1) for both the CHs and the member nodes; and (2) for both a single cluster and a multi cluster network.

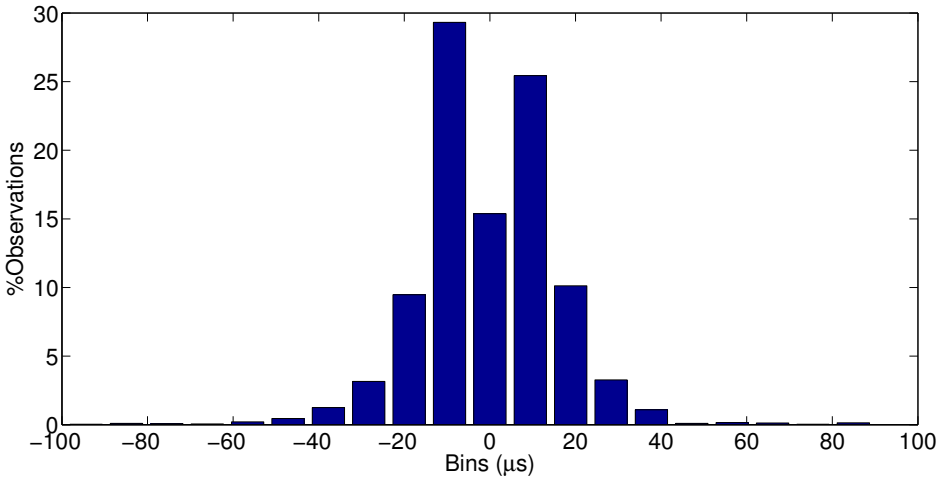
In order to account for the imperfect synchronization and to provide allowance for the radio to switch between transmission and reception modes, we utilize an interframe spacing (IFS) of $100\mu s$. This IFS value is added to the length of each slot. This IFS value is divided between the beginning and the end of the slot by scheduling the actual transmission of the packets $GBBeforeTX = 50\mu s$. A high majority (99%) of the Ta observations lie in a tight interval around the median $Cov_{Ta} = [-49.366\mu s, 42.980\mu s]$ and $Cov_{node} = [-51.327\mu s, 46.564\mu s]$ for the CH and the member node, respectively. Considering Cov_{Ta} and Cov_{node}, we reduced the IFS value to $60\mu s$ and observed a stable operation of the system. The resulting slot lengths are listed in table Table 7.4. The experiments performed in the rest of this chapter use the dual synchronization algorithm and the slot lengths listed in Table 7.4.

7.6 System Performance

The TRACE system has algorithms to minimize the interference from other TRACE nodes that are based on channel power measurements. These algorithms are designed assuming operation on a special frequency band that belongs only to the TRACE system. This dedicated band provides relatively stable channel conditions in which the channel noise is time-independent or has a large coherence time.



(a)



(b)

Figure 7.19: Histogram of timer adjustments data for the dual side synchronization algorithm in the experiment with a multi cluster network (a) for the CH, and (b) for the member node.

Table 7.4: TRACE packet sizes and corresponding slot lengths.

Type	Packet Size (Bytes)	Slot Length (μs)
Beacon	4	202.545
CA	4	202.545
Contention	4	202.545
Header	10	206.909
IS	17	212.000
Data	111	280.364
Frame		3504.364
SuperFrame		21026.182

In addition to this, TRACE is designed to operate on a closed system in which all the nodes in the network cooperate. The operation of the system depends on:

- CH selection and maintenance through Beacon and CA packets,
- contention for channel resources through contention packets,
- transmission schedule determination through Header packets, and
- channel reservation through IS packets.

All of these algorithms suffer from packet losses that, in turn, affect the transmission and the reception of the data payload.

On the other hand, our implementation of TRACE operates on the commonly used $2.4GHz$ band, sharing the channel with commonly used 802.11 systems. 802.11 systems are competition based and do not cooperate with the TRACE system for interference minimization. Hence, the interference from out of the

system devices causes random packet losses in the TRACE system, which degrades the performance.

In order to increase the robustness of the implementation against packet losses, we add redundancy to all of the TRACE algorithms. These algorithms compensating for the missing packets can be summarized as:

Missing Beacon The nodes do not send a contention but keep the CH in the CH list. The CH entries are marked as obsolete and deleted only after not receiving any packets from the CH for $CHOBSOLETETIME = 2\ SF$ durations.

Missing CA The received power level is decreased gradually following the exponential smoothing algorithm with a high smoothing factor. On the other hand, in the case of successfully detected CA, the perceived power level is increased sharply using a small smoothing factor.

Missing Contention In the case of a missing contention packet, the node attempts to access the channel resources by sending a contention in the next available superframe.

Missing Header In the case of a missing header packet, the node follows the existing transmission schedule of the previous frame and deletes the reservation at the end of the frame. Since the reservation is dropped, the node starts contending for channel resources starting from the next frame. The CH ignores the extra contention that it receives from an existing node in its transmission list and sends the header as scheduled. The node notes the slot as reserved with the next Header reception. On the other hand, if subsequent Header packets are also missed, the node does not send an IS, as the reservation has already been deleted.

Missing IS Our implementation distinguishes the missing IS packets and the IS packets indicating the slot is not required. In the case of a missing IS packet, the existing transmission schedule is not immediately altered, but a missing IS counter is incremented. The transmission schedule is only changed after 3 sequential missing IS packets, and any successful IS reception resets the counter. The CH deletes the reservation after 3 missing packets. Nodes listening to the stream also continue listening for the data packets for the upcoming 3 SFs unless another IS packet is received or the transmission schedule of the frame is changed with a Header.

Missing DATA The missing data packets do not affect the TRACE algorithm. However, the data packet IDs are tracked, and the missing packets are marked. This information can used by the application layer for error correction purposes.

Next we test the performance of our TRACE implementation under packet losses. The network topology used in the rest of this section is shown in Fig. 7.14. Although the CH selection algorithm is dynamic, the node that is started first quickly takes the CH role. However, the CH selection is dynamic and follows the TRACE CH selection rules. Packet losses may trigger CH selection and can change the roles of the nodes in the network. The system that is started later is a source node generating packets throughput the experiment. The experiments are continued until the source node transmits 10,000 packets.

Performance of Packet Loss Compensation Systems

Although both of the nodes are in the range of each other, they cannot receive all the packets sent by their neighbors. The success of packet receptions depends on the transmission power, propagation and the channel conditions at the time of reception.

In order to test the performance of the system under packet losses, we keep track of the number of transmitted and the number of received packets on each node for each packet type. Table 7.5 shows the number of transmitted/received packets for a node pair that are separated by 2m and use a transmission power of $12dBm$. The reception gain parameters are set to $rxgain = 16dB$ and $rxpa = 16dB$. It is observed that for a high transmission power, packet loss rate is low. The TRACE system is working as expected with a high data packet delivery rate.

Table 7.5: The number of transmitted/received packets and the packet loss rate (%) for a transmission power of 12dbm for the topology depicted in Fig. 7.14.

Packet	Receptions		Transmissions		Packet Loss (%)	
Type	N_1	N_2	N_1	N_2	N_1	N_2
Beacon	0	10129	10241	0	-	1
CA	0	4828	5148	0	-	6
Contention	871	0	0	944	8	-
Header	0	9362	10173	0	-	8
IS	10006	0	0	10171	2	-
Data	10084	0	0	10171	1	-
SYNC	4225	0	0	4464	5	-

Thanks to the missing beacon compensation algorithm, the clustering structure is very robust. Although there are 112 missing Beacon packets, the member node starts the CH selection algorithm only once and receives a beacon packet before assuming a CH role and transmitting its Beacon packet.

Another point that can be observed is that the number of contention packets sent is low compared to the number of transmitted IS and data packets. After the successful reception of a contention, the CH allocates one of the available data

slots to the contender node and keeps the reservation in subsequent SFs until it is not used anymore by the node.

The TRACE system provides continuous channel resources to the source nodes, which makes TRACE ideal for real time communication. In Table 7.5 it can be observed that the number of transmitted IS and the number of transmitted data packets are very high and are close to the number of transmitted beacon packets. Although, 1.62% of the IS packets are lost at the CH side, the CH keeps the reservations, providing continuous channel access to the node 99.31% of the 286 s experiment duration thanks to the missing IS packet compensation algorithm.

Performance with Varying Transmission Power

Successful packet receptions are strongly correlated with the channel conditions around the receiver at the time of the transmission. The power of the packet should be strong enough to yield a high enough signal to interference ratio. Transmissions with a higher transmission power have a higher chance of receptions.

In order to test the performance of the system, we track the number of all transmitted and received packets on each node as we vary the transmission power. The same experiment setup depicted in Fig. 7.14 is used. Based on this, we calculate the packet loss rate, p_{loss}, that is defined as

$$p_{loss} = \left(1 - \frac{\text{Number of received packets}}{\text{Number of transmitted packets}}\right) * 100 \tag{7.16}$$

We test the performance of the system for a variety of transmission power settings. The nodes are separated by $2m$. The reception gain parameters are kept constant at $rxgain = 16dB$ and $rxpa = 16dB$. For each transmission power setting, at least 10,000 data transmissions are recorded.

Table 7.6 shows the packet loss rate of each packet type together with the average packet loss rate for each transmission power setting. It is observed that

for a high transmission power, packet loss rates are low. The TRACE system is working as expected with a high data packet delivery rate.

Table 7.6: Packet loss rates (%) of each packet type with varying transmission power for the topology depicted in Fig. 7.14.

Packet	12 dBm		10 dBm		6 dBm		2 dBm		0.5 dBm	
Type	N_1	N_2	N_1	N_2	N_1	N_2	N_1	N_2	N_1	N_2
Beacon	-	1	-	3	2	4	-	23	50	30
CA	-	6	-	9	29	11	-	27	59	36
Contention	8	-	9	-	7	-	11	-	18	-
Header	-	8	-	10	8	11	-	35	56	41
IS	2	-	3	-	5	-	24	-	30	-
Data	1	-	3	-	2	-	29	-	32	-
SYNC	5	-	5	-	6	4	8	-	12	84
Average	3		5		6		23		28	

The average packet loss rate increases as the transmission power decreases, as expected.

The missing entries of Table 7.6 correspond to the cases in which no packets of that kind are transmitted. Since node 1 is not a source node, it does not transmit the contention, IS or data packets. For a transmission power of 12 dBm, the clustering structure is preserved throughout the experiment. Since node 2 is never elected as a CH, the beacon CA and header entries of node 1 and SYNC entry of node 2 are omitted. On the other hand, as the transmission power is reduced, the CH role can change between the nodes due to the higher packet loss rate of the Beacon packets.

In addition to the number of transmitted/received packets, we keep track of

the number of times each node goes through the CH selection algorithm. A high packet loss rate in the Beacon packets can potentially destabilize the clustering structure and trigger the CH selection algorithm. However, thanks to the missing beacon compensation algorithm, even for a transmission power of 0.5dBm, the selection of the CHs have changed only 3 times throughout the experiment duration of $450s$, while node 1 went through the CH selection algorithm 2 times and node 2 went through the CH selection algorithm 52 times. In 50 instances out of the 52 times that node 2 went through the CH selection algorithm, node 2 went out of the algorithm by receiving a Beacon packet before it sends its first Beacon and takes over the CH role.

The TRACE system is designed to allocate continuous channel access for voice streams under ideal conditions. The TRACE system has a total of 18 data slots. Each data slot accommodates data packets that can carry up to 100 bytes of a voice stream's payload. Every SF duration of $SFT = 21.026$ ms, one data slot is reserved for each transmitting node. Hence, under ideal conditions, a TRACE system can carry $\frac{100*8}{21.026*10^{-3}} = 38.048$ kbps for each stream. In our experiments, for a 12 dBm transmission power, our implementation leads to an average data transmission rate of 37.78 kbps and an average data reception rate of 37.46 kbps. The experiments clearly show that the TRACE system keeps its overall stability also under unfavorable channel conditions. The TRACE system with a transmission power of 0.5 dBm leads to an average data transmission rate of 25.12 kbps and average data reception rate of 18.01 kbps.

7.7 Summary

In this chapter, the details of the implementation of the TRACE protocol are presented. This is the first implementation of the TRACE protocol on real hardware. Our implementation solves various problems that are not fully reflected in sim-

ulation studies, such as synchronization, unpredictable interference patterns and limited computational and memory resources. The working prototype proves the concept of the TRACE protocol for practical wireless communication in MANETs.

In addition to this, we also presented two synchronization algorithms that are based on packet receptions. The unpredictability of packet delays that occur due to the limitations of working on real time hardware is analyzed at each stage. The algorithms are used within the TRACE implementation to provide synchronization services. The algorithms achieve a synchronization accuracy of $60\mu s$ for a multi-hop and multi-cluster network. This synchronization accuracy is sufficient for the operation of the TRACE system.

Finally, we have tested the performance of TRACE under packet losses. The TRACE system is shown to maintain its stability and provide uninterrupted services to the source nodes in the system, even under unfavorable channel conditions.

8 Conclusions and Future Work

8.1 Conclusions

In this dissertation, we have explored the efficient use of resources in MANETs at the MAC and the routing layers. The cooperation and information sharing between the nodes both in the MAC and Network layers of the protocol stack leads to more efficient use of resources in MANETs compared to competition based architectures. The techniques described in this dissertation lead to substantial savings in energy consumption and spectrum usage. The contributions of the research are summarized below.

1. We proposed a mathematical model that estimates the performance of soft-clustering MAC protocols. This model is used to estimate the performance of the MH-TRACE protocol under various conditions and for various sets of protocol parameters. The number of frames per superframe, which is the protocol parameter that determines the degree of spatial reuse, is optimized for various conditions and for both spectral efficiency and energy efficiency.

2. We proposed CDCA-TRACE, a MAC protocol for MANETs that enables cooperative load balancing, and dynamic channel allocation through spectrum

sensing. CDCA-TRACE is built on top of MH-TRACE and replaces the basic MAC functionality in other TRACE protocols. By allowing dynamic allocation of channels among clusterheads, CDCA-TRACE outperforms MH-TRACE in handling non-uniform traffic loads. Unlike MH-TRACE, CDCA-TRACE also dynamically adapts to shrinking network sizes and can effectively utilize the entire superframe duration even for single hop connected networks.

3. We proposed U-TRACE, a cross-layer protocol for MANETs that chooses the most efficient data dissemination scheme according to the network conditions. U-TRACE is based on CDCA-TRACE and includes the dynamic channel allocation features of CDCA-TRACE. It further implements multicasting and network-wide broadcasting services and includes a mechanism that chooses the most efficient data dissemination scheme.

4. We showed the benefits of network symbiosis at the network layer. The differences in the characteristics of various networks, such as node density and communication radius, are exploited through hybrid nodes in decreasing the hop count or energy consumption for the communication between two nodes. A mathematical model that estimates the length of the shortest path is constructed and used to extend the analysis of hybrid nodes to various node densities and communication radii.

5. We showed the practicality of the solutions investigated in this thesis by implementing U-TRACE on software defined radios. We investigated practical issues encountered in a real implementation of radio protocols, such as synchronization and cross band interference, and provided solutions to these problems. We showed that our implementation compensates packet losses and performs successfully even with very low transmission powers in the presence of cross-band interference.

8.2 Future Work

The demand for bandwidth intensive applications increases the importance of bandwidth management in MANETs. To meet this growing need, additional research is needed as follows:

- The TRACE family of protocols is designed as TDMA-based protocols. The underlying waveform is designed to work on the entire frequency band available. Using a large frequency band, the packets are transmitted faster, which contributes to a low latency. However, due to the limitations on the hardware, the efficiency decreases as the size of the frequency band increases. The TRACE implementation can be extended to work on multiple frequency bands. By keeping the signaling portion of the frame in a common band and moving the data portion to assigned bands, it is possible to extend the operation to work on multiple channels.

- In the TRACE protocol, nodes cooperate with the other nodes in their local neighborhood. This cooperation can allow implementation of cooperative concurrent transmissions to take advantage of the constructive interference. Using spatial diversity similar to MIMO systems, it is possible to increase the range and the reliability of the communication between nodes.

- Distributed computing is another application of increasing importance for mobile ad hoc networks. CPU intensive computing jobs are split into multiple independent tasks and are distributed to multiple clients. The cooperation among the nodes is crucial for distributing the tasks among the nodes. The TRACE communication infrastructure can be used to distribute the tasks efficiently. Depending on the requirements of the job, the clusterheads can take the responsibility for distributing the tasks to the nodes within their clusters and requesting support from other clusters.

Appendices

A TRACE Timer

```
1  #include <windows.h>
2  #include <iostream>
3
4  #define ABRUBTCHANGE (40.0E-06)
5  #define ABRUBTCHANGELIMIT (3)
6
7  struct reference_point {
8  double offset;
9  LARGE_INTEGER counter;
10 bool sync_ready;
11 };
12
13 class TRACETimer {
14 private:
15   reference_point ref_point;
16   LARGE_INTEGER frequency;
17
18   //Dummy Variables
19   LARGE_INTEGER li;
```

```
20
21    //Monitoring
22    FILE *foutputp;
23
24    int NumofAbrubtChange;
25
26  public:
27    TRACETimer() {
28      ref_point.sync_ready=false;
29      QueryPerformanceFrequency(&frequency);
30      QueryPerformanceCounter(&ref_point.counter);
31      ref_point.offset = 0.0;
32      NumofAbrubtChange = 0;
33      ref_point.sync_ready=true;
34      char foutputn [] = "TRACETimerAdjustments.txt" ;
35      if ((foutputp=fopen(foutputn,"w"))==NULL) {
36        printf("I can't open the file for writing Receptions\n↩
            ");
37        exit(1);
38      }
39    };
40
41    //GetCurrentSeconds: Calculates the time and passes to ↩
        TRACEMANAGER
42    void GetCurrentSeconds(TRACETimeType &NOW) {
43      ::QueryPerformanceCounter(&li);
44      NOW = ( ( ((double)(li.QuadPart - ref_point.counter.↩
          QuadPart))/(double)frequency.QuadPart) );
45      NOW = NOW + ref_point.offset;
46    };
```

```
47
48    // SimplisticSyncronize: The is used to change the offset value.
49    void SimplisticSyncronize(const TRACETimeType &ooffsetin) {
50      if(ooffsetin>ABRUBTCHANGE) {
51        NumofAbrubtChange++;
52        if(NumofAbrubtChange > ABRUBTCHANGELIMIT) {
53          NumofAbrubtChange = 0;
54          ref_point.offset -= ooffsetin;
55          QueryPerformanceCounter(&li);
56          TRACETimeType NOW = ( ( ((double)(li.QuadPart - ref_point.counter.QuadPart))/(double)frequency.QuadPart) );
57          NOW = NOW + ref_point.offset;
58          fprintf(foutputp,
59          "TimeAdjust: NOW = %.9f \t ooffsetin = %.3f *E-06 \t ref_point.offset = %.9f \n"
60          , NOW, ooffsetin*1.0E06, ref_point.offset
61          );
62        }
63
64      }
65      else {
66        ref_point.offset -= ooffsetin;
67        QueryPerformanceCounter(&li);
68        double NOW = (  ( ((double)(li.QuadPart - ref_point.counter.QuadPart))/(double)frequency.QuadPart) );
69        NOW = NOW + ref_point.offset;
70        fprintf(foutputp,
```

```
71          "TimeAdjust: NOW = %.9f \t ooffsetin = %.3f *E-06 \t ↩
              ref_point.offset = %.9f \n"
72          , NOW, ooffsetin*1.0E06, ref_point.offset
73          );
74        }
75      };
76  };
```

B TRACE Packet Types

```
1  #include <stdio.h>
2  #include <iostream>
3  #include <string>
4
5
6
7  #include "tracetypes.h" //BK:This has to be the last ↩
     definition line
8
9  #define NumberofPacketTypes (7)
10 enum TRACEPACKETType_Enum {
11   BEACONPACKETSeq = 0,
12   CAPACKETSeq = 1 ,
13   CONTENTIONPACKETSeq = 2 ,
14   HEADERPACKETSeq = 3 ,
15   ISPACKETSeq = 4,
16   DATAPACKETSeq = 5,
17   SYNCPACKETSeq = 6
18 };
```

```
19
20 class TRACEAllInOnePacketType {
21 public:
22   int packet_type;
23   TRACENodeIDType MAC_src;
24     int seq_number;
25   TRACENodeIDType TrSchSrcPrio_[NDS_];
26   int ActiveDataSlots_;
27   int Start_Up_Flag;
28   TRACENodeIDType NextController_;  //Controller that this ↩
       packet is addressed to
29   TRACENodeIDType LastHop_;
30   TRACENodeIDType NextHop_;
31   int MoreData_;
32   int NoData_;
33   int IsNBTRACEPacket_;
34   int IsMCTRACEPacket_;
35   int IsMHTRACEPacket_;
36   int MCastMemberNode_;   //Indicate if the packet sender is↩
        a mcast member of MC trace
37   int MCastRelayNode_;    // Indicate if the packet sender ↩
       is a mcast relay of MC trace
38   int AssociatedDataSourceID ;
39   int AssociatedDataPacketID ;
40   int SearchForData;
41
42   int lengthofpayload;
43   char payload[PAYLOADSIZE];
44   //Beacon+NCAS_+NCS_+Header+IS+Data
45   int ContSlotNumberinPckt_;
```

```
46    int DummyCont_Flag;
47    int HDTS_;
48    int mcastID_;
49
50
51    TRACEAllInOnePacketType() {
52      packet_type = 99;
53      MAC_src = 5;
54          seq_number = 12;
55      for (int i = 0; i < PAYLOADSIZE; i++){
56        payload[i] = 'A';
57      }
58      MCastMemberNode_ = 0;
59      MCastRelayNode_ = 0;
60    };
61
62    void PrintMembers() {
63      printf("**********\n");
64      printf("packet_type is %d ",packet_type);
65      printf("MAC_src is %d ",MAC_src);
66      printf("seq_number is %d ",seq_number);
67      printf("payload is %s ",payload);
68      printf("**********\n");
69    };
70
71    void PrintMembers2File() {
72      FILE *foutputp;
73      char foutputn [] = "RXPacketContents.txt" ;
74      if ((foutputp=fopen(foutputn,"a"))==NULL) {
```

```
75          printf("I can't open the file for writing Receptions\n↩
              ");
76          exit(1);
77        }
78        fprintf(foutputp, "******\n");
79        fprintf(foutputp, "packet_type=%d ",packet_type);
80        fprintf(foutputp, "MAC_src=%d ",MAC_src);
81        fprintf(foutputp, "seq_number=%d ",seq_number);
82        for (int i = 0; i < NDS_; i++){
83          fprintf(foutputp, "TrSchSrcPrio_[%d]=%d ",i,↩
              TrSchSrcPrio_[i]);
84        }
85        fprintf(foutputp, "ActiveDataSlots_ = %d ",↩
            ActiveDataSlots_);
86        fprintf(foutputp, "NextController_ = %d ",↩
            NextController_);
87        fprintf(foutputp, "LastHop_ = %d ",LastHop_);
88        fprintf(foutputp, "NextHop_ = %d ",NextHop_);
89        fprintf(foutputp, "MoreData_ = %d ",MoreData_);
90        fprintf(foutputp, "IsNBTRACEPacket_ = %d ",↩
            IsNBTRACEPacket_);
91        fprintf(foutputp, "AssociatedDataSourceID = %d ",↩
            AssociatedDataSourceID);
92        fprintf(foutputp, "AssociatedDataPacketID = %d ",↩
            AssociatedDataPacketID);
93        fprintf(foutputp, "lengthofpayload = %d ",↩
            lengthofpayload);
94        fprintf(foutputp, "payload is ");
95        for(int i=0; i<lengthofpayload; i++) {
96          fprintf(foutputp, "%c",payload[i]);
```

```
97        }
98
99        fprintf(foutputp, "\n******\n");
100       fclose (foutputp);
101     };
102
103     int GetAssociatedDataSourceID() {
104       return(AssociatedDataSourceID);
105     };
106     int GetAssociatedDataPacketID() {
107       return(AssociatedDataPacketID);
108     };
109     int GetMCastID() {
110       return(mcastID_);
111     };
112   };
113
114   class TRACEMACPacketType {
115   public:
116     TRACEAllInOnePacketType *RxPckt;
117     TRACETimeType TimeStamp;
118     TRACETimeType TimeStamp_detection;
119     TRACEPowerType PacketEnergy;
120     TRACEPacketIdentifierType packet_type;
121     bool isSuccessfullyRx;
122     TRACEMACPacketType(){
123       packet_type = -99;
124       TimeStamp = -99.0;
125       isSuccessfullyRx = true;
126     };
```

```
127    void PrintMembers2File() {
128      FILE *foutputp;
129      char foutputn [] = "RXPacketContents.txt" ;
130      if ((foutputp=fopen(foutputn,"a"))==NULL) {
131        printf("I can't open the file for writing Receptions\n↩
           ");
132        exit(1);
133      }
134      fprintf(foutputp, "*************************\n");
135      fprintf(foutputp, "TimeStamp = %.7f ",TimeStamp);
136      fprintf(foutputp, "packet_type = %d ",packet_type);
137      fclose (foutputp);
138      RxPckt->PrintMembers2File();
139    };
140  };
141
142  /********* Over the Air Packet structures  **********/
143
144
145  struct Beacon_OA {
146    unsigned char contents[BEACONPACKETSIZE];
147    // Flags-------------------------------------------
148    //contents[0]          //Reserved for encryption   8 bits
149    //contents[1]          //Mac_Frame_Type        3 bits    ↩
       (7,6,5)
150                  //Start_Up_Flag          1 bit flag  (4)
151                  //No_of_occupied_data_slots   3 bits    ↩
                       (3,2,1)
152    //contents[2]          //seq_number          8 bits
153    //contents[3]          //MAC_src
```

```
154    //contents[4]
155
156    Beacon_OA() {
157      for (int i=0;i<BEACONPACKETSIZE;i++) {
158        contents[i]= 0x00;
159      }
160      //set_Mac_Frame_Type();
161    };
162    void set_Mac_Frame_Type(){
163      contents[0] = PACKET_TYPE_Beacon<<5 ;
164    };
165    void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
       &tp) {
166      //set_Mac_Frame_Type();
167      contents[0] = char( (tp.packet_type   & 0x0007) << 5);
168      contents[0] = contents[0] | ( (tp.Start_Up_Flag   & 0x01↩
         ) << 4);
169      contents[0] = contents[0] | ( (tp.ActiveDataSlots_ & 0↩
         x07) << 1);
170
171      contents[1] = tp.seq_number & 0xFF;
172
173      contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); ↩
         // High 8 bits
174      contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); ↩
         // Low 8 bits
175
176    };
177    void ConvertToPacketType2(TRACEAllInOnePacketType * tp) {
```

```
178       tp->packet_type     = ( (int(contents[0]) >> 5) & 0x07 )↩
            ;
179       tp->Start_Up_Flag   = ( (int(contents[0]) >> 4) & 0x01 )↩
            ;
180       tp->ActiveDataSlots_  = ( (int(contents[0]) >> 1) & 0x07↩
             );
181
182       tp->seq_number      = ( (int(contents[1]))     & 0xFF   )↩
            ;
183
184       tp->MAC_src       = TRACENodeIDType( (int(contents[2]) ↩
            << 8) |  contents[3] );
185
186     };
187     void PrintMembers2File(FILE * foutputp2) {
188       int dummy = 0;
189       fprintf(foutputp2, "packettype = %d, length = %d, GP ↩
            CONTENTS: ", PACKET_TYPE_Beacon, BEACONPACKETSIZE);
190       for (int i=0; i<BEACONPACKETSIZE; i++) {
191         dummy=0;
192         memcpy(&dummy, &(contents[i]), 1);
193         fprintf(foutputp2, " [%d]%02X", i, dummy );
194       }
195       fprintf(foutputp2, " \n");
196     };
197   };
198
199   struct CA_OA {
200     unsigned char contents[CAPACKETSIZE];
201     // Flags----------------------------------------------
```

```
202
203     //contents[0]          //Mac_Frame_Type        3 bits     ↩
          (7,6,5)
204                          //Start_Up_Flag         1 bit flag  (4)
205                          //No_of_occupied_data_slots   3 bits     ↩
                               (3,2,1)
206                          //Not used            9 bits     (0)
207     //contents[1]          //seq_number          8 bits
208     //contents[3]          //MAC_src
209     //contents[4]
210
211     CA_OA() {
212       for (int i=0;i<CAPACKETSIZE;i++) {
213         contents[i]= 0x00;
214       }
215       //set_Mac_Frame_Type();
216     };
217
218     void set_Mac_Frame_Type(){
219       contents[0] = PACKET_TYPE_CA<<5 ;
220     };
221
222     void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
          &tp) {
223       contents[0] = ( (tp.packet_type   & 0x0007) << 5);
224       contents[0] = contents[0] | ( (tp.Start_Up_Flag   & 0x01↩
            ) << 4);
225       contents[0] = contents[0] | ( (tp.ActiveDataSlots_ & 0↩
            x07) << 1);
226
```

```
227       contents[1] = tp.seq_number & 0xFF;
228
229       contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); // High 8 bits
230       contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); // Low 8 bits
231
232     };
233     void ConvertToPacketType2(TRACEAllInOnePacketType *tp) {
234       tp->packet_type   = (contents[0] >> 5) & 0x0007;
235       tp->Start_Up_Flag = (contents[0] >> 4) & 0x0001;
236       tp->ActiveDataSlots_  = (contents[0] >> 1) & 0x0007;
237
238       tp->seq_number    =  contents[1]     & 0x00FF;
239
240       tp->MAC_src     =   (contents[2] << 8) |  contents[3] ;
241
242     };
243
244   };
245
246   struct Contention_OA {
247     unsigned char contents[CONTENTIONPACKETSIZE];
248     // Flags-----------------------------------------------------
249
250     //contents[1]          //Mac_Frame_Type        3 bits    (7,6,5)
251                     //Start_Up_Flag          1 bit flag  (4)
252                     //No_of_occupied_data_slots   3 bits    (3,2,1)
```

```
253                     //Not used              9 bits     (0)
254     //contents[2]           //seq_number            8 bits
255     //contents[3]           //MAC_src
256     //contents[4]
257
258     Contention_OA() {
259       for (int i=0;i<CONTENTIONPACKETSIZE;i++) {
260         contents[i]= 0x00;
261       }
262       //set_Mac_Frame_Type();
263     };
264
265     void set_Mac_Frame_Type(){
266       contents[0] = PACKET_TYPE_Contention<<5 ;
267     };
268
269     void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
        &tp) {
270       contents[0] = ( (tp.packet_type   & 0x0007) << 5);
271       contents[0] = contents[0] | ( (tp.DummyCont_Flag   & 0↩
          x01) << 4);
272       contents[0] = contents[0] | ( (tp.ContSlotNumberinPckt_ ↩
          & 0x07) << 1);
273
274       contents[1] = tp.seq_number & 0xFF;
275
276       contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); ↩
          // High 8 bits
277       contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); ↩
          // Low 8 bits
```

```
278
279     };
280     void ConvertToPacketType2(TRACEAllInOnePacketType * tp) {
281       tp->packet_type    = (contents[0] >> 5) & 0x0007;
282       tp->DummyCont_Flag  = (contents[0] >> 4) & 0x0001;
283       tp->ContSlotNumberinPckt_ = (contents[0] >> 1) & 0x0007;
284
285       tp->seq_number     =  contents[1]     & 0x00FF;
286
287       tp->MAC_src      =   (contents[2] << 8) |  contents[3] ;
288
289     };
290   };
291
292   struct Header_OA {
293     unsigned char contents[HEADERPACKETSIZE];
294     // Flags-------------------------------------------------
295
296     //contents[1]          //Mac_Frame_Type        3 bits    ↩
        (7,6,5)
297                    //Start_Up_Flag          1 bit flag  (4)
298                    //No_of_occupied_data_slots   3 bits    ↩
                         (3,2,1)
299                    //Not used             9 bits    (0)
300     //contents[2]          //seq_number          8 bits
301     //contents[3]          //MAC_src
302     //contents[4]
303
304     Header_OA() {
305       for (int i=0;i<HEADERPACKETSIZE;i++) {
```

```
306         contents[i]= 0x00;
307       }
308       //set_Mac_Frame_Type();
309     };
310
311     void set_Mac_Frame_Type(){
312       contents[0] = PACKET_TYPE_Header<<5 ;
313     };
314
315     void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
        &tp) {
316       contents[0] = ( (tp.packet_type   & 0x0007) << 5);
317       contents[1] = contents[1] | ( (tp.Start_Up_Flag   & 0x01↩
          ) << 4);
318       contents[1] = contents[1] | ( (tp.ActiveDataSlots_ & 0↩
          x07) << 1);
319
320       contents[1] = tp.seq_number & 0xFF;
321
322       contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); ↩
          // High 8 bits
323       contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); ↩
          // Low 8 bits
324
325       for(int i=0; i<NDS_; i++){
326         contents[4+2*i] = contents[4+2*i] | ( tp.TrSchSrcPrio_↩
            [i] >> 8  & 0xFF); // High 8 bits
327         contents[5+2*i] = contents[5+2*i] | ( tp.TrSchSrcPrio_↩
            [i]       & 0xFF); // Low 8 bits
328       }
```

```
329
330     };
331     void ConvertToPacketType2(TRACEAllInOnePacketType * tp) {
332       tp->packet_type    = (contents[0] >> 5) & 0x0007;
333       tp->Start_Up_Flag = (contents[0] >> 4) & 0x0001;
334       tp->ActiveDataSlots_  = (contents[0] >> 1) & 0x0007;
335
336       tp->seq_number     =  contents[1]     & 0x00FF;
337
338       tp->MAC_src      =   (contents[2] << 8) |  contents[3] ;
339
340
341       for(int i=0; i<NDS_; i++){
342
343         tp->TrSchSrcPrio_[i]      =   (contents[4+2*i] << 8) |↩
              contents[5+2*i] ;
344         if(  tp->TrSchSrcPrio_[i] > 32768 )  tp->TrSchSrcPrio_↩
            [i] = tp->TrSchSrcPrio_[i] - 65536;
345       }
346     };
347   };
348
349   struct IS_OA {
350     unsigned char contents[ISPACKETSIZE];
351     // Flags-------------------------------------------------
352
353     //contents[0]          //Mac_Frame_Type        3 bits    ↩
        (7,6,5)
354                     //MoreData_          1 bit flag  (4)
355                     //Service type        3 bits      (3,2,1)
```

```
356                    //IsNBTRACEPacket_       100    (3)
357                    //IsMCTRACEPacket_       010    (2)
358                    //IsMHTRACEPacket_       001    (1)
359
360
361
362    //contents[1]          //seq_number          8 bits
363    //contents[2]          //MAC_src            16 bits
364    //contents[3]
365    //contents[4]            //AssociatedDataSourceID     16 ↩
         bits
366    //contents[5]
367    //contents[6]            //AssociatedDataPacketID     16 ↩
         bits
368    //contents[7]
369    //contents[8]            //NextController         16 bits
370    //contents[9]
371    //contents [10]          //HDTS               8 bits
372    //comtents[11]           //mcastID            8 bits
373    //contents [12]          //LastHop_             16 bits
374    //contents [13]
375    //contents [14]          //NextHop_             16 bits
376    //contents [15]
377    //contents [16]          //Search FOr Data          1 bit    ↩
         (0)
378                   //MCastMemberNode_          1 bits     (2)
379                   //MCastRelayNode_          1 bits     (1)
380
381
382    IS_OA() {
```

```
383       for (int i=0;i<ISPACKETSIZE;i++) {
384         contents[i]= 0x00;
385       }
386       //set_Mac_Frame_Type();
387     };
388
389     void set_Mac_Frame_Type(){
390       contents[0] = PACKET_TYPE_IS<<5 ;
391     };
392
393     void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
        &tp) {
394       contents[0] = 0;
395       contents[0] = ( (tp.packet_type   & 0x0007) << 5);
396       contents[0] = contents[0] | ( (tp.MoreData_   & 0x01) <<↩
          4);
397
398       contents[0] = contents[0] | ( (tp.IsNBTRACEPacket_   & 0↩
          x01) << 3);
399       contents[0] = contents[0] | ( (tp.IsMCTRACEPacket_   & 0↩
          x01) << 2);
400       contents[0] = contents[0] | ( (tp.IsMHTRACEPacket_   & 0↩
          x01) << 1); //let MHTRACE be the default
401
402
403       //contents[1] = contents[1] | ( (tp.Start_Up_Flag   & 0↩
          x01) << 4);
404       //contents[1] = contents[1] | ( (tp.ActiveDataSlots_ & 0↩
          x07) << 1);
405
```

```
406     contents[1] = tp.seq_number & 0xFF;
407
408     contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); // High 8 bits
409     contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); // Low 8 bits
410
411     contents[4] = contents[4] | ( tp.AssociatedDataSourceID >> 8  & 0xFF); // High 8 bits
412     contents[5] = contents[5] | ( tp.AssociatedDataSourceID
                & 0xFF); // Low 8 bits
413
414     contents[6] = contents[6] | ( tp.AssociatedDataPacketID >> 8  & 0xFF); // High 8 bits
415     contents[7] = contents[7] | ( tp.AssociatedDataPacketID
                & 0xFF); // Low 8 bits
416
417     contents[8] = contents[8] | ( tp.NextController_ >> 8  &
           0xFF); // High 8 bits
418     contents[9] = contents[9] | ( tp.NextController_       &
           0xFF); // Low 8 bits
419     contents[10] = tp.HDTS_ & 0xFF;
420     contents[11] = tp.mcastID_ & 0xFF;
421
422     contents[12] = contents[6] | ( tp.LastHop_ >> 8  & 0xFF);
        // High 8 bits
423     contents[13] = contents[7] | ( tp.LastHop_       & 0xFF);
        // Low 8 bits
424
```

```
425       contents[14] = contents[6] | ( tp.NextHop_ >> 8  & 0xFF)↩
            ; // High 8 bits
426       contents[15] = contents[7] | ( tp.NextHop_       & 0xFF)↩
            ; // Low 8 bits
427
428       contents[16] = 0;
429       contents[16] = contents[16] | ( (tp.SearchForData  & 0↩
            x01)     );
430
431       contents[16] = contents[16] | ( (tp.MCastMemberNode_ & 0↩
            x01) << 2);
432       contents[16] = contents[16] | ( (tp.MCastRelayNode_  & 0↩
            x01)  << 1 );
433
434     };
435
436     void ConvertToPacketType2(TRACEAllInOnePacketType * tp) {
437       tp->packet_type    = (contents[0] >> 5) & 0x0007;
438       tp->MoreData_    = (contents[0] >> 4) & 0x0001;
439
440       tp->IsNBTRACEPacket_     = (contents[0] >> 3) & 0x0001;
441       tp->IsMCTRACEPacket_     = (contents[0] >> 2) & 0x0001;
442       tp->IsMHTRACEPacket_     = (contents[0] >> 1) & 0x0001;
443
444
445
446
447
448       tp->seq_number    =  contents[1]     & 0x00FF;
449
```

```
450     tp->MAC_src     =   (contents[2] << 8) |  contents[3] ;
451
452     tp->AssociatedDataSourceID      =   (contents[4] << 8) |↩
            contents[5] ;
453
454     tp->AssociatedDataPacketID      =   (contents[6] << 8) |↩
            contents[7] ;
455
456     tp->NextController_     =   (contents[8] << 8) |  ↩
          contents[9] ;
457
458     tp->HDTS_ = 0;
459     tp->HDTS_ = contents[10];
460     tp->mcastID_ = 0;
461     tp->mcastID_ = contents[11];
462
463     tp->LastHop_      =   (contents[12] << 8) | contents[13]↩
            ;
464     tp->NextHop_      =   (contents[14] << 8) | contents[15]↩
            ;
465
466     tp->SearchForData = (contents[16]  ) & 0x0001;
467
468     tp->MCastMemberNode_     = (contents[1] >>2 ) & 0x0001;
469     tp->MCastRelayNode_      = (contents[16] >>1    ) & 0↩
          x0001;
470   };
471
472 };
473
```

```
474 struct Data_OA {
475   unsigned char contents[DATAPACKETSIZE];
476   // Flags-----------------------------------------------------
477
478   //contents[0]          //Mac_Frame_Type         3 bits     ↩
        (7,6,5)
479                   //Not used     5 bits     (5,4,3,2,1)
480
481   //contents[1]          //seq_number           8 bits
482   //contents[2]          //MAC_src             16 bits
483   //contents[3]
484   //contents[4]            //AssociatedDataSourceID     16 ↩
        bits
485   //contents[5]
486   //contents[6]            //AssociatedDataPacketID     16 ↩
        bits
487   //contents[7]
488   //contents[8]            //lengthofpayload        8 bits
489   //contents[9]          //PAYLOAD             DATAPACKETSIZE ↩
        bits
490   // ...
491   //
492   //
493   //
494
495   Data_OA() {
496     for (int i=0;i<DATAPACKETSIZE;i++) {
497       contents[i]= 0x00;
498     }
499   };
```

```
500
501     void set_Mac_Frame_Type(){
502       contents[0] = PACKET_TYPE_DATA<<5 ;
503     };
504
505     void ConvertFromPacketType2(const TRACEAllInOnePacketType ↩
          &tp) {
506       contents[0] = ( (tp.packet_type   & 0x0007) << 5);
507
508       //contents[1] = contents[1] | ( (tp.Start_Up_Flag   & 0↩
            x01) << 4);
509       //contents[1] = contents[1] | ( (tp.ActiveDataSlots_ & 0↩
            x07) << 1);
510
511       contents[1] = tp.seq_number & 0xFF;
512
513       contents[2] = contents[2] | ( tp.MAC_src >> 8  & 0xFF); ↩
            // High 8 bits
514       contents[3] = contents[3] | ( tp.MAC_src       & 0xFF); ↩
            // Low 8 bits
515
516       contents[4] = contents[4] | ( tp.AssociatedDataSourceID ↩
            >> 8  & 0xFF); // High 8 bits
517       contents[5] = contents[5] | ( tp.AssociatedDataSourceID ↩
                  & 0xFF); // Low 8 bits
518
519       contents[6] = contents[6] | ( tp.AssociatedDataPacketID ↩
            >> 8  & 0xFF); // High 8 bits
520       contents[7] = contents[7] | ( tp.AssociatedDataPacketID ↩
                  & 0xFF); // Low 8 bits
```

```
521
522
523       if(tp.lengthofpayload>PAYLOADSIZE) {
524         printf("ERROR!! Larger than expected data packet");
525       }
526       contents[8] = tp.lengthofpayload & 0xFF;
527
528       contents[9] = 0;
529
530       contents[10] = tp.HDTS_ & 0xFF;
531       contents[11] = tp.mcastID_ & 0xFF;
532
533       int ul =  tp.lengthofpayload;
534       if(ul>DATAPACKETSIZE - 9 ) ul = DATAPACKETSIZE - 9;
535       memcpy( contents+9, tp.payload, tp.lengthofpayload);
536     };
537     void ConvertToPacketType2(TRACEAllInOnePacketType * tp) {
538       tp->packet_type   = (contents[0] >> 5) & 0x0007;
539
540       tp->seq_number    =  contents[1]     & 0x00FF;
541
542       tp->MAC_src     =   (contents[2] << 8) |  contents[3] ;
543
544       tp->AssociatedDataSourceID      =   (contents[4] << 8) |↩
              contents[5] ;
545       tp->AssociatedDataPacketID      =   (contents[6] << 8) |↩
              contents[7] ;
546
547       tp->lengthofpayload = contents[8]    & 0x00FF;
548
```

```
549       tp->HDTS_ = 0;
550       tp->HDTS_ = contents[10];
551       tp->mcastID_ = 0;
552       tp->mcastID_ = contents[11];
553
554       memcpy(tp->payload, contents+(DATAPACKETSIZE - ↩
            PAYLOADSIZE), tp->lengthofpayload);
555     };
556
557
558   };
559
560   struct GenericPacket_OA {
561     unsigned char encryption[ENCRYPTIONFLAGSSIZE];
562     unsigned char contents[10000];
563     int length;
564     TRACEPacketIdentifierType packettype;
565     GenericPacket_OA() {
566       encryption[0] = 0x00;
567       for (int i=0;i<1000;i++) {
568         contents[i]= 0x00;
569       }
570       length = 0;
571       packettype = -1;
572       //set_Mac_Frame_Type();
573     };
574
575     bool Unencrypt() {
576       unsigned char dummy = ENCRYPTIONFLAGS;
577       if(encryption[0] == dummy ) return(true);
```

```
578       else return(false);
579     };
580     void Encrpty() {
581       encryption[0] = ENCRYPTIONFLAGS;
582     };
583     void DeterminePacketType() {
584       if(Unencrypt())
585         packettype = TRACEPacketIdentifierType((contents[0] >>↩
              5) & 0x0007);
586       else
587         packettype = UNSUCCESSFULPACKETRXType; //This is not a↩
              TRACE related file
588     };
589
590     void PrintMembers2File(FILE * foutputp2) {
591       int dummy = 0;
592       fprintf(foutputp2, "packettype = %d, length = %d,", ↩
            packettype, length);
593       fprintf(foutputp2, "GP encryption:");
594       for (int i=0; i<ENCRYPTIONFLAGSSIZE; i++) {
595         dummy=0;
596         memcpy(&dummy, &(encryption[i]), 1);
597         fprintf(foutputp2, " [%d]%02X", i, dummy );
598       }
599       fprintf(foutputp2, ", GP CONTENTS:");
600       for (int i=0; i<length; i++) {
601         dummy=0;
602         memcpy(&dummy, &(contents[i]), 1);
603         fprintf(foutputp2, " [%d]%02X", i, dummy );
604       }
```

```
605       fprintf(foutputp2, " \n");
606     };
607
608     void ConvertfromTRACEMACPacketType(const ↩
          TRACEMACPacketType *tmp){
609       Encrpty();
610
611       switch(tmp->packet_type) {
612         case PACKET_TYPE_Beacon:  {
613           length = BEACONPACKETSIZE;
614           packettype = tmp->packet_type;
615
616           Beacon_OA OAPckt;
617           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
618           memcpy (contents, OAPckt.contents, length);
619         }
620         break;
621         case PACKET_TYPE_CA:  {
622           CA_OA OAPckt;
623           length = CAPACKETSIZE;
624
625           encryption[0] = ENCRYPTIONFLAGS;
626           packettype = tmp->packet_type;
627           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
628           memcpy (contents, OAPckt.contents, length);
629         }
630         break;
631         case PACKET_TYPE_Contention:  {
632           Contention_OA OAPckt;
633           length = CONTENTIONPACKETSIZE;
```

```
634
635           encryption[0] = ENCRYPTIONFLAGS;
636           packettype = tmp->packet_type;
637           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
638           memcpy (contents, OAPckt.contents, length);
639         }
640         break;
641         case PACKET_TYPE_Header:  {
642           Header_OA OAPckt;
643           length = HEADERPACKETSIZE;
644
645           encryption[0] = ENCRYPTIONFLAGS;
646           packettype = tmp->packet_type;
647           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
648           memcpy (contents, OAPckt.contents, length);
649         }
650         break;
651         case PACKET_TYPE_IS:  {
652           IS_OA OAPckt;
653           length = ISPACKETSIZE;
654
655           encryption[0] = ENCRYPTIONFLAGS;
656           packettype = tmp->packet_type;
657           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
658           memcpy (contents, OAPckt.contents, length);
659         }
660         break;
661         case PACKET_TYPE_DATA:  {
662           Data_OA OAPckt;
663           length = DATAPACKETSIZE;
```

```
664
665           encryption[0] = ENCRYPTIONFLAGS;
666           packettype = tmp->packet_type;
667           OAPckt.ConvertFromPacketType2(*(tmp->RxPckt));
668           memcpy (contents, OAPckt.contents, length);
669         }
670         break;
671
672         default:
673           printf("!!! INVALID CONVERSION @ ↩
                ConvertfromTRACEMACPacketType typ = %d!!! \n", tmp↩
                ->packet_type);
674         break;
675       }
676     }
677
678     void ConvertToTRACEMACPacketType(TRACEMACPacketType *tmp) ↩
        {
679       tmp->RxPckt->packet_type = packettype;
680       tmp->packet_type = packettype;
681       if (packettype == UNSUCCESSFULPACKETRXType) {
682         tmp->isSuccessfullyRx = false;
683       }
684       else {
685         switch(packettype) {
686           case PACKET_TYPE_Beacon:  {
687             Beacon_OA OAPckt;
688             length = BEACONPACKETSIZE;
689             tmp->packet_type = packettype;
690             memcpy (OAPckt.contents, contents, length);
```

```
691             OAPckt.ConvertToPacketType2(tmp->RxPckt);
692           }
693           break;
694           case PACKET_TYPE_CA:  {
695             CA_OA OAPckt;
696             length = CAPACKETSIZE;
697             tmp->packet_type = packettype;
698             memcpy (OAPckt.contents, contents, length);
699             OAPckt.ConvertToPacketType2(tmp->RxPckt);
700           }
701           break;
702           case PACKET_TYPE_Contention:  {
703             Contention_OA OAPckt;
704             length = CAPACKETSIZE;
705             tmp->packet_type = packettype;
706             memcpy (OAPckt.contents, contents, length);
707             OAPckt.ConvertToPacketType2(tmp->RxPckt);
708           }
709           break;
710           case PACKET_TYPE_Header:  {
711             Header_OA OAPckt;
712             length = HEADERPACKETSIZE;
713             tmp->packet_type = packettype;
714             memcpy (OAPckt.contents, contents, length);
715             OAPckt.ConvertToPacketType2(tmp->RxPckt);
716           }
717           break;
718           case PACKET_TYPE_IS:  {
719             IS_OA OAPckt;
720             length = ISPACKETSIZE;
```

```
721             tmp->packet_type = packettype;
722             memcpy (OAPckt.contents, contents, length);
723             OAPckt.ConvertToPacketType2(tmp->RxPckt);
724           }
725           break;
726           case PACKET_TYPE_DATA:  {
727             Data_OA OAPckt;
728             length = DATAPACKETSIZE;
729             tmp->packet_type = packettype;
730             memcpy (OAPckt.contents, contents, length);
731             OAPckt.ConvertToPacketType2(tmp->RxPckt);
732           }
733           break;
734
735           default:
736             printf("!!! INVALID CONVERSION @ ↩
                 ConvertToTRACEMACPacketType !!! typ = %d \n",↩
                 packettype);
737           break;
738         }
739       }
740     };
741   };
```

C TRACE Extension

```
1  #include <stdio.h>
2  #include <Windows.h>
3  #include <math.h>
4  #include <stdlib.h>
5
6
7
8  #include "tracepackettypes.h"
9  #include "trace_timer.h"
10 #include "sora.h"
11 #include "DataPacketUIDControlRegister.h"
12
13 #define MAX_NF_p_CH (1)
14 #define LimitNumTimesCHFullObserved (3)
15 #define IS_DCA_ON (0)
16 #define IS_P_ON (0)
17
18 #define INTERFERENCETHRESHOLD (ULONG_MAX)
19
```

```
20 #define MONITOR_PRINT_TRACEMANUPDATES (0)
21 #define MONITOR_PRINT_TRACEDEBUGGER (1)
22
23 #define FirstContentionSYNC (200)
24
25 #define NumberofDestinations (4)
26
27
28 class MAC_CDCAUTRACE {
29
30 public:
31
32   int GlobalStartUpSwitch;
33   void SetGlobalStartUpSwitch() {
34     GlobalStartUpSwitch = 1;
35   };
36   int GlobalListenSwitch;
37
38   int BeaconPacketState; //States for packets: last one sent↩
       -1;  not ready = 0; ready =1
39   int CAPacketState;
40   int ContentionPacketState;
41   int HeaderPacketState;
42   int ISPacketState;
43   int DataPacketState;
44
45   int IDBeaconPacketState; //States for packets: last one ↩
      sent -1;  not ready = 0; ready =1
46   int IDCAPacketState;
47   int IDContentionPacketState;
```

```
48    int IDHeaderPacketState;
49    int IDISPacketState;
50    int IDDataPacketState;
51
52    int BeaconPacketState2; //States for packets: last one ↩
        sent -1;  not ready = 0; ready =1
53    int CAPacketState2;
54    int ContentionPacketState2;
55    int HeaderPacketState2;
56    int ISPacketState2;
57    int DataPacketState2;
58
59    int ISLostLimit;
60    int numoftimesInputfileisread;
61
62    int IsAutoGTDMASelection;
63    int presetGTDMA;
64
65    bool TimeUpdateLock;
66    TRACETimeType LastTimeUpdateLockTime;
67
68
69    int t_mode_operation;
70    TRACETimeType Timein_t_mode_Operation;
71    TRACETimeType Timein_slot;
72    TRACETimeType NOW;
73    TRACETimeType TimeinFrame;
74    int FNumber;
75    int SFNumber;
76    FILE *foutputpState;
```

```
77    FILE *foutputp;
78    FILE *foutputp2;
79    FILE *fout_ChannelPower;
80    FILE *fprx;
81    FILE *fp_indatafile;
82
83    FILE *fp_outdatafile;
84    FILE *fp_outdatafile1;
85    FILE *fp_outdatafile2;
86    FILE *fp_outdatafile3;
87    FILE *fp_outdatafile4;
88    int NumberofRxDataPackets[NumberofDestinations];
89    int TotalNumberofRxPckt;
90    int TotalNumberofTxPckt;
91    int TotalNumberofTxDataPckt;
92    int NumberofRxPckt[NumberofPacketTypes];
93    int NumberofTxPckt[NumberofPacketTypes];
94    char dummypayload[PAYLOADSIZE];
95
96    TRACETimeType LastDataACKTime_;
97    TRACETimeType StartUPTimeLimit_;
98
99    int LastTx_tmode;
100   int LastTx_slotnumber;
101
102   TRACETimeType FrameLimits[1+NCAS_+NCS_+1+NDS_+NDS_];
103   //int RadioStateInSlot[][1+NCAS_+NCS_+1+NDS_+NDS_];
104
105   TRACEMACPacketType * pktTx_;
106
```

```
107   // Primary variables
108   CHTableStructure CHTable_;  //The information kept about ↩
        the CH that the node is aware of
109   ULONG InFrameIntfLevels[2+2*MAXNDS_]; //The interfrance ↩
        levels measured in each slot
110   ULONG InterFrameIntfLevels[MAXNTD_][2+2*MAXNDS_] ; //↩
        Global understanding of the interference levels updated ↩
        through measurements
111
112   int OFTNum_;
113   int CyclicOFTNum_;
114   int MyContentionSlotNumber_;
115   int NextController_;  //After Receiving beacon record the ↩
        controller of the frame to this temporary storage.
116   int DoIHaveAReservedSlot_;
117   int DoINeedExtraContention_;
118   int DoIHaveExtraContention_;
119   int ExtraContentionMode_;
120   int DoINeed2SendContention_;
121   int DoINeed2SendContention();
122   int DoIHaveData_;
123   int MyGTDMASlot_;
124   void SelectGTDMASlot();
125   int DoIHaveAReservedSlot();
126   bool DoISendCAinThisFrame_;
127   bool DoISendSYNCimThisFrame_;
128   void DetermineDoISendCAinThisFrame() {
129     if(GetProbability(2) == 1) DoISendCAinThisFrame_= true;
130     else DoISendCAinThisFrame_= false;
131   };
```

```
132
133
134     int IsStartupSituation_;
135
136     void TakeCHRole();
137     void ResignfromCHRole(int newCHid);
138     int TotalNumofCHResigns;
139
140     bool isThisMyGTDMASlot(int t=-1);
141
142     int AmIASourceNode_;
143
144     int ActiveDataSlots_;
145     int AllActiveDataSlots_;
146
147     ISTableStructure ISTable_[StdVecSize];
148
149     int MyDataTransSlot_;
150     TRACENodeIDType DataSlotSourceList[MAXNDS_];
151
152     void GetActiveDataSlotsFromTrSchSrcPrio();
153
154     TRACETimeType LastBeaconTime_;
155     TRACETimeType LastDataTime_;
156
157     TRACETimeType sstime_ ;
158     int ScheduleControlTable_[MAXNTD_][MAXNDS_];
159
160     TRACENodeIDType index_  ; //ID of the current node
161
```

```
162    TRACETimer TRACEClock;
163    int ISAutoCHSelection_; //This determines wheter CHs are ↩
         auto selected or preset. Default value is 1;
164    int AmITheController_;
165    int AmITheController(){
166      return(AmITheController_);
167    };
168    ULONG DoIListenThisSlot_;
169    FLAG* PtrRxOperationFlag;
170    bool PowerDetected;
171    bool PowerStateChanged;
172
173    int FL[NTD_];
174    int SelectedFrames[NTD_];
175
176    int RadioState;
177    bool isStateChanged;
178
179    MAC_CDCAUTRACE() ;
180
181
182    MAC_CDCAUTRACE(FLAG* RxWorkFlag) ;
183
184    void InitializeVariables();
185
186    int Calc_t_mode_Operation();
187
188    void BeaconSyncronize(double  &endofbeacon_t) ;
189
```

```
190    void SimpleSyncronize(const double  &t_rx_end, const int &↩
         pckt_type);
191
192    void ContentionSyncronize(const double  &t_rx_end, const ↩
         int &pckt_type, const int &SlotNumberinPckt_);
193
194
195    void recv(TRACEMACPacketType* p); //This is the generic ↩
         entry to the program. Any outside function should ↩
         interact by calling this function
196    void HandleIncoming(TRACEMACPacketType* p);//This is to ↩
         perform receiving operations on the recveived packet
197    void sendUp(TRACEMACPacketType* p);//Differentiatiate ↩
         between different packets and call them one by one
198
199    int MCastRelayNode_;
200    int MCastGroupID_;
201    bool DoIBelongtoMcastGroup(int qmcastid){
202      if(qmcastid == MCastGroupID_) return(TRUE);
203      return(FALSE);
204    };
205    int ISMHSTREAM;
206    int ISNBSTREAM;
207    int ISMCSTREAM;
208
209
210    //Receptions
211    void MAC_CDCAUTRACE::RecordChannelPower(ULONG  &el);
212    void RxBeacon(TRACEMACPacketType* p);//RX Beacon
213    void RxCA(TRACEMACPacketType* p);//RX Beacon
```

```
214     void RxContention(TRACEMACPacketType* p);//RX Beacon
215     void RxHeader(TRACEMACPacketType* p);//RX Beacon
216     void RxIS(TRACEMACPacketType* p);//RX Beacon
217     void RxDATA(TRACEMACPacketType* p);//RX Beacon
218
219     void UpdateCHTable(TRACENodeIDType src, int ads, ↩
          TRACEPowerType rxpwr, TRACEPriorityType prio);
220     void ProcessCHTable();
221     void DelCHTableEnt(TRACENodeIDType nodeid);
222     void ReArrangeCHTable();
223
224     void fetchSlot(TRACEMACPacketType* p);
225
226     void CalculateTimeInFrame() ;
227
228     void CalculateFrameNumber() ;
229
230     void CalculateSFrameNumber() ;
231
232     int Get_t_mode_Operation() ;
233
234     void Set_t_mode_Operation(int const &m);
235
236     void CheckForObsoletePackets(int k) ;
237
238     void UpdateTraceTime();
239
240     bool DoIListenThisSlot();
241     bool DoIListenThisSlot2();
242
```

```
243     bool DoITXThisSlot() ;
244
245
246     bool AmIInListenState();
247
248     double GetCurrentTime();
249
250     double GetTimeinFrame();
251
252     int SuperFrameNumber();
253
254     int FrameNumber();
255
256     void MarkSlot_TX(int i);
257     void MarkSlot_RX(int i);
258
259     void SetMyDownStreamNodeID2(int dst);
260     void SetMyUpStreamNodeID2(int dst);
261
262     void ToggleSourceMode(int k);
263     //SEND PACKETS
264     bool preparePacket(TRACEMACPacketType* p, ↩
          TRACEPacketIdentifierType typ, int seq_number);
265     bool prepareBeacon(TRACEMACPacketType* p);
266     bool prepareCA(TRACEMACPacketType* p);
267     bool prepareContention(TRACEMACPacketType* p);
268     bool prepareHeader(TRACEMACPacketType* p);
269     bool prepareIS(TRACEMACPacketType* p);
270     bool prepareData(TRACEMACPacketType* p);
271
```

```
272     void HandleOutgoing(int sentpckttype);
273     void sendBeacon();
274     void sendCA();
275     void sendContention();
276     void sendHeader();
277     void sendIS();
278     void sendData();
279     void sendGenericPacket();
280
281
282       //DCA TRACE Related
283     int NumTimesCHFullObserved;
284     int NumofNeededTimeDivisions;
285
286     void prepareGenericPacket(TRACEMACPacketType* p);
287
288       // Periodic functions
289     void ResetNumIntf();
290     void ResetNumIntf2();
291     void WakeForBeacon();
292     void SleepAfterBeacon();
293     void WakeForHeader();
294     void SleepAfterIS();
295     void WakeForDataSlot();
296     void SleepAfterDataSlot();
297     void SetSST();
298     void CheckDataPacketDelay();
299     void CheckDACK();
300     void CheckCHIST();
301     int GetProbability(int i);
```

```
302     void ProcessDACKTable();
303     void CheckReStartUp();
304
305     TrSchSrcPrioType TrSchSrcPrio_[MAXNTD_][MAXNDS_];
306     StreamRegister StreamRegister_;
307     Stream* Stream2BeSent;
308     void SelectStream2BeSent();
309
310
311     int LastReceivedDataSourceID ;
312     int LastReceivedDataPacketID ;
313     int LastReceivedDataPacketID1 ;
314     int LastReceivedDataPacketID2 ;
315     int LastReceivedDataPacketID3 ;
316     int LastReceivedDataPacketID4 ;
317
318     bool CheckTransmissionSchedule(int nodeid) ;
319     void InsertTransmissionSchedule(int nodeid, int prio, int ↩
          targetframe) ;
320     int InsertTransmissionScheduleAtLowestInterference(int ↩
          nodeid, int prio, int targetframe=-1);
321
322     bool DoISendData();
323     bool DoISendDataInTheFollowingSlot();
324     bool DoISendISInTheFollowingSlot();
325     bool DoIListenToTheFollowingDataSlot();
326
327     void RearrangeTransmissionSchedule();
328
329
```

```
330   int ContentionSlotNumber_;
331   int GetContentionSlotNumber() {
332     return(ContentionSlotNumber_);
333   };
334   void CalculateContentionSlotNumber() {
335   double length =  ContentionSlotLength_;
336   // return the Data slot number for this time
337     ContentionSlotNumber_ = (int) ( Timein_t_mode_Operation ↩
        / length );
338   };
339   int ISSlotNumber_;
340   int GetISSlotNumber() {
341     return( ISSlotNumber_ );
342   };
343   void CalculateISSlotNumber() {
344     // return the Data slot number for this time
345     double length =  ISSlotLength_;
346     ISSlotNumber_ = (int) ( Timein_t_mode_Operation / length↩
          );
347   };
348   int DataSlotNumber_;
349   int GetDataSlotNumber() {
350     return(DataSlotNumber_ );
351   };
352   void CalculateDataSlotNumber() {
353     double length =  DataSlotLength_;
354     // return the Data slot number for this time
355     DataSlotNumber_ = (int) ( Timein_t_mode_Operation / ↩
        DataSlotLength_ );
356
```

```
357     };
358
359     int GetSlotNumber() {
360       switch(t_mode_operation) {
361         case    T_MODE_CONTENTIONSLOT      :
362           return(GetContentionSlotNumber());
363         case    T_MODE_ISSLOT        :
364           return(GetISSlotNumber());
365         case    T_MODE_DATASLOT        :
366           return(GetDataSlotNumber());
367         default:
368           return(-1);
369         break;
370       };
371     };
372     TRACETimeType GetSlotLength() {
373       switch(t_mode_operation) {
374         case    T_MODE_BEACONSLOT        :
375           return(BeaconSlotLength_);
376         case    T_MODE_CASLOT          :
377           return(CASlotLength_);
378         case    T_MODE_CONTENTIONSLOT      :
379           return(ContentionSlotLength_);
380         case    T_MODE_HEADERSLOT        :
381           return(HeaderSlotLength_);
382         case    T_MODE_ISSLOT        :
383           return(ISSlotLength_);
384         case    T_MODE_DATASLOT        :
385           return(DataSlotLength_);
386         default:
```

```
387         return(95.0);
388       break;
389     };
390   };
391
392   TRACETimeType GetPacketSlotLength(int pckt_typ) {
393     switch(pckt_typ) {
394       case    PACKET_TYPE_Beacon       :
395       return(BeaconSlotLength_ - (RampTimeAdjustment + ↩
          TracePreamble_ + IFS_) );
396       case    PACKET_TYPE_CA           :
397         return(CASlotLength_ - (RampTimeAdjustment + ↩
            TracePreamble_ + IFS_) );
398       case    PACKET_TYPE_Contention     :
399         return(ContentionSlotLength_ - (RampTimeAdjustment +↩
             TracePreamble_ + IFS_) );
400       case    PACKET_TYPE_Header       :
401         return(HeaderSlotLength_ - (RampTimeAdjustment + ↩
            TracePreamble_ + IFS_) );
402       case    PACKET_TYPE_IS         :
403         return(ISSlotLength_ - (RampTimeAdjustment + ↩
            TracePreamble_ + IFS_) );
404       case    PACKET_TYPE_DATA         :
405         return(DataSlotLength_ - (RampTimeAdjustment + ↩
            TracePreamble_ + IFS_) );
406       default:
407         return(-1.0);
408       break;
409     };
410   };
```

```
411
412     TRACETimeType GetTimeinSlotLength(TRACETimeType t=-1.0) {
413       if(t<0) t = NOW;
414       int sn = GetSlotNumber();
415       if(sn>0) return(Timein_t_mode_Operation - sn * ↩
            GetSlotLength() );
416       else return(Timein_t_mode_Operation);
417     };
418
419     int MyGTDMASlots_[MAXNTD_];//BK
420
421     void GetRandomContentionSlotNumber();
422     void contentionTimer();
423
424     //RESET Funcion
425     void ResetCHTable();
426     void ResetScheduleControlTable();
427     void ResetTransmissionSchedule();
428     void ResetRcvPwrLevel();
429     void ResetISTable();
430     void ResetLCTable();
431     void ResetDACKTable();
432     void ResetDataSlotSourceList();
433     void ResetIntfLevel();
434     void ResetGTDMASlots();
435
436
437     //MONITORING
438     void PrintTimeFrameParameters();
439     void PrintTraceManegerState();
```

```
440     void PrintTransmissionSchedule(FILE *foutp);
441     void PrintDataSlotSourceList();
442     void PrintIntfLevel(FILE *foutp);
443     void PrintCHTable(FILE *foutp);
444     void ClearMONITORFlags();
445
446     int ListenDataState;
447
448     TRACETimeType LastQueryTime;
449     void Query(){
450       if (NOW - LastQueryTime > 1) {// 1 sec interval{
451         TRACETimeType BeforeQuery, AfterQuery;
452         TRACEClock.GetCurrentSeconds(BeforeQuery);
453         LastQueryTime = NOW;
454         PrintTRACEState(stdout);
455         TRACEClock.GetCurrentSeconds(AfterQuery);
456         StreamRegister_.PrintList(stdout);
457         printf("******End of TRACE Query Report Took = %.7f ↩
              secs*******\n", AfterQuery - LastQueryTime );
458
459       }
460       };
461     void PrintTRACEState(FILE *foutp){
462       fprintf(foutp,"******TRACE Query Report*******\n");
463       fprintf(foutp,"<%d> t_mode_operation = %d [FNum= %d ↩
          Timein_t_mode_Operation = %.7f @ %.7f]\n", index_, ↩
          t_mode_operation, FNumber, Timein_t_mode_Operation, ↩
          NOW);
464
465       if(Stream2BeSent != (Stream*)0)
```

```
466     fprintf(foutp," Stream2BeSent->sourceID = %d ↩
          DoIHaveData_ = %d tDoIHaveAReservedSlot_=%d ↩
          numoftimesInputfileisread =%d \n", Stream2BeSent->↩
          sourceID, DoIHaveData_, DoIHaveAReservedSlot_,↩
          numoftimesInputfileisread);
467
468     fprintf(foutp,"BeaconPacketState = %d  CAPacketState = %↩
          d ContentionPacketState = %d HeaderPacketState = %d ↩
          ISPacketState = %d DataPacketState = %d \n \n"
469         , BeaconPacketState, CAPacketState, ↩
            ContentionPacketState, HeaderPacketState, ↩
            ISPacketState, DataPacketState
470         );
471
472     for (int i=0; i<NumberofDestinations; i++) {
473       fprintf(foutp,"NumberofRxDataPackets[%d] = %d ", i, ↩
            NumberofRxDataPackets[i] );
474     }
475     fprintf(foutp,"\n");
476     fprintf(foutp,"TotalNumberofTxDataPckt \t= %d \↩
          tTotalNumberofRxPckt \t= %d \tTotalNumberofTxPckt \t= ↩
          %d\n", TotalNumberofTxDataPckt, TotalNumberofRxPckt, ↩
          TotalNumberofTxPckt);
477     for (int i=0; i<NumberofPacketTypes; i++) {
478       fprintf(foutp,"NumberofRxPckt[%d] = %d \t ↩
            NumberofTxPckt[%d] = %d \n", i, NumberofRxPckt[i],i ↩
            ,NumberofTxPckt[i] );
479     }
480     fprintf(foutp,"\n");
481
```

```
482     fprintf(foutp,"LastReceivedDataSourceID \t= %d  \↩
          tLastReceivedDataPacketID \t= %d \n", ↩
          LastReceivedDataSourceID, LastReceivedDataPacketID);
483     fprintf(foutp,"LastReceivedDataPacketID1 \t= %d \↩
          tLastReceivedDataPacketID2 \t= %d \↩
          tLastReceivedDataPacketID3 \t= %d \↩
          tLastReceivedDataPacketID4 \t= %d\n"
484     , LastReceivedDataPacketID1, LastReceivedDataPacketID2, ↩
          LastReceivedDataPacketID3, LastReceivedDataPacketID4);
485
486     fprintf(foutp,"AmIASourceNode_ \t \t= %d  feof(↩
          fp_indatafile) = %d DoINeed2SendContention_ = %d \n", ↩
          AmIASourceNode_, feof(fp_indatafile), ↩
          DoINeed2SendContention_);
487     fprintf(foutp,"NextController_ \t \t= %d \↩
          tDoIHaveAReservedSlot_ \t \t= %d \↩
          tMyContentionSlotNumber_ \t= %d\n", NextController_, ↩
          DoIHaveAReservedSlot_, MyContentionSlotNumber_);
488     fprintf(foutp,"AmITheController_ \t \t= %d \↩
          tMyGTDMASlot_ \t \t \t= %d \tActiveDataSlots_ \t \t= %↩
          d \tAllActiveDataSlots_ \t= %d TotalNumofCHResigns = %↩
          d \n \n", AmITheController_, MyGTDMASlot_, ↩
          ActiveDataSlots_, AllActiveDataSlots_, ↩
          TotalNumofCHResigns);
489     for(int t=0; t<NTD_; t++){
490       fprintf(foutp,"MyGTDMASlots[%d] = %d \t", t, ↩
            MyGTDMASlots_[t]);
491       }
492     fprintf(foutp,"\n\n");
493     for (int t= 0; t<NTD_; t++) {
```

```
494         for (int i= 0; i<NDS_; i++) {
495           if(TrSchSrcPrio_[t][i].src_ >= 0 ) //If valid node
496             fprintf(foutp,"TrSchSrcPrio_[%d][%d].src_ = %d \t"↩
                  ,t,i,TrSchSrcPrio_[t][i].src_);
497           else
498             fprintf(foutp,"TrSchSrcPrio_[%d][%d].src_ = --- \t↩
                  ",t,i);
499         }
500         fprintf(foutp,"\n");
501       }
502       PrintIntfLevel(foutp);
503       PrintCHTable(foutp);
504     };
505
506     bool DataReady;
507     char DataPayload[int(PAYLOADSIZE)];
508
509
510   protected:
511
512   static int ISFLOODINGENERGYSAVINGON;
513   static int MONITOR_fetchSlot;
514
515   static int DEBUG_FUNCTION_INPUT_OUTPUT ;
516   static int MONITOR_SetSST ;
517   static int MONITOR_BEACON_TRANSMISSION ;
518   static int MONITOR_CA_TRANSMISSION ;
519   static int MONITOR_HEADER_TRANSMISSION ;
520   static int MONITOR_PI_TRANSMISSION ;
521   static int MONITOR_PI_RECEPTION ;
```

```
522 static int MONITOR_SOURCE_DATA_TRANSMISSION ;
523 static int MONITOR_ALL_DATA_TRANSMISSION ;
524 static int MONITOR_CH_DATA_TRANSMISSION ;
525 static int MONITOR_ALL_PACKET_RECEPTIONS ;
526 static int MONITOR_DATA_PACKET_RECEPTION ;
527 static int MONITOR_MCastNode_DATA_PACKET_RECEPTION ;
528 static int MONITOR_LATE_PACKET_DROPS ;
529 static int MONITOR_NEW_CLUSTERHEAD ;
530 static int MONITOR_RESIGNING_CLUSTERHEAD ;
531 static int MONITOR_TIME_FRAME_PARAMETERS ;
532 static int MONITOR_CHTable_AT_LATE_PACKET_DROPS ;
533 static int ↩
    MONITOR_TransmissionSchedule_AT_HEADER_TRANSMISSION ;
534 static int MONITOR_TransmissionSchedule;
535 static int DEBUG_DOUBLE_LISTING_IN_TRANSMISSION_SCHEDULE ;
536 static int MONITOR_VIRTUAL_DATA_ACKs ;
537 static int MONITOR_CH_NODATA ;
538
539 static int MONITOR_PACKET_PREP ;
540
541 static int MONITOR_GATEWAYS ;
542 static int MONITOR_ND_TRANSMISSION ;
543 static int MONITOR_ND_RECEPTION ;
544 static int MONITOR_MCastRelayNode_DATA_TRANSMISSION ;
545 static int MONITOR_CheckReStartUp ;
546 static int NO_ND ;
547 static int NO_CH_DATA_TRANS ;
548
549 static int MONITOR_SENDCONTENTION ;
550 static int MONITOR_DoINeed2SendContention ;
```

```
551  static int MONITOR_CONTENTION_RECEPTION ;
552  static int MONITOR_EXTRACONTENTION ;
553
554  static int MONITOR_UNIQUE_DATA_RECEPTION ;
555  static int MONITOR_DUPLICATE_DATA_RECEPTION ;
556  static int MONITOR_HANDLEOUTGOING ;
557  static int MONITOR_SelectGTDMA ;
558  static int MONITOR_IntLevelAtHeader ;
559  static int MONITOR_CheckDACK ;
560  static int MONITOR_FRAME_SELECTION ;
561  static int MONITOR_CHTABLE_AT_PROCESSING ;
562  static int MONITOR_OBSOLETE_CHS ;
563  static int MONITOR_PRINT_CHS_FILE ;
564  static int MONITOR_PRINT_RX_FILE ;
565  static int MONITOR_PRINT_TX_FILE ;
566  static int SOURCEISACLUSTERHEAD ;
567  static int MONITOR_RecordChannelPower;
568  };
```

Bibliography

[1] T. Numanoglu, B. Tavli, and W. Heinzelman. An analysis of coordinated and non-coordinated medium access control protocols under channel noise. *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 2642–2648 Vol. 4, Oct. 2005.

[2] A. Chandra, V. Gummalla, and J. O. Limb. Wireless medium access control protocols. *IEEE Communications Surveys and Tutorials*, 3:2–15, 2000.

[3] P. Mohapatra, J. Li, and C. Gui. Qos in mobile ad hoc networks. *IEEE Wireless Communications Magazine*, 10:44–52, 2003.

[4] B. Tavli and W. B. Heinzelman. MH-TRACE: Multi hop time reservation using adaptive control for energy efficiency. *IEEE Journal on Selected Areas of Communications*, 22(5):942–953, June 2004.

[5] T. Cooklev. *Wireless Communication Standarts*. IEEE Press, 2004.

[6] J. Karaoguz. High-rate wireless personal area networks. *Communications Magazine, IEEE*, 39(12):96–102, Dec 2001.

[7] T. Numanoglu, B. Tavli, and W. B. Heinzelman. The effects of channel errors on coordinated and non-coordinated medium access control protocols. In *Proceedings of IEEE International Conference on Wireless and Mobile Computing*, volume 1, pages 58–65, Aug 2005.

[8] Bora Karaoglu, Tolga Numanoglu, and Wendi Heinzelman. Analytical performance of soft clustering protocols. *Ad Hoc Networks*, 9(4):635 – 651, 2011.

[9] Lifei Huang and Ten-Hwang Lai. On the scalability of ieee 802.11 ad hoc networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '02, pages 173–182, New York, NY, USA, 2002. ACM.

[10] IEEE 802.15.3 Working Group. Part 15.3: Wireless medium access control (MAC) and physical layer (PHY) specifications for high rate wireless personal area networks (WPAN). *IEEE Draft Standard, Draft P802.15.3/D16*, Feb 2003.

[11] Mikko Kohvakka, Mauri Kuorilehto, Marko Hännikäinen, and Timo D. Hämäläinen. Performance analysis of ieee 802.15.4 and zigbee for large-scale wireless sensor network applications. In *Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, PE-WASUN '06, pages 48–57, New York, NY, USA, 2006. ACM.

[12] M. Rahnema. Overview of the gsm system and protocol architecture. *Communications Magazine, IEEE*, 31(4):92–100, Apr 1993.

[13] T. S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[14] Jason Redi, Bill Watson, Ram Ramanathan, Prithwish Basu, Fabrice Tchakountio, Michael Girone, and Martha Steenstrup. Design and implementation of a mimo mac protocol for ad hoc networking. volume 6248, page 624802. SPIE, 2006.

[15] V. Tippanagoudar, I. Mahgoub, and A. Badi. Implementation of the sensor-mac protocol for the jist/swans simulator. In *Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on*, pages 225 –232, may 2007.

[16] Heping Wang, Xiaobo Zhang, Farid Naït-Abdesselam, and Ashfaq Khokhar. Dps-mac: an asynchronous mac protocol for wireless sensor networks. In *Proceedings of the 14th international conference on High performance computing*, HiPC'07, pages 393–404, Berlin, Heidelberg, 2007. Springer-Verlag.

[17] Jun Zhang, F.A. Nait, and B. Bensaou. Performance analysis of an energy efficient mac protocol for sensor networks. In *Parallel Architectures, Algorithms, and Networks, 2008. I-SPAN 2008. International Symposium on*, pages 254 –259, may 2008.

[18] Sung hwa Hong and Hoon ki Kim. A multi-hop reservation method for end-to-end latency performance improvement in asynchronous mac-based wireless sensor networks. *Consumer Electronics, IEEE Transactions on*, 55(3):1214 –1220, august 2009.

[19] Anna Förster, Alexander Förster, and Amy L. Murphy. Optimal cluster sizes for wireless sensor networks: An experimental analysis. In *ADHOCNETS 2009: First International Conference on Ad Hoc Networks*, September 2009.

[20] Chonggang Wang, K. Sohraby, Bo Li, M. Daneshmand, and Yueming Hu. A survey of transport protocols for wireless sensor networks. *IEEE Network*, 20(3):34–40, May-June 2006.

[21] Dawei Xia and Natalija Vlajic. Near-optimal node clustering in wireless sensor networks for environment monitoring. *Advanced Information Networking and Applications, International Conference on*, 0:632–641, 2007.

[22] Ying-Ju Chen and Jin-Fu Chang. Per connection delay analysis of a frame-based tdma/cdma mac protocol. *Perform. Eval.*, 57(1):19–55, 2004.

[23] M.F. Neuts, Jun Guo, M. Zukerman, and Hai Le Vu. The waiting time distribution for a tdma model with a finite buffer and state-dependent service. *Communications, IEEE Transactions on*, 53(9):1522 – 1533, sept. 2005.

[24] G. Bianchi. Performance analysis of the ieee 802.11 distributed coordination function. *Selected Areas in Communications, IEEE Journal on*, 18(3):535 –547, mar 2000.

[25] Sangkyu Baek and Bong Dae Choi. Performance analysis of power save mode in ieee 802.11 infrastructure wlan. In *Telecommunications, 2008. ICT 2008. International Conference on*, pages 1 –4, 16-19 2008.

[26] Jie Hui and M. Devetsikiotis. A unified model for the performance analysis of ieee 802.11e edca. *Communications, IEEE Transactions on*, 53(9):1498 – 1510, sept. 2005.

[27] Yaser Pourmohammadi Fallah and Hussein Alnuweiri. Hybrid polling and contention access scheduling in ieee 802.11e wlans. *J. Parallel Distrib. Comput.*, 67(2):242–256, 2007.

[28] COPCA. An improved model for gsm/gprs/edge performance evaluation. In *LANC '07: Proceedings of the 4th international IFIP/ACM Latin American conference on Networking*, pages 23–33. ACM, 2007.

[29] N.F. Timmons and W.G. Scanlon. Analysis of the performance of ieee 802.15.4 for medical sensor body area networking. In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 16 – 24, 4-7 2004.

[30] Chandramani Kishore Singh, Anurag Kumar, and P. M. Ameer. Performance evaluation of an ieee 802.15.4 sensor network with a star topology. *Wirel. Netw.*, 14(4):543–568, 2008.

[31] Mikko Kohvakka, Mauri Kuorilehto, Marko Hännikäinen, and Timo D. Hämäläinen. Performance analysis of ieee 802.15.4 and zigbee for large-scale wireless sensor network applications. In *PE-WASUN '06: Proceedings of the 3rd ACM international workshop on Performance evaluation of wireless ad hoc, sensor and ubiquitous networks*, pages 48–57, New York, NY, USA, 2006. ACM.

[32] Iyappan Ramachandran, Arindam K. Das, and Sumit Roy. Analysis of the contention access period of ieee 802.15.4 mac. *ACM Trans. Sen. Netw.*, 3(1):4, 2007.

[33] S.A. Khan and F.A. Khan. Performance analysis of a zigbee beacon enabled cluster tree network. In *Electrical Engineering, 2009. ICEE '09. Third International Conference on*, pages 1 –6, 9-11 2009.

[34] Lin X. Cai, Jon W. Mark, Xuemin Shen, Kuang-Hao Liu, and Humphrey Rutagemwa. Performance analysis of hybrid medium access protocol in ieee 802.15.3 wpan. In *Consumer Communications and Networking Conference, 2007. CCNC 2007. 4th IEEE*, pages 445 –449, jan. 2007.

[35] Yi-Hsien Tseng, Eric Hsiao kuang Wu, and Gen-Huey Chen. Maximum traffic scheduling and capacity analysis for ieee 802.15.3 high data rate MAC protocol. *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, 3:1678–1682 Vol.3, Oct. 2003.

[36] B.M. Leiner, D.L. Nielson, and F.A. Tobagi. Issues in packet radio network design. *Proceedings of the IEEE*, 75(1):6 – 20, jan. 1987.

[37] I. Cidon and M. Sidi. Distributed assignment algorithms for multi-hop packet-radio networks. In *INFOCOM '88. Networks: Evolution or Revolution, Proceedings. Seventh Annual Joint Conference of the IEEE Computer and Communcations Societies, IEEE*, pages 1110 –1118, march 1988.

[38] Lin Gao and Xinbing Wang. A game approach for multi-channel allocation in multi-hop wireless networks. In *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '08, pages 303–312, New York, NY, USA, 2008. ACM.

[39] M. Felegyhazi, M. Cagalj, S.S. Bidokhti, and J.-P. Hubaux. Non-cooperative multi-radio channel allocation in wireless networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 1442 –1450, May 2007.

[40] R. Ramaswami and K.K. Parhi. Distributed scheduling of broadcasts in a radio network. In *INFOCOM '89. Proceedings of the Eighth Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging, IEEE*, pages 497 –504 vol.2, april 1989.

[41] A. Ephremides and T.V. Truong. Scheduling broadcasts in multihop radio networks. *Communications, IEEE Transactions on*, 38(4):456 –460, apr 1990.

[42] A. Sen and M.L. Huson. A new model for scheduling packet radio networks. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 3, pages 1116 –1124 vol.3, mar 1996.

[43] A. Raniwala and Tzi cker Chiueh. Architecture and algorithms for an ieee 802.11-based multi-channel wireless mesh network. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications*

Societies. Proceedings IEEE, volume 3, pages 2223 – 2234 vol. 3, march 2005.

[44] L. Kleinrock and F. Tobagi. Packet switching in radio channels: Part i–carrier sense multiple-access modes and their throughput-delay characteristics. *Communications, IEEE Transactions on*, 23(12):1400 – 1416, dec 1975.

[45] F. Tobagi and L. Kleinrock. Packet switching in radio channels: Part ii–the hidden terminal problem in carrier sense multiple-access and the busy-tone solution. *Communications, IEEE Transactions on*, 23(12):1417 – 1433, dec 1975.

[46] P. Karn. Maca-a new channel access method for packet radio. In *ARRL/CRRL Amateur radio 9th computer networking conference*, volume 140, pages 134–140, 1990.

[47] S. Toumpis and A.J. Goldsmith. New media access protocols for wireless ad hoc networks based on cross-layer principles. *Wireless Communications, IEEE Transactions on*, 5(8):2228 –2241, aug. 2006.

[48] Deepanshu Shukla, Leena Chandran-Wadia, and Sridhar Iyer. Mitigating the exposed node problem in ieee 802.11 ad hoc networks. In *Computer Communications and Networks, 2003. ICCCN 2003. Proceedings. The 12th International Conference on*, pages 157 – 162, oct. 2003.

[49] N. Jain, S.R. Das, and A. Nasipuri. A multichannel csma mac protocol with receiver-based channel selection for multihop wireless networks. In *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, pages 432 –439, 2001.

[50] A. Nasipuri and S.R. Das. Multichannel csma with signal power-based channel selection for multihop wireless networks. In *Vehicular Technology Con-*

ference, 2000. IEEE-VTS Fall VTC 2000. 52nd, volume 1, pages 211 –218 vol.1, 2000.

[51] Jungmin So and Nitin H. Vaidya. Multi-channel mac for ad hoc networks: handling multi-channel hidden terminals using a single transceiver. In *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*, MobiHoc '04, pages 222–233, New York, NY, USA, 2004. ACM.

[52] V. H. MacDonald. Advanced mobile phone service: The cellular concept. *Bell Syst. Tech. J.*, 58:15–41, 1979.

[53] E. Standard. Global system for mobile communications (gsm) standard for mobile phones. 1990.

[54] I. Katzela and M. Naghshineh. Channel assignment schemes for cellular mobile telecommunication systems: a comprehensive survey. *Personal Communications, IEEE*, 3(3):10 –31, June 1996.

[55] Chih-Lin I and Pi-Hui Chao. Distributed dynamic channel allocation algorithms with adjacent channel constraints. In *Personal, Indoor and Mobile Radio Communications, 1994. Wireless Networks - Catching the Mobile Future., 5th IEEE International Symposium on*, volume 1, pages 169 –177 vol.1, September 1994.

[56] Chih-Lin I and Pi-Hui Chao. Local packing-distributed dynamic channel allocation at cellular base station. In *Global Telecommunications Conference, 1993, including a Communications Theory Mini-Conference. Technical Program Conference Record, IEEE in Houston. GLOBECOM '93., IEEE*, 1993.

[57] Jianping Jiang, Ten-Hwang Lai, and Neelam Soundarajan. On distributed dynamic channel allocation in mobile cellular networks. *IEEE Trans. Parallel Distrib. Syst.*, 13:1024–1037, October 2002.

[58] E. Del Re, R. Fantacci, and G. Giambene. Handover and dynamic channel allocation techniques in mobile cellular networks. *Vehicular Technology, IEEE Transactions on*, 44(2):229 –237, may 1995.

[59] Wei Song, Weihua Zhuang, and Yu Cheng. Load balancing for cellular/wlan integrated networks. *Network, IEEE*, 21(1):27 –33, jan.-feb. 2007.

[60] Celimuge Wu, K. Kumekawa, and T. Kato. A manet protocol considering link stability and bandwidth efficiency. In *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on*, pages 1 –8, oct. 2009.

[61] A. Adya, P. Bahl, J. Padhye, A. Wolman, and Lidong Zhou. A multi-radio unification protocol for ieee 802.11 wireless networks. In *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on*, pages 344 – 354, oct. 2004.

[62] Yu-Chee Tseng, Chih-Min Chao, Shih-Lin Wu, and Jang-Ping Sheu. Dynamic channel allocation with location awareness for multi-hop mobile ad hoc networks. *Computer Communications*, 25(7):676 – 688, 2002.

[63] P.G. Namboothiri and K.M. Sivalingam. Capacity analysis of multi-hop wireless sensor networks using multiple transmission channels: A case study using ieee 802.15.4 based networks. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 168 –171, oct. 2010.

[64] A.A. Bertossi and M.A. Bonuccelli. Code assignment for hidden terminal interference avoidance in multihop packet radio networks. *IEEE/ACM Transactions on Networking (TON)*, 3(4):441–449, 1995.

[65] Roberto Battiti, Alan A. Bertossi, and Maurizio A. Bonuccelli. Assigning codes in wireless networks: Bounds and scaling properties. *Wireless Networks*, 5:195–209, 1999.

[66] Limin Hu. Distributed code assignments for cdma packet radio networks. *Networking, IEEE/ACM Transactions on*, 1(6):668 –677, dec 1993.

[67] C. Schurgers, G. Kulkarni, and M.B. Srivastava. Distributed on-demand address assignment in wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 13(10):1056 – 1065, oct 2002.

[68] K.R. Chowdhury, P. Chanda, D.P. Agrawal, and Qing-An Zeng. Dca-a distributed channel allocation scheme for wireless sensor networks. In *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, volume 2, pages 1297 –1301 Vol. 2, sept. 2005.

[69] Carlos de Morais Cordeiro, Hrishikesh Gossain, and Dharma Agrawal. Multicast over wireless mobile ad hoc networks: Present and future directions. *IEEE Network*, 17:52–59, 2003.

[70] P. Mohapatra, C. Gui, and J. Li. Group communications in mobile ad hoc networks. *IEEE Computer Magazine*, 37:52–59, 2004.

[71] Lap Kong Law, Srikanth V. Krishnamurthy, and Michalis Faloutsos. Understanding and exploiting the trade-offs between broadcasting and multicasting in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 6(3):264–279, 2007.

[72] B. Tavli and W. B. Heinzelman. Energy and spatial reuse efficient network-wide real-time data broadcasting in mobile ad hoc networking. *accepted to IEEE Journal on Selected Areas of Communications*, 2005.

[73] B. Tavli and W.B. Heinzelman. MC-TRACE: multicasting through time reservation using adaptive control for energy efficiency. In *Military Communications Conference, 2005. MILCOM 2005. IEEE*, pages 2672–2678 Vol. 4, 2005.

[74] Osamu Takizawa, Masafumi Hosokawa, Ken'Ichi Takanashi, Yasushi Hada, Akihiro Shibayama, and Byeong-Pyo Jeong. Three-way pinpointing of emergency call from RFID-Reader-equipped cellular phone. In *Mobile Response: Second International Workshop on Mobile Information Technology for Emergency Response, MobileResponse 2008. Bonn, Germany, May 29-30, 2008, Revised Selected Papers*, pages 66–75, Berlin, Heidelberg, 2009. Springer-Verlag.

[75] Michael Nathaniel Rosenblatt and Steve Porter Hotelling. *Touch screen RFID tag reader*. U. S. Patent Office, July 2009. Patent application number 20090167699.

[76] Amjad Soomro and Dave Cavalcanti. Opportunities and challenges in using WPAN and WLAN technologies in medical environments. *IEEE Communications Magazine*, 45(2):114–122, Feb. 2007.

[77] Soo Young Shin, Hong Seong Park, Sunghyun Choi, and Wook Hyun Kwon. Packet error rate analysis of ZigBee under WLAN and Bluetooth interferences. *IEEE Transactions on Wireless Communications*, 6(8):2825–2830, August 2007.

[78] A. Vaios, K. Oikonomou, and I. Stavrakakis. Analysis of a topology control paradigm in WLAN/WPAN environments. *Computer Communications*, 29(11):2096 – 2108, 2006.

[79] Naoki Wakamiya and Masayuki Murata. Toward overlay network symbiosis. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing*, pages 154–155, Washington, DC, USA, 2005. IEEE Computer Society.

[80] Eli Poorter, Benoît Latré, Ingrid Moerman, and Piet Demeester. Symbiotic

networks: Towards a new level of cooperation between wireless networks. *Wirel. Pers. Commun.*, 45(4):479–495, 2008.

[81] M.I. Brownfield and N.J. Davis. Symbiotic highway sensor network. In *Vehicular Technology Conference, 2005. VTC-2005-Fall. 2005 IEEE 62nd*, volume 4, pages 2701–2705. IEEE Vehicular Technologies Society, Sept., 2005.

[82] S. Gurumani, H. Moradi, H.H. Refai, P.G. LoPresti, and M. Atiquzzaman. Dynamic path reconfiguration among hybrid FSO/RF nodes. In *IEEE Global Telecommunications Conference, GLOBECOM'08.*, pages 1–5, Dec 2008.

[83] Carlos De M. Cordeiro, Sachin Abhyankar, Rishi Toshiwal, and Dharma P. Agrawal. Bluestar: enabling efficient integration between Bluetooth WPANs and IEEE 802.11 WLANs. *Mob. Netw. Appl.*, 9(4):409–422, 2004.

[84] Tomas Sanchez Lopez and Daeyoung Kim. A context middleware based on sensor and RFID information. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 331–336, Washington, DC, USA, 2007. IEEE Computer Society.

[85] D. Cavalcanti, D. Agrawal, C. Cordeiro, Bin Xie, and A. Kumar. Issues in integrating cellular networks WLANs, and MANETs: a futuristic heterogeneous wireless network. *Wireless Communications, IEEE*, 12(3):30–41, June 2005.

[86] Hung yu Wei and R.D. Gitlin. Two-hop-relay architecture for next-generation WWAN/WLAN integration. *Wireless Communications, IEEE*, 11(2):24–30, Apr 2004.

[87] A.K. Salkintzis, C. Fors, and R. Pazhyannur. WLAN-GPRS integration for next-generation mobile data networks. *Wireless Communications, IEEE*, 9(5):112–124, Oct. 2002.

[88] B. Liu, Z. Liu, and D. Towsley. On the capacity of hybrid wireless networks. In *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 2, pages 1543–1552 vol.2, March-3 April 2003.

[89] Ashish Agarwal and P. R. Kumar. Capacity bounds for ad hoc and hybrid wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(3):71–81, 2004.

[90] R. Gedge. Symbiotic networks. *BT Technology Journal*, 21(3):67–73, 2003.

[91] B. Karaoglu, T. Numanoglu, and W. Heinzelman. Adaptation of tdma parameters based on network conditions. In *Wireless Communications and Networking Conference, 2009. WCNC 2009. IEEE*, pages 1–6, April 2009.

[92] J.-F. Frigon, V. C. M. Leung, and H. C. B. Chan. Dynamic reservation tdma protocol for wireless atm networks. *IEEE Journal on Selected Areas in Communications*, 19:370–383, 2001.

[93] D. J. Goodman, R. Valenzuela, K. Gayliard, and B. Ramamurthi. Packet reservation multiple access for local wireless communications. *IEEE Transactions on Communications*, 37:885–890, 1989.

[94] D. J. Goodman and S. W. Wei. Efficiency of packet reservation multiple access. *IEEE Transactions on Vehicular Technology*, 40:170–176, 1991.

[95] Network simulator (ns-2). `http://www.isi.edu/nsnam/ns`.

[96] D. J. Goodman, R. A. Valenzuela, K. T. Gayliard, and B. Ramamurthi. Packet reservation multiple access for local wireless communications. *IEEE Trans. Commun.*, 37:885–890, Aug 1989.

[97] D. J. Goodman and S. X. Wei. Efficiency of packet reservation multiple access. *IEEE Trans. Vehic. Tech.*, 40(1):170–176, Feb 1991.

[98] W. B. Heinzelman, A. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1:660–670, 2002.

[99] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing (WCMC)*, 2(5):483–502, 2002.

[100] C. Bettstetter and C. Wagner. The spatial node distribution of the random waypoint mobility model. In *Mobile Ad-Hoc Netzwerke, WMAN 2002*, pages 41–58. GI, 2002.

[101] Bora Karaoglu, Tolga Numanoglu, and Wendi Heinzelman. Analytical performance of soft clustering protocols. *Ad Hoc Netw.*, 9(4):635–651, June 2011.

[102] IEEE 802 LAN/MAN Standards Committee et al. Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Standard*, 802(1):999, 1999.

[103] Hai L Vu and Taka Sakurai. Collision probability in saturated ieee 802.11 networks. In *Australian Telecommunication Networks and Applications Conference, Australia*, 2006.

[104] A. Morton and B. Claise. Packet Delay Variation Applicability Statement. RFC 5481 (Informational), March 2009.

[105] C. Bettstetter, G. Resta, and P. Santi. The node distribution of the random waypoint mobility model for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 2:257–269, 2003.

[106] J. Yoon, M. Liu, and B. Noble. Sound mobility models. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MOBICOM)*, pages 205–216, 2003.

[107] Jaekyu Cho, Yoonbo Shim, Taekyoung Kwon, and Yanghee Choi. SARIF: A novel framework for integrating wireless sensor and RFID networks. *IEEE Wireless Communications*, 14(6):50–56, December 2007.

[108] D. Bertsekas and R. Gallager. *Data Networks*. Prentice-Hall, 1992.

[109] Swades De, Antonio Caruso, Tamalika Chaira, and Stefano Chessa. Bounds on hop distance in greedy routing approach in wireless ad hoc networks. *Int. J. Wire. Mob. Comput.*, 1(2):131–140, 2006.

[110] Michele Zorzi and Ramesh R. Rao. Geographic random forwarding (GeRaF) for ad hoc and sensor networks: Multihop performance. *IEEE Transactions on Mobile Computing*, 2(4):337–348, 2003.

[111] Weisstein Eric W. Circle-circle intersection. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/Circle-CircleIntersection.html`. Accessed: 2013-06-23.

[112] Kun Tan, He Liu, Jiansong Zhang, Yongguang Zhang, Ji Fang, and Geoffrey M. Voelker. Sora: high-performance software radio using general-purpose multi-core processors. *Commun. ACM*, 54(1):99–107, January 2011.

[113] Game timing and multicore processors. `http://msdn.microsoft.com/en-us/library/ee417693%28VS.85%29.aspx`. Accessed: 2013-06-23.