ABSTRACT

Title of Dissertation: Distributed Topology Organization and Transmission Scheduling in Wireless Ad Hoc Networks

Theodoros Salonidis, Doctor of Philosophy, 2004

Dissertation directed by: Professor Leandros Tassiulas
Department of Electrical and Computer Engineering

A wireless ad hoc network is a set of nodes that form an all-wireless infrastructure without the aid of any centralized administration. In this dissertation, we study two fundamental distributed resource allocation problems that arise in the ad hoc network setting: topology organization and transmission scheduling.

Topology organization is studied in the first part of the dissertation. We consider ad hoc networks where multiple channels are available and defined by distinct frequency hopping sequences. Multi-channel systems can increase throughput by assigning simultaneous co-located transmissions to different communication channels. However, hosts must first synchronize their frequency hopping and transmission/reception patterns before any communication can take place. Due to this lack of initial synchronization, neighborhood discovery and network formation become non-trivial and time-consuming

processes. To address these issues, we first devise a symmetric technique where two nodes use a randomized schedule to synchronize and establish a link in minimum time. This method forms the basis of a distributed topology construction protocol that starts with a set of non-synchronized nodes and quickly forms a multi-channel ad hoc network satisfying certain connectivity or throughput requirements.

The second part of this dissertation introduces a novel distributed transmission scheduling framework for provision of Quality of Service (QoS) guarantees in wireless ad hoc networks. Due to the multi-access nature of the wireless medium, the perceived QoS in ad hoc networks depends heavily on the underlying medium access protocol. Such a protocol must use local information and coordinate transmissions so that bandwidth is shared in a controlled fashion. Fulfilling both requirements is a well-known problem with no satisfactory solutions to date. Random access methods-like that used in the 802.11 standard-use local information at the expense of unpredictable transmission conflicts and lack strict allocation guarantees. On the other hand, scheduled access methods–like Time Division Multiple Access (TDMA)–achieve deterministic allocations via perfect coordination of transmissions, but typically rely on two restrictive assumptions to reach their goal: network-wide slot synchronization and global knowledge of network topology and traffic requirements.

We first relax on network-wide slot synchronization and study asynchronous TDMA ad hoc networks. In these systems, each link uses a different local time slot reference provided by the hardware clock of a node endpoint. We introduce a framework of conflict-free scheduling and bandwidth allocation for such systems. Inevitably, slots will be wasted when nodes switch time slot references. This restricts the rate allocations that can be supported had the ad hoc network been perfectly synchronized. We show that the performance degradation due to lack of synchronization can be significant

and propose scheduling algorithms for overhead minimization that also have guaranteed upper bounds on the generated overhead.

We then introduce an asynchronous TDMA architecture for reaching global QoS objectives using only local information. The QoS objective is a set of link rates to be realized by a slotted network TDMA schedule where at each slot, several transmissions occur such that no conflicts occur at the intended receivers. Using only local information, nodes asynchronously adjust the rates of their adjacent links by local slot reassignments. The core idea is to modify the TDMA schedule online in a continuous and incremental manner until the QoS objective is reached. The incremental property allows for natural adaptation to changes in network topology or traffic requirements.

The TDMA architecture consists of a QoS-aware distributed bandwidth allocation algorithm and a generic distributed coordination mechanism. The bandwidth allocation algorithm determines the amount of link rate adjustments for steering the network to the desired objective. The coordination mechanism ensures that the local modifications on the schedule maintain its conflict-free property.

We first introduce a bandwidth allocation algorithm aiming for the max-min fairness objective. In this case, the optimal link rates are not known but are computed along with the schedule modification process. Analysis and experiments show that the proposed scheme has very good properties in tracking the optimal even in regimes of constant topology changes. We then extend the bandwidth allocation framework for the provision of rate guarantees to multi-hop sessions. Both Constant Bit Rate (CBR) and Available Bit Rate (ABR) services are considered. We show that CBR service can be provided using simple admission control rules and QoS routing mechanisms, similar to wireline networks; for ABR service, we introduce an asynchronous distributed algorithm for computing session max-min fair rates. The session rates computed by the end-to-end

bandwidth allocation algorithm are translated to link demands that must be enforced using a TDMA schedule. We solve this dynamic link scheduling problem for the special case of tree topologies and provide upper bounds on convergence delay.

An important feature of both our topology organization and transmission scheduling techniques is that they are amenable to distributed implementation on existing wireless technologies. To this end, we present an implementation and performance evaluation over Bluetooth, a wireless technology that enables ad hoc networking applications.

Distributed Topology Organization and Transmission Scheduling in Wireless Ad Hoc Networks

by

Theodoros Salonidis

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2004

Advisory Committee:

Professor Leandros Tassiulas, Chairman/Advisor
Professor Anthony Ephremides
Professor Sennur Ulukus
Professor Richard La
Professor A.Udaya Shankar

DEDICATION

To my grandmother, Eleftheria

my sister, Maria

my parents, Nikos and Anastasia

ACKNOWLEDGEMENTS

Thank you to my research advisor, Professor Leandros Tassiulas. His remarkable insight contributed substantially to the results of this dissertation; his high standards of intellectual integrity inspired me to continually challenge myself. Despite his return to Greece during the last three years of work on my dissertation, Leandros continued to provide insightful guidance and support on both academic and professional matters. I will always be grateful to him for having faith in my abilities and for helping make my first significant research experience extremely enjoyable.

I am thankful to Professors Anthony Ephremides, Sennur Ulukus, Richard La and Udaya Shankar for reviewing my dissertation and kindly consenting to serve on my defense committee. My special thanks to Professor Ephremides for his constructive criticism and advice. His early work on TDMA scheduling in ad hoc networks was one of the factors that inspired the choice of my dissertation topic.

My work on distributed topology organization for ad hoc networks–the first part of my dissertation–was completed during my one-year internship at the IBM T.J. Watson Research Center. I owe a debt of gratitude to my supervisors and colleagues there: Dr. Mahmoud Naghsineh, then head of the Pervasive Computing division, for his constant encouragement and guidance in what turned out to be a fruitful area of research; Dr. Richard Lamaire for numerous discussions on random graphs and optimization; Dr. Chatschik Bisdikian for sharing his authoritative knowledge on the Bluetooth standard;

and Stefan Berger for his invaluable help on implementation issues related to the IBM Bluetooth stack. My special thanks go to Dr. Pravin Bhagwat, my mentor at IBM and very good friend. Pravin's vast knowledge on networking and his practical yet precise approaches to solving complex problems set an example on how to perform systems research.

As a graduate teaching assistant, I was introduced to and discovered the joy in teaching. The commitment and enthusiasm shown by Professors Andre Tits and Eyad Abed for their profession have influenced me to consider pursuing a career in academia.

The bureaucracy at UMD can make life difficult for graduate students but I had little trouble due to the efforts of Elizabeth El Khodary, Frank Briggs and Maria Hoo. Their friendly attitude and sense of humor made it easy to cope with any and all administrative issues.

During my stay at Maryland I was fortunate to meet and befriend several interesting people that helped maintain the much-needed balance between research life and social life. It would be virtually impossible to list everybody here–I thank them all and wish them all the best in their future plans.

I would like to especially thank the members of Digenis, the Greek Student Association of UMD, for creating a reference point that made me feel at home. The stimulating atmosphere in 3187 AVW with Majid Raissi-Dehkordi, Tassos Michail and Kyriakos Manousakis made it a pleasure to work there every day. I would like to thank them and Jan Neumann for the inspiring technical and not-so-technical discussions.

Special thanks to my housemates at Stewart Court–Nikos, Tasos, Iordanis, Kyriakos, Laurent, Ermos, George, Dimitri and Andy–for sharing the house harmoniously and for being there in both good and bad times. To Grace Santos for showing me what scrutinous editing really means. And to Thanasis Kyparlis, George Apostolopoulos

and Lampros Kalampoukas who constantly provided me with a view of the networking industry world–this helped keep my research in due perspective.

Words cannot express my most sincere gratitude to my parents, my sister and my grandmother who send me their love and support from back home. They are the main reasons I have been able to reach this point. This dissertation is dedicated to them.

This work would not have been possible without the financial support of IBM Research and the Institute of Systems Research (ISR). My work at IBM Research was supported by an IBM University Partnership Award; at ISR, it was supported by a grant from the Air Force Office of Scientific Research (AFOSR), contract F49620-01-1-0197.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

A wireless ad hoc network consists of a set of geographically dispersed wireless nodes that can spontaneously form an all-wireless infrastructure without the need for any centralized administration. In such a network all nodes may be mobile, transmit on a shared wireless medium and act as routers forwarding packets of other nodes toward the intended destinations.

The ad hoc network concept is not new; it originates from the early DARPA packet radio network (PRN) and Survivable radio network (SURAN) projects in the 1980's [1][2][3] [4]. Since that time, the dynamic self-configuring nature of ad hoc networks has attracted attention for several military applications in all sectors, including the Army [5] [6] [7] [8], the Navy [9] or the Air Force [10]. Due to the rapidly growing user demand for wireless access, ad hoc networks have also started to appear recently within the commercial sector in various forms and scales. As "mesh networks", they are currently being considered as complements or alternatives to cellular networks for broadband wireless data access [11] [12] [13][14]. This is due to their minimal deployment cost and their potential to increase capacity and offer better robustness as more users are added to the network. Sizewise, mesh networks can be found between personal area

networks (PANs) and sensor networks. PANs are short-range ad hoc networks spontaneously formed by lightweight mobile devices to perform an interactive or collaborative task [15][16]. At the other extreme, sensor networks may consist of thousands of tiny inexpensive nodes deployed in an area to perform various sensing and collaborative processing tasks. Sensor networks are envisioned to operate unattended in diverse environments for extended periods of time [17][18].

While an ad hoc network shares the internet distributed communication paradigm and its exciting visions and applications, it also differs in some fundamental aspects: available bandwidth is scarce, the wireless medium is shared, nodes may have power and memory limitations, and the network topology can be highly dynamic. In view of these restrictions, classic problems of wireline data networks, including resource allocation and routing, become more difficult and require a fresh treatment. In addition, topology organization and mobility management are issues unique to this environment. In this dissertation, two fundamental problems related to the performance of an ad hoc network will be addressed: topology organization and transmission scheduling.

The aim of topology organization is to form and maintain a communication infrastructure from a set of geographically dispersed wireless nodes. In order to communicate, nodes must first be able to discover other nodes in proximity. The discovered topology can then be further controlled by transmission power adjustment or channel partitioning. Topology organization plays a key role in the performance of routing or transmission scheduling protocols used in the ad hoc network. A wrong topology may considerably reduce network capacity, increase end-to-end packet delay and decrease robustness due to node mobility and failures.

Given a network topology, transmission scheduling seeks to coordinate transmissions such that bandwidth is allocated to the entities competing for the wireless medium

according to a Quality of Service (QoS) objective. The entities may be nodes, links or multi-hop sessions. The QoS objective depends on the intended application. In some applications, traffic requirements are known in advance; in other applications the entities request fair service from the network.

The remainder of this chapter presents the two problems in more detail and introduces the basic terminology that will be used throughout the dissertation. The chapter concludes with a summary of our contributions.

1.1 Topology Organization in wireless ad hoc networks

Topology organization consists of two operations: neighborhood discovery and topology control. In neighborhood discovery nodes seek other nodes within proximity; in topology control the discovered topology is restricted and shaped according to certain performance criteria.

1.1.1 Topology Control

One way to exercise topology control in an ad hoc network is through transmission power adjustments. Consider a set of geographically dispersed wireless nodes. The ad hoc network topology depends on both the node locations as well as their transmission power levels. By increasing its power level, each node can reach a larger part of the network with a single transmission. However, this results in increased interference and higher energy expenditure. On the other hand, low power levels may result in a disconnected network. The problem of finding the minimum node power levels to maintain a connected topology has been addressed in [19][20]. Minimum power assignments for constructing and maintaining a multicast tree structure have also been considered [21].

In [22] each node is allowed to use different transmission power levels for each link; a method based on Delauney triangulation is used to select logical links in the network.

Given node transmission power levels, the ad hoc network can be represented as a visibility graph $G(N, E)$, where the vertices correspond to wireless nodes and the edges correspond to pairs of nodes that can hear each other. In addition to wireless proximity, the visibility graph also determines interference–the broadcast nature of the wireless medium induces location-dependent contention. Interfering co-located transmissions can be assigned to different *channels* to reach conflict-free the intended receivers. On the other hand, the same channel can be reused by transmissions that occur sufficiently apart in space. Channels can be defined in the time, frequency or code domains as time slots, frequency bands or spread spectrum codes (frequency hopping (FH) sequences or Direct Sequence (DS) codes), respectively.

Throughout this dissertation we will use the term ”channel” only for a frequency band or spread spectrum code; time slot assignments will be studied separately in the context of transmission scheduling. Topology control through channel assignment seeks to partition the visibility graph in multiple interconnected channels such that the resultant network topology–a subgraph of the visibility graph–satisfies specific performance objectives. Such objectives include connectivity (assuming the visibility graph is connected), energy efficiency or robustness to mobility; they may be sought under constraints such as maximum number of channels available in the network [23] and/or maximum number of participants per channel [24] [25][26][27][28] [29] [30][31].

A network operation related to topology control is clustering. The main purpose of clustering is to facilitate management of the ad hoc network by electing a certain node subset as ”clusterheads”. Clusterheads are vested with the most intensive network management tasks and coordinate operations within their cluster. A typical application of

clustering in ad hoc networks is hierarchical routing protocols [32][33][34]. Clusterheads are elected using distributed election algorithms. These algorithms may be based on local criteria such as node identities [35], node degree [36] or, more generally, node weights that reflect power reserve or mobility [37][38]. More sophisticated distributed election algorithms take into account constraints on cluster size [39] or cluster diameter [40]. Clustering differs from topology control in that it is not primarily intended to restrict the physical topology structure. However, it may facilitate the topology control operation by distributing it over the clusterhead nodes.

1.1.2 Neighborhood Discovery

The neighborhood discovery problem in ad hoc networks was introduced in [41][42] and subsequently addressed in [43][44][45]. In this problem, nodes need to coordinate their transmissions so that they discover their neighbors in minimum time. This resembles the transmission scheduling problem in that the nodes need to transmit in a shared channel. However, it differs in two main aspects: first, the nodes do not know their intended recipients; second, the emphasis is not on communication performance but rather on the delay of each node to discover its neighbors subject to all nodes performing the discovery protocol. Discovery delay can be defined as the time needed for all neighbors to successfully receive a discovery packet (asymmetric discovery) or the time needed for all nodes to acquire knowledge about each other (symmetric discovery).

In [41] we studied symmetric discovery for a pair of nodes using a channel implemented as a frequency hopping sequence. Multiple nodes were subsequently considered in [42]. Alonso et. al. [43] study symmetric discovery in a single frequency band. The frequency hopping channel and multiple node cases have also been studied in [44][45]. The analyses in [43][44][45] assume nodes perform discovery in synchronous rounds;

in [41][42] no synchronization is needed.

1.1.3 The challenge

So far, research on distributed topology organization has primarily focused on the topology control aspect–nodes start forming the topology, aware of their neighbors. The need for neighborhood discovery makes the problem more difficult for two reasons: First, it requires algorithms operating in an incremental manner. Second, it introduces delay as an additional performance objective. Hence, in the topology organization problem we will seek efficient topologies that must also be formed in minimum time.

1.2 Transmission scheduling in wireless ad hoc networks

1.2.1 Interference constraints and traffic models

The ad hoc network is represented by a visibility graph $G(N, E)$. We define interference in terms of transmissions occurring within a single channel, defined as a frequency band or spread spectrum code. Within a channel a *collision* occurs when multiple transmissions arrive simultaneously at a receiver. We assume that the radios do no support capture–upon a collision, all received transmissions are lost[1].

Due to cost restrictions, each node in an ad hoc network typically has a single communication transceiver and hence is unable to transmit and receive simultaneously. This hardware constraint together with location-dependent contention gives rise to the notions of primary and secondary interference. Primary interference occurs if a node is scheduled to transmit and receive simultaneously, or if a node receives simultaneously

[1]Capture refers to the ability of some radios to recover the strongest out of a set of simultaneously arriving transmissions.

from multiple transmissions intended to it. Secondary interference is due to unintended broadcast transmissions. It arises when a receiver R tuned to a particular transmitter $T1$ is within range of another transmitter $T2$ whose transmissions, though not intended for R, collide with the intended transmissions of $T1$. Figure 1.1 illustrates the different manifestations of primary and secondary interference.

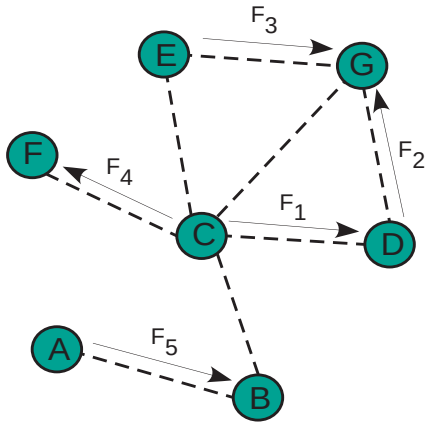


Figure 1.1: Dotted lines denote wireless proximity. The arrows denote intended point-to-point transmissions (flows). Primary interference occurs if any flows adjacent to a node are activated simultaneously. For example F_1 and F_2 force node D to transmit and receive simultaneously. F_2 and F_3 result in node E receiving from two intended transmissions. Also, since a packet is destined to a single destination, node C cannot transmit on F_1 and F_4 simultaneously. Secondary interference: If flows F_1 and F_5 transmit at the same time, they will result in a conflict at node B, the receiver of F_5.

Depending on the intended neighbors of each node, three traffic models exist for single-hop transmissions. In the point-to-point traffic model, each transmission is intended for a single neighbor; in the broadcast traffic model each transmission is intended for all neighbors; in the multicast traffic model a transmission is intended for a subset of neighbors. Unicast (or multicast) data applications at higher layers are naturally

mapped to the point-to-point (or multicast) single-hop traffic models. In applications where control information needs to be quickly disseminated to the entire ad hoc network, the single-hop broadcast model is more suitable. In the broadcast traffic model, only primary interference exists because each node transmission is intended for all its neighbors, or equivalently, all transmissions occurring around a node are intended to this node. In the point-to-point and multicast traffic models both primary and secondary interference exist.

In this dissertation our primary focus will be on scheduling data traffic of unicast applications sharing the ad hoc network. Such applications are naturally mapped to the point to point traffic model, where the entities to be scheduled are links. Secondary interference can be mitigated by scheduling intended transmissions that satisfy the primary interference constraints on different channels. One way to achieve this is to assign a different channel to each link. However, as links may grow exponentially with the network size, they may easily outnumber the number of available channels. In addition, determining which channel to use for each link requires global topology information. These issues are addressed by associating each node with a unique channel; if each link is assigned the channel of one of the node endpoints, then all transmissions satisfying the primary interference constraints will occur in different channels. Bluetooth is a wireless technology that implements this method using spread spectrum signaling. Each node is associated with a unique frequency hopping (FH) sequence derived from its unique address. Upon link establishment, one of the node endpoints is assigned as master and the other as slave. The link is assigned the FH sequence of the master. Although not orthogonal, Bluetooth FH sequences have been shown to perform well in practice [46]. Interference can be further mitigated using distributed assignment mechanisms that minimize the number of FH channels per locality [42][45][25]. In [23] it

is shown that perfectly orthogonal access can be achieved if nodes within two wireless hops of each other are assigned orthogonal channels–if D_{max} is an upper bound on the intended adjacent links per node, a total of $2D_{max}(D_{max} - 1) + 1$ (instead of N) channels are needed. References [23][47] propose distributed dynamic algorithms performing such assignments. An alternative (and potentially more expensive) technique to using multiple channels for mitigating secondary interference is to use a single channel and directional antennae on the intended communication links.

We will call systems that tolerate secondary interference *multi-channel systems*, while systems where both primary and secondary interference exist *single-channel systems*. When we refer to multi-channel systems, we will also assume that different channels are orthogonal–transmissions on a channel are correctly received by a node listening on that channel despite any in-range transmissions that may be occurring at different channels. Notice that, in addition to suppressing secondary interference, multi-channel systems can also perform topology control: channel assignments to selected links can restrict the network visibility graph in a desirable manner. As will be demonstrated later, this feature will play a key role in determining the network capacity used for provision of QoS guarantees.

1.2.2 Medium Access Control Protocols

Transmissions in the time domain are coordinated by a Medium Access Control (MAC) protocol. MAC protocols are based on either random (or contention-based) access or Time Division Multiple Access (TDMA).

Random access protocols typically operate over a single channel for which the nodes compete. Each node bases its transmission decisions on carrier sense and random backoff in case the channel is sensed busy. Representative examples are ALOHA [48],

CSMA [49], or the Distributed Coordination Function (DCF) of the IEEE 802.11 standard for Wireless LANs [50]. The attractiveness of random access protocols for ad hoc networks lies in ease of implementation–nodes base their transmission decisions only on local information. While they work well under light traffic, under heavy traffic they may incur high delays and low throughput. There is currently intense research effort for improving the performance of random access protocols; however, by nature, such protocols cannot be used for the provision of strict bandwidth allocation guarantees.

Time Division Multiple Access (TDMA) is based on a different philosophy. The ad hoc network operates according to a slotted schedule of period T_{system} slots. During each slot transmissions are scheduled such that no conflicts occur at the intended receivers. Depending on whether a broadcast or point-to-point traffic model is sought, the scheduled entities can be nodes or links, respectively. The bandwidth allocated to each entity equals the number of slots it receives during the schedule period.

Due to its conflict-free nature TDMA provides better utilization of the wireless medium compared to random access. Since the scheduled entities can be either nodes or links both broadcast and point-to-point traffic can be supported naturally and efficiently. In addition, TDMA can support multiple channels at a lower cost. Consider a multi-channel ad hoc network where each node has the capability of transmitting to or receiving from one channel at a time due to the single-transceiver constraint. If a random access MAC protocol is used, in addition to the transmit/receive uncertainty, a node must be able to decide which channel to use. Due to this constraint most multi-channel random access MAC implementations require multiple communication transceivers per node (equal to the maximum number of channels the node participates)[51]; this increases system cost. In TDMA only a single transceiver is needed: at every slot in the TDMA schedule, each node knows on which channel to transmit or receive.

1.2.3 TDMA challenges

The two major advantages of TDMA over random access are the ability to provide bandwidth allocation guarantees and conflict-free access to the wireless medium; however, exploiting these advantages in the distributed ad hoc network setting has been a notoriously challenging task.

Bandwidth allocation guarantees

Provision of bandwidth allocation guarantees typically requires global network topology information and a priori knowledge of traffic requirements. To make this argument more concrete we review the TDMA approaches for QoS routing in ad hoc networks. Consider a set of unicast multi-hop sessions sharing the ad hoc network with traffic requirements expressed in slots. Assume for the time being that the routes are fixed. To support τ_i slots for session i, each intermediate link over the route of this session must support τ_i slots. Hence, the demands of all routed sessions determine a set of link slot demands to be realized by a TDMA schedule. Since the ad hoc network operates with a period equal to T_{system} slots, we ask whether there exists a schedule of length less than T_{system} slots that can realize the link demand allocation. To answer this feasibility question for any given link demand allocation, we must be able to compute a minimum-length schedule. This problem is NP-complete for both single channel [52] and multi-channel TDMA systems [53]. Efficient centralized heuristics have been proposed in [54][55][56].

In practice the sessions and their routes are not all given in advance; in a more realistic model sessions arrive one at a time. In this case, we need to find a route that supports the traffic requirement of each incoming session. A candidate route can be tested by 1) increasing the current link loads over the route by the session slot require-

ment and 2) using the heuristics in [54][55][56] to compute a TDMA link schedule of short length. The session will be admissible over the candidate route if the schedule length does not exceed T_{system} slots. Two problems arise here. First, the computation of the TDMA schedule requires global topology information. Second, if the session is admitted, the new TDMA schedule must be disseminated to the entire network. Hence, this centralized approach is not practical for the distributed ad hoc network setting.

Zhu and Corson [57] and Lin [58] use a different approach for QoS routing. Instead of computing a new network-wide TDMA schedule for each incoming session, they fix the slot positions of existing sessions and, for each route, they seek the maximum available number of slots subject to the fixed slot positions in the route. This problem is also NP-complete even if global information is available. However, this approach allows distributed heuristics for computing available bandwidth and reserving slots over a route. Other distributed QoS routing approaches for ad hoc networks focus on mobility issues but do not take into account the MAC layer [59][36][60].

QoS routing operates according to a Constant Bit Rate (CBR) service model where sessions arrive with known bandwidth requirements. However, some applications have no more specific requirements than asking for the maximum possible bandwidth from the network. In this case, it is intuitive to allocate bandwidth according to a fairness objective. The algorithms in [54][55][56][57][58] cannot be used for fair allocations because they are specific to admission control. Sarkar and Tassiulas propose a distributed algorithm for computing max-min fair rates for multi-hop sessions in multi-channel wireless ad hoc networks [61]. However, the schedule computation that enforces these rates requires global topology information.

Conflict-free access

Mobility and traffic dynamics in an ad hoc network require a mechanism to ensure the TDMA schedule remains free of transmission conflicts. A common technique is to split the schedule of T_{system} slots in a control part of $T_{control}$ slots and data part of T_{data} slots. During the control part the network TDMA schedule is reorganized, i.e. the nodes exchange control information and reassign slots to update their transmission schedules in a consistent manner. Then the data part uses the new schedule for the actual data transmissions. This technique introduces a fixed control overhead equal to $T_{control}/T_{system}$. It also requires a priori knowledge of two types of global information:

1. **Network-wide slot synchronization:** all nodes know when a slot starts.

2. **Universal slot enumeration:** all nodes know the slot boundaries of the control/data parts.

Network-wide slot synchronization can be supported by equipping all nodes with GPS clocks or by using a protocol that periodically synchronizes the network (NTP). Such solutions are costly but may be justified in specialized applications (e.g. military missions); however they may not necessarily hold in more generalized ad hoc network settings (e.g. civilian applications) and may not even be supported by certain TDMA-based wireless technologies (e.g. Bluetooth).

Knowledge of universal slot enumeration in TDMA-based ad hoc networks is an issue that, to the best of our knowledge, has yet to be raised. In the distributed ad hoc network setting this information is not available in advance–when powered on, each node only knows the slot enumeration of its own local schedule. A common slot enumeration might be established by electing a leader node that provides its local slot enumeration as a reference to the rest of the nodes in the network. Such a distributed leader elec-

tion protocol would need to run continuously because network dynamics (node power-ons/power-offs and mobility) may generate multiple slot enumeration references in the network. Upon detection of such an event, the nodes must suspend communications until the election protocol converges to a common slot enumeration. This would be clearly inefficient in a mobile setting.

Even with global synchronization and enumeration, further difficulties are associated with splitting the TDMA frame into control and data parts. The control part may use either a TDMA or a contention-based mechanism. In the first case the control part consists of $T_{control} = N$ slots, where N is the number of nodes in the network. At slot i of the control part, node i transmits and all other nodes listen [9][58]. This approach provides a natural order for the nodes to perform scheduling decisions and ensures that control information will be exchanged conflict-free during the control portion. However, it requires a priori knowledge of the number of nodes in the network. In addition, large network sizes imply a large control part with respect to the data part–hence it is not scalable. Alternatively, $T_{control}$ can be fixed and independent of the network size; nodes compete during the control slots using a random access protocol (e.g. slotted ALOHA). In this case, the control messages may collide and there are no guarantees that the intended schedule reorganizations will occur during the control part. Clearly there is a performance vs. overhead trade-off associated with the choice of $T_{control}$ in this case; an appropriate value can be determined using simulations [62].

Summary

TDMA allows for provision of bandwidth guarantees but typically requires global information to achieve this goal. While distributed TDMA protocols for supporting CBR service do exist, a flexible framework for supporting more general QoS objectives such

as fairness is a problem that has remained unaddressed. In addition, current techniques for maintenance of the TDMA schedule conflict-free property may generate excessive control overhead and rely on restrictive assumptions such as network-wide slot synchronization, global slot enumeration and knowledge of the number of nodes in the network. In the transmission scheduling part of this dissertation we aim to address these fundamental issues within the framework of a novel distributed TDMA architecture.

1.3 Contributions of this dissertation

The goal of this dissertation is to study distributed mechanisms for topology organization and coordination of transmissions in order to achieve global performance objectives.

In Chapter 2 we address an instance of the topology organization problem that arises in Bluetooth scatternets[41][42][63]. Bluetooth scatternets are multi-channel ad hoc networks where all channels (including the discovery channel) are implemented as frequency hopping sequences and communication channels have a limit on the number of participants.

Neighborhood discovery in a frequency hopping channel is particularly challenging because the nodes need to coordinate both in time and frequency. We first devise a symmetric link establishment protocol where two nodes try to discover each other using a schedule that alternates between two complementary sender/receiver states. We show that if the schedules are deterministic the mean discovery delay can be arbitrarily large; for randomized schedules we show that the mean and standard deviation of the discovery delay are finite and derive analytical expressions given distributions on the schedule state residence times. We then use the link establishment protocol as a basic building block of a topology construction protocol. In addition to ensuring network connectivity

subject to the Bluetooth channel participation constraints, the protocol offers the flexibility to realize topologies with additional desirable properties such as minimization of the number of channels used in the network.

Chapters 3 through 6 focus on the transmission scheduling problem. In Chapter 3, we relax the global slot synchronization assumption and introduce an asynchronous TDMA communication model, where slot reference for each link is provided locally by the hardware clock of one of the node endpoints [64]. We study the overhead introduced when nodes switch among multiple time references and propose algorithms for overhead minimization.

Chapter 4 introduces and analyzes a distributed asynchronous TDMA protocol for multi-channel ad hoc networks where nodes reassign slots on their adjacent links in response to asynchronous events triggered by a higher layer. The protocol can be executed in parallel and can maintain conflict-free transmissions in the network. The TDMA protocol can also provide bandwidth guarantees by incrementally reaching a TDMA schedule that realizes a set of slot demands on the network links. We derive local conditions the nodes can use to generate demands on their adjacent links so that the induced global link demand allocations are always feasible.

The local conditions provide a subset of feasible allocations over which any QoS objective can be defined and enforced. In Chapter 5 we consider the max-min fairness (MMF) objective for the network links [65]. We introduce a fluid bandwidth allocation algorithm that computes the MMF rates while, at the same time, guides slot reassignments in the distributed TDMA protocol to reach a schedule that enforces these rates.

In Chapter 6, a framework is introduced for provision of rate guarantees to multi-hop sessions [66]. This framework consists of two components: an end-to-end bandwidth allocation algorithm that allocates rates to the sessions according to a QoS objective and

a link scheduling algorithm that reaches a TDMA schedule enforcing these rates. For the end-to-end bandwidth allocation component we introduce algorithms for allocating bandwidth according Constant Bit Rate (CBR) and Available Bit Rate (ABR) service objectives. For the link scheduling component, we solve the dynamic link scheduling problem for tree networks and provide upper bounds on convergence delay. Both end-to-end and link scheduling algorithms rely only on local information.

Chapter 7 contains a summary of the major contributions of this dissertation along with some suggestions for further work.

Chapter 2

Distributed topology construction of Bluetooth Wireless Personal Area Networks

Most experimental ad hoc networks to date have been built on top of single-channel, broadcast-based 802.11 wireless LANs or IR LANs. In such networks, all nodes within direct communication range of each other share a common channel using a CSMA MAC protocol. In addition, multi-hop routing is used as a means for forwarding packets beyond the communication range of the source's transmitter. Since a single channel is used throughout the network, the topology of the ad hoc network is implicitly (and uniquely) determined by distance relationship among the participating nodes.

We aim to address a problem that arises when multiple channels are available for communication in an ad hoc network. The problem is determining which subgroup of nodes should share a common channel and which nodes should act as relays, forwarding traffic from one channel to another. The channel assignment should be performed so that all constraints posed by the underlying physical layer are satisfied, while ensuring that the resultant topology is connected.

We address an instance of the above problem which occurs in Bluetooth-based ad hoc networks, known as scatternets. Bluetooth is a promising technology that aims to

support wireless connectivity among cell phones, headsets, PDAs, digital cameras, and laptop computers. Initially, the technology will be used as a replacement for cables, but in due time, solutions for point-to-multipoint and multi-hop networking will evolve.

Bluetooth is a frequency hopping system which defines multiple channels for communication (each channel defined by a different frequency hopping sequence). A group of devices sharing a common channel is called a piconet. Each piconet has a master unit which selects a frequency hopping sequence for the piconet and controls access to the channel. Other participants of the group, known as slave units, are synchronized to the hopping sequence of the piconet master. Within a piconet, the channel is shared using a slotted Time Division Duplex (TDD) protocol where a master uses a polling protocol to allocate time-slots to slave nodes. The maximum number of slaves that can simultaneously be active in a piconet is seven.

Multiple piconets can co-exist in a common area because each piconet uses a different hopping sequence. Piconets can also be interconnected via bridge nodes to form a larger ad hoc network known as a scatternet. Bridge nodes are capable of timesharing between multiple piconets, receiving data from one piconet and forwarding it to another. There is no restriction on the role a bridge node can play in each piconet it participates in. A bridge can be a master in one piconet and slave in others (M/S bridge) or a slave in multiple piconets (S/S bridge).

It is possible to organize a given set of Bluetooth devices in many different configurations. Figures 3.1(b) and 3.1(c) show two example configurations in which nodes in a Bluetooth network can be arranged. All nodes are assumed to be in radio proximity of each other. In Figure 3.1(b) all nodes are part of a single piconet. Figure 3.1(c) illustrates another configuration where node A is master of piconet 1, node E is master of piconet 3, node B is an M/S bridge (master of piconet 2 and a slave of piconet 1), node

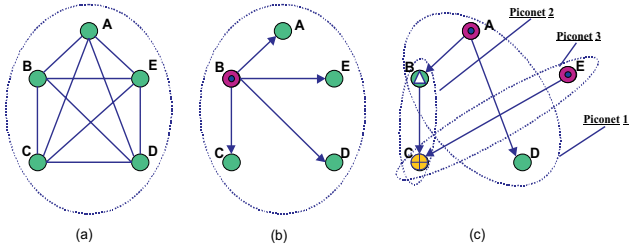


Figure 2.1: (a) Single channel topology. (b),(c) Different configurations according to the Bluetooth multi-channel topology model.

D is a slave of piconet 1 and node C is an S/S bridge (slave in piconets 2 and 3). In contrast to these scatternet configurations the node interconnection topology in a single channel system will be a complete graph (Fig. 3.1(a)) since all nodes will hear each other's transmissions.

Given a collection of Bluetooth devices, an explicit topology construction protocol is needed for forming piconets, assigning slaves to piconets, and interconnecting piconets via bridges such that the resulting scatternet is connected. Such a protocol should be asynchronous, distributed and may start with nodes not having any information about their surroundings.

The problem of constructing distributed self-organizing networks has been addressed in the past [35][9][67][36][68] [19] [69]. All approaches assume existence of a broadcast channel through which neighborhood or control information can become available. The Bluetooth setting introduces two unique challenges: first, no broadcast channel exists for facilitating the exchange of any control information, including proximity information; second, even if proximity information is available, the piconet membership constraint renders the formation of a connected topology a very challenging task.

The scatternet formation problem was introduced in [24] and subsequently addressed in [25][26][27][28] [29] [30][31]. Degree-constrained scatternet formation for multi-

hop topologies has been investigated in [25][27][28][31]. The problem is NP-complete for some instances and can be solved by a polynomial algorithm under certain assumptions [31]. All proposed solutions are distributed: starting with the sole knowledge of their one-hop neighbors, the nodes perform role assignments on their adjacent links to reach a connected topology that satisfies the Bluetooth connectivity requirements.

The scatternet formation problem becomes significantly harder if nodes start with no knowledge about their surroundings. The discovery channel is a frequency hopping sequence; nodes in proximity need to synchronize both their timing and frequency hopping patterns before being able to communicate. In this setting, even the formation of individual links becomes an issue—delays are random and can be arbitrarily large if no proper measures are taken.

We introduce and analyze a randomized symmetric protocol that yields link establishment delay with predictable statistical properties. Such a protocol is necessary for pairs of identical devices or in situations when any external means for selecting initial device states are not available. We then propose the Bluetooth Topology Construction Protocol (BTCP), an asynchronous distributed protocol that extends the point-to-point symmetric mechanism to the case of several nodes. BTCP is based on a distributed leader election process where proximity information is discovered in a progressive manner and eventually accumulated to an elected coordinator node. Given a view of the topology, the coordinator can then use a centralized algorithm to form a connected scatternet topology.

We present a version of BTCP optimized for the single-hop case (i.e. all nodes are within wireless range of each other). This is a valid assumption for Wireless Personal Area Networks (WPANs), currently considered by the IEEE 802.15 standard [70]. Compared to other forms of ad hoc networks, such as Mobile Ad Hoc Networks (MANETs)

or sensor networks, WPANs are characterized by a relatively small number of low-power devices operating within a limited geographic area (e.g. a conference room). In addition to connectivity, WPAN applications require scatternet formation in a short amount of time that is tolerable by a human user.

Zero-knowledge distributed scatternet formation has also been addressed in [26][29]. Similar to BTCP, the protocols are distributed and are targeted for single-hop environments. However, they construct and re-arrange the scatternet topology as links are discovered. The protocol of Law et.al. [26] constructs bipartite topologies while the protocol of Tan et.al. [29] focuses on the construction of tree topologies. Compared to [26][29], BTCP is more flexible in constructing the topology because it uses a centralized algorithm for the role assignment phase.

The remainder of the Chapter is organized as follows: Section 2.1 introduces the asymmetric link establishment protocol as defined by the Bluetooth Specification. In Section 2.2 we propose and analyze the symmetric link establishment protocol. Sections 2.3 and 2.4 describe the WPAN application requirements and detailed operation of BTCP, respectively. Since the total number of participants is not known, each node uses a timeout to assume leader election termination. The timeout introduces a correctness-delay tradeoff in the network formation. Using the delay analysis of Section 2.2 we show in Section 2.5 how to best choose the protocol parameters in order to maximize the probability of forming a connected scatternet while minimizing delays. Section 6.4 provides a detailed survey of the state-of-the-art in Bluetooth scatternet formation. Finally, Section 5.6 concludes the Chapter.

2.1 Link establishment in Bluetooth

Bluetooth link establishment is a two-step process that involves the Inquiry and Paging procedures [15]. Both procedures are asymmetric, involving two types of nodes that perform different actions: during Inquiry, senders discover and collect neighborhood information provided by receivers; during Paging, senders connect to previously discovered receivers.

When senders and receivers use the same (Inquiry or Paging) frequency hopping sequence[1], they will most likely start at different frequency hops derived from their local clock readings. To overcome this frequency uncertainty senders and receivers hop at different rates. A receiver changes hops slowly (every $1.28s$), listening for sender messages; a sender transmits at a much higher rate (every $625\mu s$) while listening in-between transmissions for an answer. The term Frequency Synchronization delay (FS delay) refers to the time needed until the sender transmits on which the frequency the receiver is currently listening[2].

The functional difference between the two procedures is that Inquiry uses a universal frequency hopping sequence while Paging uses a common point-to-point frequency hopping sequence. Using a universal frequency hopping sequence, a sender node effectively broadcasts an Inquiry Access Code (IAC) packet that can be heard by receiver nodes listening for such a packet. During the paging procedure, a sender uses a receiver's page hopping sequence and effectively unicasts a Device Access Code (DAC) packet to be heard only by this receiver. Hence, Inquiry involves many units where a

[1] N_f, the number of frequencies in the inquiry or page hopping set, is equal to 32 for systems operating in Europe and US and 16 for systems operating in Japan, Spain and France.

[2] The time needed by the sender to cover the entire inquiry hopping frequency set is $T_{coverage} = N_f \times 625\ \mu s$ which is $10\ ms$ ($20\ ms$) for the 16 (32) hop system.

sender can discover more than one receiver while Paging involves only two units where a sender pages and connects to a specific receiver.

2.1.1 The Asymmetric Protocol

The asymmetric Bluetooth link establishment protocol (Fig. 2.2) begins by the sender entering the INQUIRY state and the receiver entering the INQUIRY SCAN state. After an initial FS delay, the sender transmits on the frequency hop the receiver to which is listening. Upon reception of the IAC packet, the receiver sleeps for a random time interval (called RB delay), uniformly distributed between 0 and $r_{max}(= 639.375ms)$. The random back-off is performed to avoid collision at the sender in case two or more receivers were listening on the same frequency hop and responded simultaneously.

When the receiver wakes up, it tunes to the hop it was listening before the back-off occurred. After a second FS delay, an IAC packet is received; the receiver replies with an FHS packet and starts listening on its page hopping sequence by entering the PAGE SCAN state. The FHS packet contains the identity and clock of the receiver. Upon reception of the FHS packet, the sender initiates the Paging procedure by entering the PAGE state. The identity and clock in the FHS packet are used to determine the receiver's page hopping sequence and current listening hop, respectively. Thus, when paging follows inquiry, the FS delay is eliminated and the sender transmits a DAC packet on the receiver's listening hop.

The remaining control messages are exchanged in consecutive slots of $625\mu s$ each. The receiver replies with a DAC packet. The sender then transmits a FHS packet to let the receiver determine its channel hopping sequence and phase. The receiver acknowledges with another DAC packet and becomes the link slave. As soon as the sender receives the DAC acknowledgment, it becomes the link master. After an ad-

ditional POLL/NULL packet exchange, the synchronized nodes may start exchanging data. Figure 2.2 illustrates the components of the overall protocol delay. The Inquiry

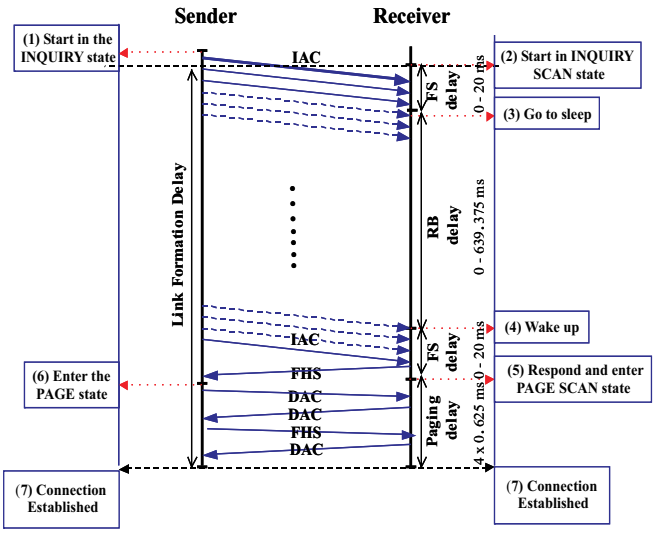


Figure 2.2: The Bluetooth asymmetric link establishment protocol

.

delay consists of one RB delay and two FS delays. Since the FS delay is bypassed when paging follows inquiry, paging delay (6 slots, $625\mu s$ each) is assumed negligible. Thus, the overall delay of the asymmetric link establishment protocol can be approximated by:

$$R = 2FS + RB \tag{2.1}$$

where FS and RB are uniform random variables in $[0, T_{coverage}]$ and $[0, r_{max}]$, respectively.

2.2 A symmetric link establishment protocol

The asymmetric protocol yields a short link establishment delay[3] provided that the sender and receiver roles are pre-assigned. This may not be possible in an ad hoc network setting. For example, in a ”conference room” scenario, users are not able to explicitly assign sender and receiver roles on their devices. They just press a button and expect to connect with their peers.

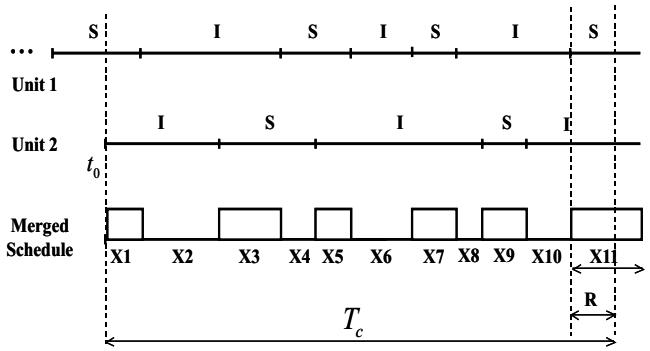


Figure 2.3: The symmetric link establishment protocol: Each node alternates independently between INQUIRY(I) and SCAN(S) state. Connection can be established only during the intervals where nodes are in opposite states. The time interval T_c from t_0 up to the point where the two units are in opposite states for a sufficient amount of time is the link establishment delay.

Links can be automatically established using the following symmetric mechanism: When a node is powered on, it arbitrarily assumes sender or receiver role by entering the INQUIRY or INQUIRY SCAN state, respectively. The node remains in the selected state for a period of of time. If during this time no connection is established, it switches to the opposite state. State alteration continues until a connection occurs.

[3]According to eq. (2.1), the maximum delay of the asymmetric protocol is $r_{max} + 2 \cdot T_{coverage} = 639.375\ ms + 40\ ms = 679.375\ ms$ for the 32-hop system and 659.375 ms for the 16-hop system.

Nodes execute the protocol independently; they will be able to connect only during intervals where they are in opposite states. During such an interval, the asymmetric protocol is automatically executed. The sender will become aware of the receiver only when it receives the FHS packet after a random delay R (given by eq. (2.1)). If during this time the sender independently switches to the receiver state, connection will not occur. On the receiver end, the reception of the IAC packets, back-off activity and transmission of FHS packets are not communicated to the upper layers of the Bluetooth stack. Since we have only access to the upper layers and because we need to devise a symmetric protocol without modifying the Bluetooth Specification, we assume the receiver becomes aware of the sender only after paging and link establishment have occured.

The symmetric protocol operation is depicted in Figure 2.3. During each ”on” interval X_n, the asymmetric protocol restarts execution. Connection is established only if the generated random delay R_n is less than X_n. Since R is random, the number of ”on” intervals needed until connection will be random. Therefore, the symmetric protocol is expected to have a random delay, typically greater than the delay of the asymmetric protocol.

Some interesting questions arise regarding the performance of such a symmetric protocol. Should the state residence intervals be constant or random? How can link establishment delay be minimized? First, assume the nodes switch states according to a schedule of period T. Since the state residence intervals are constant, the ”on” intervals of the merged process X_n in Figure 2.3 are also constant. For a specific protocol run, the ”on” intervals can be arbitrarily small and the unsuccessful executions of the asymmetric protocol can be many; the delay, then, will be arbitrarily large. Alternatively, if the state residence intervals are drawn from a random distribution of finite mean and variance,

the mean and variance of the symmetric protocol link establishment delay are finite and can be expressed analytically. More specifically, the following holds:

Theorem 2.2.1 *Let each node alternate states such that the state residence intervals form an i.i.d. random process Z_n with mean $E[Z]$ and variance $V[Z]$. If $E[Z]$ and $V[Z]$ are finite, the mean and variance of the link establishment delay T_c are finite and given by:*

$$E[T_c] = \frac{E[X]}{2} + \frac{(E[X|R > X] + E[X])(1-p)}{p} + E[R] \tag{2.2}$$

$$V[T_c] = \frac{V[X]}{2} + \frac{(V[X|R > X] + V[X])(1-p)}{p} + V[R] \tag{2.3}$$

where R is the random link establishment delay of the asymmetric protocol, X_n is the interval process formed by merging the state switching times of the two random schedules, and $p = P[R \leq X]$.

Proof Without loss of generality, assume node 1 starts alternating first, and node 2 starts alternating at an arbitrary time instant t_0 (Fig. 2.3). Let $N_i(t)$ be the number of state switches of node i from time t_0 up to time t. $N_i(t)$ is a renewal process induced by the i.i.d. interval process Z_n. Since the units alternate independently, $N_1(t)$ and $N_2(t)$ are independent. Consider the merged process $N(t)$ consisting of the combined state switches $N_i(t)$ from t_0 up to time t. The interval process X_n induced by $N(t)$ is i.i.d. with the following cdf [71]:

$$F_x(x) = F_z(x) + \frac{1 - F_z(x)}{E[Z]} \int_0^x [1 - F_z(z)]dz \tag{2.4}$$

Then the pdf of X_n is the derivative of $F_x(x)$ with respect to x:

$$f_x(x) = f_z(x) - \frac{f_z(x)}{E[Z]} \int_0^x [1 - F_z(z)]dz + \frac{(1 - F_z(x))^2}{E[Z]} \tag{2.5}$$

Depending on the time t_0 where node 2 starts alternating states, we consider two cases:

Case A: Let t_0 be such that the nodes start in opposite states. The nodes will have the chance to connect during odd-numbered intervals X_n. During each such "on" interval, the asymmetric protocol will restart execution from scratch. Connection will be established only if the random delay R_n of the asymmetric protocol is less than X_n. Since the random processes X_n and R_n are each i.i.d and independent with respect to each other, this is equivalent to a coin-toss experiment with probability of "connection-success" $p = P[X \leq R]$. Let the composite ("on"+"off") interval Y_n corresponding to a failure be defined as:

$$Y_n = \begin{cases} X_n + X_{n+1} & \text{if } R_n > X_n \\ 0 & \text{otherwise} \end{cases}, \; n = 2k+1, \; \forall k \geq 0 \tag{2.6}$$

The overall connection establishment delay T_c^{opp} is:

$$T_c^{opp} = \sum_{n=1}^{N} Y_n + R_{N+1} \tag{2.7}$$

where N is the number of failures until a success occurs and is geometrically distributed with parameter $p = P[R \leq X]$. Thus, the average link establishment delay when nodes start in opposite states can be computed as follows:

$$\begin{aligned} E[T_c^{opp}] &= E[E[T_c^{opp}]|N] + E[R] \\ &= \sum_{n=0}^{\infty} E[T_c^{opp}|N=n] \cdot P[N=n] + E[R] \\ &= \sum_{n=0}^{\infty} E[\sum_{i=1}^{n} Y_i] \cdot P[N=n] + E[R] \\ &= \sum_{n=0}^{\infty} n \cdot E[Y_i] \cdot P[N=n] + E[R] \\ &= E[Y_i] \cdot E[N] + E[R] \\ &= (E[X|R>X] + E[X]) \cdot E[N] + E[R] \Rightarrow \end{aligned}$$

$$E[T_c^{opp}] = \frac{(E[X|R > X] + E[X])(1-p)}{p} + E[R] \tag{2.8}$$

Case B: Let t_0 be such that the nodes start at the same state. The only difference with the previous case is that the first "off" interval introduces a constant delay factor on the overall delay. Therefore:

$$T_c^{same} = X + T_c^{opp} \tag{2.9}$$

where T_c^{opp} is given by eq. (2.7). Then,

$$E[T_c^{same}] = E[X] + E[T_c^{opp}] \tag{2.10}$$

Since t_0 is arbitrary, the cases A and B are equiprobable. Combining eq. (2.8) and eq. (2.10), we reach the desired expression for $E[T_c]$:

$$\begin{aligned} E[T_c] &= \frac{1}{2}E[T_c^{opp}] + \frac{1}{2}E[T_c^{same}] \\ &= \frac{1}{2}E[X] + E[T_c^{opp}] \Rightarrow \end{aligned}$$

$$E[T_c] = \frac{E[X]}{2} + \frac{(E[X|R > X] + E[X])(1-p)}{p} + E[R]$$

To derive the variance $V[T_c]$, observe that eq. (2.7) and eq. (2.9) are sums of independent random variables. In this case, the linearity of variance holds in the same way as linearity of expectation. Repeating the same calculation as in $E[T_c]$, we reach the desired expression for $V[T_c]$:

$$V[T_c] = \frac{V[X]}{2} + \frac{(V[X|R > X] + V[X])(1-p)}{p} + V[R]$$

The quantities $E[X]$ and $V[X]$ can be derived from eq. (2.5). The quantities $E[X|R > X]$ and $V[X|R > X]$ can be computed by first considering the conditional pdf of X

given that $X < r$:

$$f_x(x|x < r) = \begin{cases} \frac{f_x(x)}{F_x(r)} & \text{if } x < r \\ 0 & \text{otherwise} \end{cases}$$

Then,

$$\begin{aligned} E[X|X < R] &= E[E[X|X < r]] \\ &= \int_{r=0}^{A} \int_{x=0}^{r} x \cdot f_X(x|x < r) \cdot f_R(r) dx dr \Rightarrow \\ E[X|X < R] &= \int_{r=0}^{A} \int_{x=0}^{r} x \cdot \frac{f_X(x) \cdot f_R(r)}{F_X(r)} dx dr \end{aligned} \tag{2.11}$$

where $A = r_{max} + 2 \cdot T_{coverage}$. Also,

$$E[X^2|X < R] = \int_{r=0}^{A} \int_{x=0}^{r} x^2 \cdot \frac{f_X(x) \cdot f_R(r)}{F_X(r)} dx dr \tag{2.12}$$

where $A = r_{max} + 2 \cdot T_{coverage}$.

The conditional variance is given by:

$$V[X|X < R] = E[X^2|X < R] - (E[X|X < R])^2 \tag{2.13}$$

and can be computed using equations (2.11) and (2.12). ■

Equations (2.2) and (2.3) hold for any distribution of finite mean and variance. Figure 2.4 is a comparative plot of $E[T_c]$ as a function of the mean state residence interval for the cases of uniform and exponential distributions. Both distributions yield U-shaped curves. Very small and very large mean state residence intervals yield high delays. For very small state residence intervals, many short ”on” intervals are needed until connection occurs. For very large state residence intervals, the high delay is due to the uncertainty in the initial state assignment: if the nodes start at the same state, they will wait for a large ”off” interval before the first ”on” interval occurs. The exponential distribution yields a lower delay for large mean state residence intervals. However,

both distributions perform similarly in the minimum delay region: for a mean state residence interval of $600\ ms$ the average delay is approximately $1\ s$. This is approximately three times greater than the average delay of the asymmetric protocol given by eq. (2.1) ($\approx r_{max}/2 = 319.688ms$).

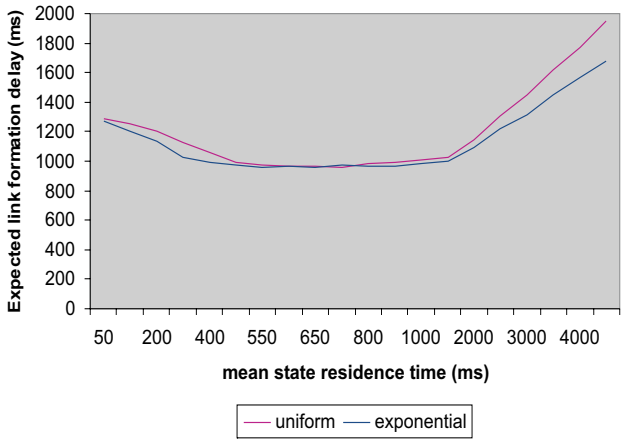


Figure 2.4: Symmetric protocol: Average link establishment delay for uniformly and exponentially distributed state residence intervals.

We have also investigated whether different mean state residence interval per state yields a lower delay. In this case we use simulations to determine $E[T_c]$. Figure 2.5 depicts $E[T_c]$ with respect to the INQUIRY mean state residence interval μ_{I}. Each "$\times N$" curve corresponds to the INQUIRY SCAN mean state residence interval μ_{S} being $N \times \mu_{\mathrm{I}}$. We observe that no benefit arises from using different mean state residence intervals: In the minimum delay region of all curves, the "$\times 1$" curve yields the lowest average delay.

The randomized symmetric mechanism guarantees automatic link establishment between two Bluetooth devices in finite mean time. When more than two devices need to form a scatternet a protocol must be devised on top of this mechanism. This protocol must yield a connected topology with high probability while doing so in minimum time.

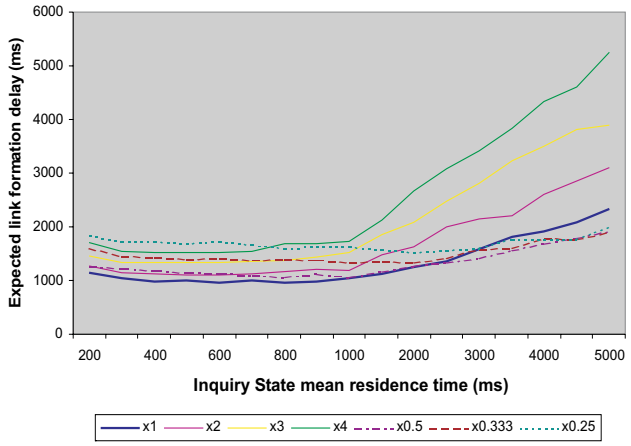


Figure 2.5: Symmetric protocol, uniform distribution: Delay for different mean residence intervals per state ($\mu_{\mathrm{S}} \neq \mu_{\mathrm{I}}$) vs. delay for equal mean residence intervals per state ($\mu_{\mathrm{S}} = \mu_{\mathrm{I}}$). Dotted curves correspond to ($\mu_{\mathrm{S}} < \mu_{\mathrm{I}}$), while solid curves correspond to ($\mu_{\mathrm{S}} \geq \mu_{\mathrm{I}}$).

The delay analysis of the point-to-point symmetric mechanism will provide a valuable tool for balancing these conflicting objectives.

2.3 Scatternet formation

Our motivation for the scatternet formation problem arises from a ”conference meeting” scenario. Suppose that several users wish to form an ad hoc network using their Bluetooth devices. Each user powers on his/her device and expects to see a ”network established” message after a short period of time. After this message appears, the user will be able to exchange information with every other user. The high-level description of this application embodies the elements of a successful scatternet formation protocol:

- Network establishment must be performed in a distributed manner. Each device must start operating asynchronously on its own without any prior knowledge of

the identities or number of nodes participating in the process.

- Network establishment delay must be tolerable by the end-user and minimized as much as possible.

- Upon completion, the protocol must yield a connected scatternet that satisfies the Bluetooth degree constraint of 7 slaves per piconet.

In addition to satisfying connectivity, a desirable protocol feature would be to shape the scatternet topology according to application-specific performance criteria. For example, a node may need to assume different roles in different application scenarios. Also, due to its own nature, a node may pose more restrictive degree constraints: a Palm Pilot may not have the processing power to be a master of a 7-slave piconet. Criteria may also exist in the form of traffic requirements to be satisfied by the nodes participating in the network construction process. Marsan et al. [72] have devised a centralized role assignment algorithm that minimizes the energy consumption of the most overloaded node subject to node traffic requirements. In absence of preexisting scatternet formation criteria, and in order to design a simpler and faster protocol, we propose the following default properties that the resulting topology will satisfy:

R1 **Each master may be connected to at most D slaves:** This condition restricts the number of participants of each piconet to $D + 1$. The Bluetooth specification requires $D = 7$.

R2 **Each node will be either master or slave on all its adjacent links:** The Bluetooth specification does not prevent a node being master in one piconet and slave in others (M/S bridge); However, M/S bridges may result in high delays: when the master visits other piconets as slave, no communication can occur in the piconet it controls.

Therefore, we use only S/S bridges to interconnect piconets. Note that with this restriction the resulting topology will be bipartite.

R3 **A bridge node will connect only two piconets:** A bridge node forwards data by switching between piconets in a time division manner. A portable device may have limited processing capabilities. A maximum degree of two relieves the bridge from being an overloaded crossroad of multiple originated data transfers. In addition, the slot overhead incurred by switching multiple piconet time references is minimized [73] [74].

R4 **Every piconet will be connected to *all* other piconets through S/S bridges:** A fully-connected scatternet in its initial state provides higher robustness against topology changes. Also, according to this property, no routing is needed: every master can reach every other master through a bridge node and every slave can reach every other node via its master in at most 3 hops.

R5 **Any two piconets will share only one bridge:** This condition seeks to minimize the total number of piconet interconnection points. Two masters may later use a topology maintenance protocol to share more than one bridges.

Given a number of nodes N, we seek the minimum number of piconets P_{min} that satisfy constraints R1-R5. The motivation for this objective is similar to finding the minimum number of routers in an ad hoc network [69]: A minimum number of piconets yields an easier scatternet to control.

We now proceed to the derivation of P_{min}. According to condition $R2$, the bipartite scatternet consists of masters, slaves that belong to only one piconet (”pure slaves”), and slaves that belong to multiple piconets (S/S bridges). In such a scatternet, the number of masters equals the number of piconets.

Let P be the number of piconets and let piconet i consist of s_i pure slaves and b_i bridge slaves for a total of n_i slaves:

$$n_i = s_i + b_i,\ 1 \leq i \leq P \tag{2.14}$$

Also, due to the piconet membership constraint $R1$:

$$n_i \leq D,\ 1 \leq i \leq P \tag{2.15}$$

According to $R4$ and $R5$, each master will have $b_i = P-1$ bridges and $s_i = n_i-(P-1)$ pure slaves. The total number of masters is P and, according to R4 the total number of bridges should be $\frac{P(P-1)}{2}$. Therefore, the following holds:

$$P + \sum_{i=1}^{P} s_i + \frac{P(P-1)}{2} = N,\ 0 \leq s_i \leq D-(P-1),\ \forall i \tag{2.16}$$

where the three terms at the LHS are the total number of assigned masters, pure slaves and bridges in the scatternet respectively.

Equation (2.16) represents the values for P and N that satisfy the scatternet formation requirements $R1 - R5$. For a specific P, there is a range of values of N that can be covered, depending on the possible values s_i. For example, a single piconet ($P = 1$) can accommodate from $N = 1$ up to $N = D + 1$ nodes. Two piconets ($P = 2$) can cover from $N = D + 2$ to $N = 2D + 1$ nodes–in this case, the two masters are connected by a common bridge and each master has $D - 1$ pure slaves.

According to eq. (2.16), the maximum N (denoted by N_{max}) for a given P can be obtained if we set $s_i = D - (P - 1),\ \forall i$. Then, eq. (2.16) becomes:

$$P + \sum_{i=1}^{P} (D - (P - 1)) + \frac{P(P-1)}{2} = N_{max} \Rightarrow$$

$$P^2 - (3 + 2D)P + 2N_{max} = 0 \tag{2.17}$$

Solving eq. (2.17) for N_{max} we get the maximum number of nodes that can be supported by a specific P without violating conditions R1-R5:

$$N_{max} = f(P) = \frac{P((3+2D)-P)}{2} \tag{2.18}$$

According to $R4$ and $R5$, each master must be connected to every other master via exactly one bridge node. Hence, the maximum number of piconets that can be supported is $P = D + 1$. In this case every master has D bridge slaves to all other masters.

Using $P = D + 1$ in eq. (2.18) yields $N_{max} = \frac{(D+1)(D+2)}{2}$, the maximum number of nodes yielding a topology satisfying conditions R1-R5. Using eq. (2.18) we generate the (ordered) set:

$$\boldsymbol{N}^{max} = \{f(1), ..., f(P), ..., f(D+1)\}$$

Also, solving eq. (2.18) for P and keeping the "-" solution we get:

$$P = f^{-1}(N_{max}) = \frac{(3+2D) - \sqrt{(3+2D)^2 - 8N_{max}}}{2},\ N_{max} \in \boldsymbol{N}^{max} \tag{2.19}$$

Since eq. (2.19) is the inverse function of eq. (2.18), for any value of N in the set $\boldsymbol{N}^{max}$, eq. (2.19) yields an integer P. Also, P is a strictly increasing (discrete) function of N_{max}. Any two consecutive numbers $N_{max1} = f(P_1)$ and $N_{max2} = f(P_1 + 1)$ in $\boldsymbol{N}^{max}$ correspond to two values P_1 and $P_1 + 1$ respectively. Since P is strictly increasing function of N, any values of N not in $\boldsymbol{N}^{max}$ in the ordered set $\boldsymbol{S} = \{N_{max1} + 1, ..., N_{max2} - 1\}$ that are used in eq. (2.19) will yield a real number between P_1 and $P_1 + 1$. Thus the values in the set $\boldsymbol{S}$, including N_{max2}, are the values of N supported by a number of piconets $P_1 + 1$. Hence, using any value of N in eq. (2.19) and rounding the resulting real number to the next integer will always yield the minimum number of piconets P_{min} that can support N:

$$P_{min} = \left\lceil \frac{(3+2D) - \sqrt{(3+2D)^2 - 8N}}{2} \right\rceil,\ 1 \leq N \leq \frac{(D+1)(D+2)}{2} \tag{2.20}$$

In the case of Bluetooth ($D = 7$), eq. (2.20) holds for N up to 36 devices; we believe this is a sufficiently large number for the envisioned WPAN application scenarios. Note that this restriction holds if we need to satisfy *all* criteria R1-R5. A larger number of nodes can be supported by either not requiring a minimum number of piconets or by relaxing one or more of conditions R1-R5.

2.4 The Bluetooth Topology Construction Protocol (BTCP)

BTCP is based on a leader election process. Leader election is an important tool for breaking symmetry in a distributed system. Since the nodes start asynchronously and without any knowledge of the number of participating nodes, an elected coordinator will be able to control the process and ensure that the resulting topology will satisfy the scatternet formation criteria. The protocol consists of 3 phases:

2.4.1 Phase I: Coordinator Election

Phase I consists of an asynchronous distributed election of a coordinator node that will eventually know the count, identities and clocks of all nodes participating in the topology construction process.

Each node has an integer variable called VOTES. Upon power-on, a node initializes VOTES to 1, and starts executing the symmetric link establishment protocol using a randomized schedule.

Any two nodes that discover each other and connect enter a one-on-one confrontation by comparing their VOTES. The node with the larger VOTES wins the confrontation. If the VOTES are equal, the winner is the node with the larger Bluetooth address. The loser provides the winner with all the FHS packets (i.e. identities and clocks) of

the nodes it has won thus far. Then, it disconnects and enters the PAGE SCAN state. In this way, it will hear only page messages from nodes that will page it in the future. This action eliminates the loser from the leader election and prepares it for the next phases of the protocol. Upon receiving the FHS packets, the winner increases its VOTES by the loser VOTES and continues participating in the leader election by resuming execution of the symmetric protocol.

If N nodes are participating in the leader election, there will be $N-1$ confrontations. The winner of the $N-1^{st}$ confrontation becomes the coordinator. At this final state, the rest of the nodes are in the PAGE SCAN state, waiting to be paged by a node that has information about them.

2.4.2 Phase II: Role Determination

After the election in Phase I, the coordinator has acquired the identities and clocks of all nodes participating in scatternet formation. The coordinator initiates Phase II by checking if the number of discovered nodes N is less than $D+1$. If this is the case, it pages and connects to all other nodes that are waiting in PAGE SCAN; a single piconet is formed with the coordinator as master and the rest of the nodes as slaves. In this special case, the protocol terminates at this point. If $N \geq D+1$, several piconets must be formed and interconnected via bridge nodes. Using eq. (2.20), the coordinator computes the minimum number of piconets P_{min} that satisfy the default criteria R1-R5. Then, the coordinator selects itself and $P_{min}-1$ nodes as the designated masters and $\frac{P_{min}(P_{min}-1)}{2}$ other nodes to be S/S bridges. The remaining $N-(P_{min}+\frac{P_{min}(P_{min}-1)}{2})$ nodes are assigned as ”pure” slaves; they are equally distributed among the coordinator and the rest of the masters.

After role assignment, the coordinator constructs for every master X (and itself) a

connectivity list set (SLAVESLIST(X), BRIDGELIST(X)). Each list contains contains FHS packets (id+clock) to aid the designated master to page its assigned slaves instantaneously. Next, the coordinator pages and connects to the nodes it selected as masters. (Recall that, at the end of Phase I, the rest of the nodes wait in the PAGE SCAN state). A temporary piconet is formed with the coordinator as master and the designated masters as slaves[4]. The coordinator transmits to each designated master its connectivity list set and instructs the designated masters to start Phase III; then it disconnects the temporary piconet and starts Phase III as a master.

2.4.3 Phase III: Connection Establishment

Phase III is initiated by the designated masters (including the coordinator). Each master pages and connects to the slaves and bridges provided in its SLAVESLIST and BRIDGELIST, respectively. As soon as a node is notified by its master that it is a bridge, it waits to be paged by its second master (requirement $R3$). When this happens, the bridge node sends a CONNECTED notification to its masters. When a master receives a CONNECTED notification from *all* its assigned bridges, a fully connected scatternet of P_{min} piconets is guaranteed to be formed and the protocol terminates.

2.4.4 Leader election termination

The most time-consuming part of the protocol is the leader election phase. Phases II and III involve only paging and connecting, which occur instantaneously due to the previous inquiry procedures.

Ideally, election should stop as soon as the coordinator is elected. However, since a node is not aware of the total number of participants, it will never know whether or not

[4]According to eq. (2.20), P_{min} is always less than D and the temporary piconet can always be formed.

it is the winner of the election. Each node maintains a ”state alteration” timeout variable called ALT_TIMEOUT. ALT_TIMEOUT is set upon power-on and reset each time the node wins a confrontation and restarts the symmetric link establishment protocol. When ALT_TIMEOUT expires, the node assumes it is the elected coordinator.

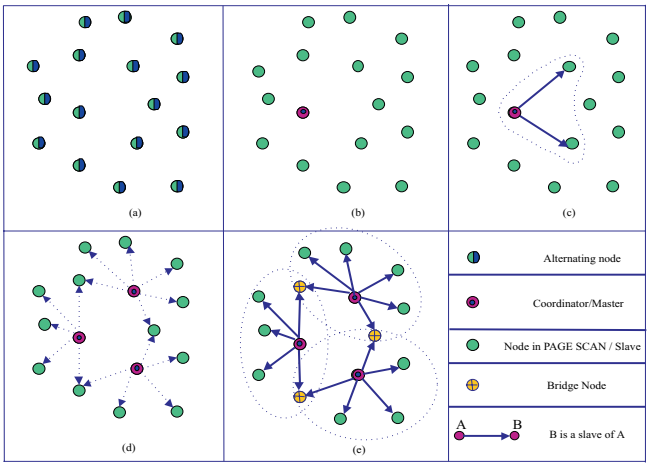


Figure 2.6: BTCP operation: (a) Start of Phase I: All nodes begin alternating, trying to discover other nodes in wireless proximity. (b) End of Phase I: Coordinator has been elected. Given N=16, coordinator computes $P_{min} = 3$ using eq. (2.20). Next, the masters, bridges, and slaves are selected accordingly. (c) Phase II: Coordinator forms a temporary piconet with the designated masters and sends them their connectivity lists. (d) Phase III: Each master pages the nodes specified within its connectivity list. (e) The scatternet is formed.

It is important to determine an appropriate value for ALT_TIMEOUT. A very large value will result in a node having won the competition and continuing alternating without knowing it is the only one left. This implies a slow Phase I and, consequently, slow scatternet formation. On the other hand, using a very short ALT_TIMEOUT, several nodes may assume the role of coordinator; this will result in a disconnected scatternet. We address this issue using the following observation: the link formation delay between

any two out of N alternating nodes is statistically less than the delay of only two alternating nodes. Thus, the delay analysis of the two-node symmetric link establishment protocol can be used to provide a tight estimate for ALT_TIMEOUT.

2.5 Experiments

2.5.1 Emulating Bluetooth

We have implemented BTCP on top of an existing prototype implementation that emulates the Bluetooth environment on a Linux platform. The emulator is used instead of actual Bluetooth devices because it allows testing the protocol for a wide range of parameters and for a large number of nodes.

Each Bluetooth host is implemented as a Linux process consisting of two interacting modules. The Bluetooth Baseband (BB) module emulates in software the Inquiry, Paging and piconet switching procedures, as defined in the Bluetooth Baseband specification [75]. The BTCP module interacts with the BB module through Bluetooth Host Controller Interface (HCI) functions [76]. The use of HCI functions allow us to later replace the BB module with an actual Bluetooth unit.

The wireless medium is simulated by a N_f-hop channel process. The channel process is responsible for the exchange of IAC and FHS packets during the inquiry and paging procedures. It also simulates the occasional frequency collisions and FS delays. Note that the channel process is not similar to a CSMA broadcast channel–the senders and receivers cannot perform any carrier sensing nor any form of intelligent back-off.

We also assume that all devices are within range of each other. This is a valid assumption for networking many short-range wireless devices in a single room. This is mapped in the architecture by having all Bluetooth host processes connected to the

N_f-hop channel process and executing the scatternet formation protocol.

2.5.2 Determining ALT_TIMEOUT

Using the the Periodic_Inquiry_Mode HCI command [76], it is possible to program Bluetooth units to alternate between INQUIRY and INQUIRY SCAN with uniformly distributed state residence intervals. Figure 2.7 plots the mean $E[T_c]$ and standard deviation $\sqrt{V[T_c]}$ of the two-node link establishment delay as a function of the mean state residence interval. Given $E[T_c]$ and $V[T_c]$, ALT_TIMEOUT is determined by the

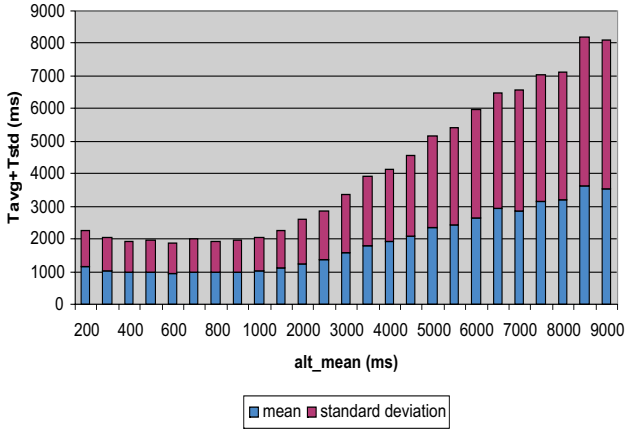


Figure 2.7: The node alternate, with state residence intervals drawn from a uniform distribution of mean μ $msec$. The mean $E[T_c]$ and standard deviation $\sqrt{V[T_c]}$ of the delay of the symmetric protocol, are plotted as a function of μ

.

following empirical formula:

$$ALT\_TIMEOUT = E[T_c] + \sqrt{V[T_c]} + r_{max} \tag{2.21}$$

According to Figure 2.7, for every mean state residence interval, the standard deviation is comparable to the mean. This indicates that the distribution of T_c is not centered

around the mean and justifies the inclusion of the term $\sqrt{V(T_c)}$ in eq. (2.21). The term r_{max} was determined by experimentation. During many protocol runs, the following frequent phenomenon was observed: after the $N - 2^{nd}$ confrontation, the winner A would start alternating by resetting ALT_TIMEOUT while another node B was in SLEEP mode due to a previous back-off. A and B were the last nodes in the election process and would start trying to form the $N - 1^{st}$ connection only after B woke up. The term r_{max} is the upper bound on the back-off interval of the asymmetric protocol and was included in eq. (2.21) to take this case into account.

In the experiments we use a mean state residence interval of $600ms$ which, according to Fig. 2.7 and eq. (2.21), yields a minimum ALT_TIMEOUT of 2527.223ms.

2.5.3 Protocol Performance

We use the average scatternet formation delay and the probability of connection as the protocol performance metrics. The scatternet formation delay is dominated by the delay to elect the coordinator (Phase I). Phases II and III are very fast since they involve only paging and connection establishment. Without loss of accuracy we will represent the overall scatternet formation delay by the leader election delay.

We also distinguish between the "ideal" and "actual" leader election delays, termed as T_{ideal} and T_{actual}, respectively. T_{ideal} is the delay from the time when the first node is powered-on until the coordinator is elected. It is ideal in the sense that the protocol would terminate at this point had the nodes known the number of participants; however, a node will assume it is the coordinator after an additional delay of ALT_TIMEOUT. Therefore, the actual scatternet formation delay T_{actual} is given by:

$$T_{actual} = T_{ideal} + ALT\_TIMEOUT \tag{2.22}$$

The probability of connection is the fraction of experiments where only a single node

assumes the role of coordinator. This metric depends on the value of ALT_TIMEOUT. The higher ALT_TIMEOUT is, the higher the probability of connection, but the longer the scatternet formation delay.

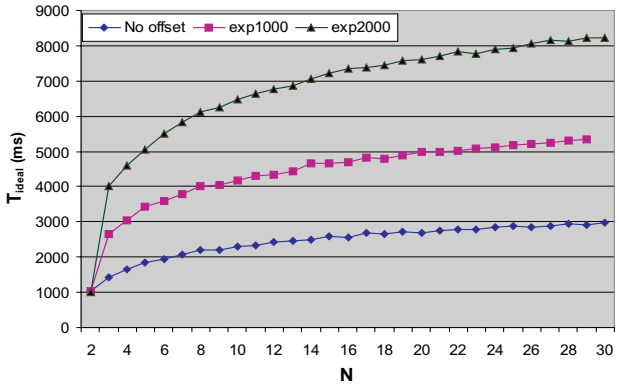


Figure 2.8: Average ideal scatternet formation delay for various application scenarios. Units alternate according to uniformly distributed state residence intervals of $600\ ms$ on the average. Each data point is the average of 10,000 runs.

The protocol delay performance is summarized in Figure 2.8. The ”no offset” curve corresponds to T_{ideal} when all nodes start alternating simultaneously. Delay increases with the number of nodes in a sub-linear manner. This is due to the multiple one-on-one confrontations that occur in parallel during the leader election process. This behavior is a desirable property of a scatternet formation protocol. We would not like, for example, the delay increasing linearly with N. The delay ranges from $1\ s$ to $3\ s$ for $N = 2$ to $N = 30$ nodes.

The ”no offset” curve yields very small delays partly because all nodes start participating in the network formation at the same time instant. In a real world scenario, users will power on their devices in an asynchronous manner. We model the power-ons as a Poisson arrival process within a $W = 10\ s$ application window: after the first user, each

user i arrives after an exponentially distributed delay L_i of mean μ_p and truncated within the $W = 10\ s$ application window. The truncated exponential distribution is preferred to others (e.g. uniform) because it spreads the arrivals over the entire application window. The process is shown in Figure 2.9.

The curves "exp1000" and "exp2000" in Figure 2.8 illustrate T_{ideal} when each user is expected to arrive after the first user within $\mu_p = 1\ s$ and $\mu_p = 2\ s$ on the average, respectively. As μ_p increases, the system becomes more asynchronous and less one-on-one confrontations occur in parallel. This yields an increase in the scatternet formation delay. Nevertheless, the protocol's immunity to the increase of N is preserved. This is illustrated by a constant delay offset between the curves for a fixed N.

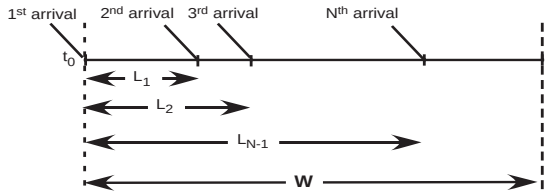


Figure 2.9: The device power-on arrival process. The first user arrives at t_0. Each user i arrives after an interval L_i, drawn from a truncated exponential distribution of mean μ_p and upper bound W.

.

The timeout can be viewed as a delay overhead due to the need for a distributed algorithm. A large ALT_TIMEOUT will yield a connected scatternet with higher probability, but will accumulate a larger actual connection delay T_{actual}. Figure 2.10 illustrates this trade-off by depicting the probability of connection ("timeout efficiency") for several candidate values of ALT_TIMEOUT. For all application scenarios, the timeout efficiency initially increases rapidly with ALT_TIMEOUT and then reaches a steady state. It is clear that the value of ALT_TIMEOUT where the curves start stabilizing is

at $2500\ ms$–very close to the value $2527.223\ ms$ chosen by our empirical formula (eq. (2.21)).

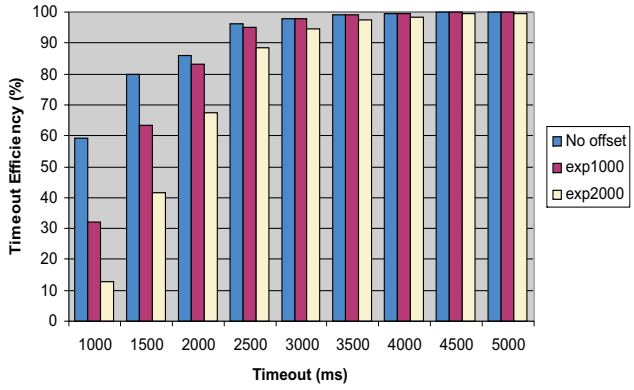


Figure 2.10: Timeout efficiency: Each bar graph is the probability of connection, averaged over N=5,10,20 and 30 nodes (10000 runs for each N).

When an upper bound estimate exists on the number of nodes participating in the protocol, the combination of Figures 2.8 and 2.10 provides practical guidelines. For example, if the expected number of nodes is 30 and an ALT_TIMEOUT of $2500ms$ is used, the average delay experienced by each user will be $3000ms + 2500ms = 5.5s$ (Fig. 2.8) and a connected scatternet will be formed with a probability of 96.13% in the ”no offset” application scenario (Fig. 2.10).

2.6 Related Work

The scatternet formation problem can be summarized as follows: ”Given the network visibility graph induced by the nodes’ wireless proximity, establish a subset of master/slave links such that the resulting communication graph is connected and satisfies the Bluetooth degree constraints”.

Using a Minimum Spanning Tree (MST) framework, Guerin et. al. [31] show that the scatternet formation problem is NP-complete for general visibility graphs[5]. When nodes are distributed on a 2-dimensional plane (Euclidean visibility graphs), the problem can be solved by a MST construction algorithm of polynomial complexity[6]. This is because every node belonging to a Euclidean MST has at most 6 adjacent links–less than the Bluetooth constraint of 7.

Most proposed solutions to the scatternet formation problem are distributed. The protocols can be classified according to the initial information available to the nodes and the structure of the generated topologies.

In [25][31][27] [28][30] the nodes start with a priori knowledge of their one-hop neighbors. Záruba et. al. [25] present a protocol for Euclidean visibility graphs where a designated root node initiates scatternet formation and forms a tree topology. A geometric argument[7] is used to re-assign roles on links in case some nodes exceed the degree constraints during the formation process. It is not analytically proven whether the re-organizations converge to a connected topology. As mentioned in [31], for Euclidean visibility graphs, existing distributed dynamic MST algorithms such as [77] can generate a connected tree topology at the expense of high communication complexity. Sacrificing analytical connectivity guarantees, [31] proposes a heuristic of low communication complexity. The approach requires additional GPS hardware on the Bluetooth nodes to know the coordinates of nodes in proximity.

Li and Stojmenovic [30] generate connected non-tree scatternet topologies for the

[5]NP-completeness holds if a node is forced to act as master or slave to all its adjacent links. If M/S bridges are allowed it is not known whether or not the problem is NP-complete

[6]The MST is constructed by considering the node distances in the visibility graph as the edge weights

[7]In a Euclidean graph, if a node has more than 5 neighbors, then at least 2 of them are within wireless proximity of each other.

Euclidean case. The protocol applies Yao structure, which also requires knowledge of the neighbor coordinates. Petrioli and Basagni [28] trade off the cost of extra GPS hardware by extending the required initial knowledge to two hops. They combine clustering techniques with the geometric argument of [25] to yield connected non-tree scatternet topologies. The BlueNet protocol [27] operates for general visibility graphs but does not guarantee scatternet connectivity.

The problem does not become easier when the nodes start with no knowledge about their surroundings. Due to the random discovery delays it is difficult to make any deterministic claims regarding connectivity, even for the Euclidean case. It is not straightforward to extend the multi-hop protocols in [25][31][27] [28][30] to the zero-knowledge setting because they assume static topologies and do not operate in an incremental manner.

On the other hand, BTCP and the protocols in [26][29], are targeted for the zero-knowledge setting but are currently restricted to the single-hop environment (the visibility graph is complete). Law et. al. [26] construct a connected bipartite scatternet topology with high probability. The protocol operates in synchronous rounds of fixed length where nodes assume sender and receiver roles with a certain probability. The round length is assumed sufficiently large to guarantee connection of two nodes that start in opposite states. However, synchronous operation is difficult to support in a zero-knowledge setting. Tan et. al. [29] propose an asynchronous incremental protocol that creates tree fragments, continuously merged to yield a single tree topology. To avoid loops, only the root nodes in each fragment are allowed to connect. This feature makes it unclear how both the degree constraint and overall scatternet connectivity can be satisfied. BTCP is both distributed and asynchronous; it also provides more flexibility in forming the final WPAN topology due to its centralized role assignment phase.

2.7 Further issues

In ad hoc networks using frequency hopping technology, nodes can be grouped in multiple communication channels. This physical layer setting provides a new way of viewing higher layer functions like topology construction algorithms. Motivated by this environment and using the Bluetooth technology as our research vehicle, we first investigate the Bluetooth standard asymmetric ”sender-receiver” point to point link establishment scheme and then propose a symmetric mechanism for establishing a connection without any role pre-assignment. Based on the ad hoc link formation mechanism we present BTCP, a distributed topology construction protocol where nodes start asynchronously without any prior neighborhood information and result in a network satisfying the connectivity constraints imposed by the Bluetooth technology. The protocol is centered on a leader election process where a coordinator is elected in a distributed fashion and consequently assigns roles to the rest of the nodes in the system.

BTCP was tested under a conference scenario where users arrive in a room and try to form a scatternet by turning on their Bluetooth-enabled devices. An attractive feature of the protocol is that the network formation delay is sub-linear with the number of participating nodes (implying that the users do not need to wait proportionately longer when more users are present). Although the delay is small, each node must have an estimate of how long it must participate in the protocol before assuming protocol termination. A conservative estimate of the timeout will introduce unnecessary delays in network formation while an aggressive estimate may leave the network disconnected. Our analysis of the delay statistics of the symmetric link formation protocol provides a tight estimate of the appropriate timeout value, making the protocol fast while ensuring high probability of scatternet connectedness.

The protocol needs to be extended for the multi-hop case. The leader election mech-

anism can serve as a building block for discovering, connecting partial topology views and then merging them in larger components. A possible implementation of this idea is as follows: During the election process a node maintains a topology map in addition to the FHS packets of the nodes it has won so far. After a one-on-one confrontation, the loser communicates its FHS packets and topology map to the winner. Before starting alternating, the winner pages the nodes indicated in the loser topology map. (Temporary) connections will be established only with the paged nodes that are within proximity of the winner. This results in the winner node updating its local topology map; this process continues until the node loses a one-on-one confrontation or becomes the coordinator. The coordinator uses a centralized algorithm to produce an optimized scatternet based on the discovered topology graph. Using this modified leader election mechanism, it is likely that multiple leaders will be elected and form scatternet clusters with no nodes in common. The clusters are further discovered and merged using a new leader election process operating at the cluster level.

Given a set of nodes with zero knowledge of each other that need to form quickly an initial connnected ad hoc network, BTCP focuses on minimizing the connection delay while providing connectedness with high probability. This is a desired property in application scenarios where ad hoc networks continuously connect (birth), perform a coordinated function for a short amount of time (live) and disconnect (die); connection setup delays should be a small fraction of these ”birth-live-die” cycles. Keeping this network operation model in mind, alternative methods for topology construction need to be studied and compared in terms of delay with the one presented here.

In addition to zero-knowledge network initialization, the reformation of an existing network in the face of dynamic changes can be viewed as a separate but equally important issue. After network connection, a separate topology maintenance and optimization

protocol need to be run to accommodate mobility and/or nodes entering and leaving the network while ensuring that the scatternet is reformed accordingly. Such a protocol should be the subject of future research efforts.

Chapter 3

Asynchronous TDMA: Scheduling and Performance

TDMA is a well known access method for provision of bandwidth guarantees in wireless ad hoc networks. According to TDMA, the system operates using a schedule of period equal to T_{system} slots; at every slot, entities (nodes or links) are scheduled such that there are no conflicts at the intended receivers. The number of conflict-free slots each entity receives determines its allocated bandwidth.

A central performance issue that arises in a TDMA-based ad hoc network is determination of the set of feasible allocations. A demand allocation is feasible if the slot demand of every entity can be satisfied by a TDMA schedule of length less than T_{system} slots. Feasibility characterization is intrinsically coupled with an optimization problem: being able to compute the minimum-length schedule for every demand allocation is equivalent to being able to detect all feasible allocations.

Most studies of the above optimization problem, along with most proposed centralized or distributed TDMA-based protocols, assume the slot boundaries are provided by a global system clock. However, a system-wide synchronization mechanism is not always possible to implement in the distributed ad hoc network setting. In this chapter we

introduce an asynchronous TDMA communication model where time slot reference is provided on a local basis. Since data (as opposed to control) traffic is usually mapped to the point-to-point (as opposed to broadcast) service, we define a link-oriented communication model where time slot reference is provided locally for each link by the hardware clock of one of the node endpoints. Bluetooth scatternets are ad hoc networks that operate according to this model. Asynchronous TDMA removes the need for a global slot synchronization mechanism; however, certain slots are inevitably wasted when nodes switch time slot references on their adjacent links. This phenomenon has been reported in the scatternet scheduling literature [73][78][79] [80][81] as a source of overhead. However, no formal study has examined its effect on the system's ability to allocate bandwidth. This ability is linked to the determination of the region of feasible allocations or, equivalently, to the solution of the related link schedule optimization problem.

Due to the slots wasted for time reference alignment, the minimum period required for realizing a given allocation will be greater than the minimum period required by a perfectly synchronized system. This increase can be seen as overhead due to system asynchronicity. Based on this observation, we can use a two-step procedure to address the optimal link scheduling problem for asynchronous TDMA ad hoc networks. The first step involves finding a minimum-period synchronized schedule for the demand allocation at hand. The second step, our contribution, utilizes the reference synchronized schedule to find an asynchronous schedule of minimum overhead.

The amount of overhead depends on the order by which links are activated in the reference synchronized schedule. We first introduce an algorithm that derives a minimum-overhead asynchronous schedule for a specific ordering. The generated overhead is always upper-bounded regardless of ordering or network configuration. Using this al-

gorithm, it is possible to determine the optimal solution by searching over all possible orderings. This leads to a combinatorial problem where exhaustive search is not feasible for large problem sizes. To this end, we introduce a heuristic algorithm of reduced complexity. The heuristic performs excellent for problem sizes where an optimal solution can be computed. When this is not possible, we investigate the effect of various system parameters on the generated overhead and use the upper bound as the performance measure.

The remainder of this chapter is organized as follows: Section 3.1 introduces a conflict-free scheduling framework for asynchronous TDMA ad hoc networks. In Sections 3.2, 3.3 and 3.4 the problem is introduced and formulated and the overhead minimization algorithms are presented. Section 3.5 evaluates the algorithms performances in various scenarios. Section 3.6 concludes.

3.1 Asynchronous TDMA communication model

Every wireless node has a hardware clock that determines the timing of the radio transceiver. The clocks of different nodes are not synchronized and no mechanism exists for synchronizing them under a global time slot reference.

The ad hoc network is represented as a directed graph $G(N, E)$. A directed edge from node i to node j signifies that i and j are within range and communicate on a link where i has been assigned the role of master and j the role of slave.

The system is slotted and carries point-to-point traffic–each transmission slot carries a packet destined to a single outgoing link. The time slot reference of each link is provided locally by the hardware clock of the master node endpoint. Each slot supports full-duplex communication initiated by the master: During the first part of the slot the

master polls a slave; during the second part a slave responds if polled by the master.

Each node has a single radio transceiver and can communicate (either transmit or receive) to at most one link at a time. Thus, nodes need to coordinate their presence on links in mutual time intervals. Based on its own hardware clock, each node i divides time in fixed-size slots–each equal to the duration of a full-duplex communication slot. Transmissions on adjacent links are coordinated using a local link schedule $\boldsymbol{S}_i$ of period T_{system} slots. The local schedule determines communication action for the duration of a slot: the node can either be active on a single link (start acting as master or slave) or remain idle.

Local schedules of different nodes are not necessarily time-aligned. Every node i maintains a *relative phase* $\phi_{i\to j}$ with respect to each adjacent link (i, j). If $\phi_{i\to j} = -1$, slot p in the local schedule $\boldsymbol{S}_i$ overlaps in time with slots ($p-1$, p) in the local schedule $\boldsymbol{S}_j$. If $\phi_{i\to j} = 1$, then slot p in $\boldsymbol{S}_i$ overlaps with slots (p, $p+1$) in $\boldsymbol{S}_j$. A relative phase $\phi_{i\to j} = 0$ indicates that the hardware clocks of the endpoints happen to be perfectly synchronized. The relative phase maintained at the other link endpoint j is $\phi_{j\to i} = -\phi_{i\to j}$. Given the relative phases and master-slave role assignment on link l, the *link phase* ϕ_l is defined as the relative phase of the master node endpoint.

According to primary interference constraints, communication is successful on a link l only if both node endpoints assign time-overlapping slots in their local schedules. The assignment must be such that when the master starts polling in slot p of its local schedule, the slave must have assigned slots $p + \frac{\phi_l(1+\phi_l)}{2} - 1$ and $p + \frac{\phi_l(1+\phi_l)}{2}$ in its own local schedule for listening to this master. For conflict-free communication on τ_l consecutive slots on link l, the master must allocate τ_l slots in its local schedule for polling while the slave must allocate at least $\tau_l + 1$ time-overlapping slots for aligning to the time reference of this master. In general, an extra slot is needed every time a node

switches to a new time reference as slave.

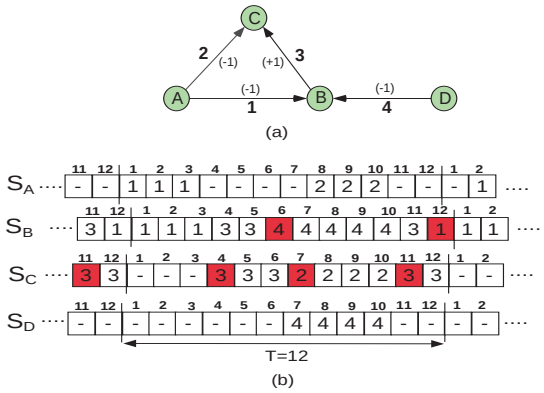


Figure 3.1: (a) Network configuration: Directed edges denote master-slave relationships. Nodes A and D act as masters on all their adjacent links, B is slave on links 1,4 and master on link 3 and C acts as slave on all its links. The numbers in parentheses denote link phases. As an example, since link 1 has a link phase of (-1), slot p in the local schedule S_A of master A must overlap with slots $(p-1,\ p)$ in the local schedule S_C of slave C. (b) The asynchronous TDMA schedule refers to a system that tolerates secondary interference: only links 2 and 4 can transmit simultaneously. Slots where nodes switch time reference as slaves are marked in red. The realized slot allocation is $\boldsymbol{\tau} = (\tau_1, \tau_2, \tau_3, \tau_4) = (3, 3, 3, 4)$.

The communication model captures both single channel systems, where both primary and secondary interference exist as well as multi-channel systems where only primary interference exists. The interference constraints define which links can be activated conflict-free in each case. For both types of systems, a *link slot allocation* $\boldsymbol{\tau} = [\tau_l]$ realized by the network asynchronous TDMA schedule is the number of slots every link l transmits conflict-free during T_{system} slots which equals the number of slots allocated to l in the local schedule of the master endpoint. A *network configuration* consists of the ensemble of a network topology, link phases and master-slave link role assignments.

Figure 3.1 illustrates an example of a network configuration, asynchronous TDMA link schedule and the link slot allocation realized by this schedule.

3.2 Problem formulation and approach

Given a network configuration, it would be of interest to determine feasibility of any given link demand allocation. A demand slot allocation $\boldsymbol{\tau}$ is feasible if it can be realized by a schedule of length less than T_{system} slots. Being able to find the minimum length for any demand allocation is equivalent to detecting all feasible demand allocations.

For synchronized TDMA ad hoc networks the optimal scheduling problem can be described by a generic formulation. Let the ad hoc network be shared by a set E of entities being either nodes or links. An activation set $\boldsymbol{T}_i$ is a set of entities that can transmit conflict-free given the interference constraints in the network. Define $\boldsymbol{T} = \{\boldsymbol{T}_k : 1 \leq k \leq |T|\}$ to be the set consisting of all transmission sets in the ad hoc network. Given a set of demands (time durations) on the entities $\boldsymbol{\tau} = (\tau_1, ..., \tau_{|E|})$, we seek a minimum-length TDMA schedule that can realize these demands. The TDMA schedule can be represented as a sequence of activation sets and their transmission durations. Hence, the optimal TDMA scheduling problem can be solved if we can find the activation duration $\lambda_i \geq 0$ of each transmission set $\boldsymbol{T}_i$ such that $\boldsymbol{\tau}$ is realized in minimum time. More formally:

$$minimize \sum_{i=1}^{|T|} \lambda_i \tag{3.1}$$

subject to:

$$\sum_{i=1}^{|T|} \lambda_i \boldsymbol{I}_i = \boldsymbol{\tau} \tag{3.2}$$

where $\boldsymbol{I}_i$ is the indicator vector of transmission set $\boldsymbol{T}_i$.

The above formulation applies to both node and link scheduling. The interference constraints (single-channel or multi-channel system) are captured by the indicator vectors $\boldsymbol{I}_i$. The problem has been addressed in both continuous time and slotted time. In continuous time, the demands $\boldsymbol{\tau}$ and the solution weights λ_i are real numbers (rates) while in slotted time, they are both integer multiples of a constant time interval (slots).

Almost all instances of this problem are NP-complete. The difficulty in solving it partially stems from the fact that the number of activation sets increases exponentially with the network size. In continuous time the problem for single-channel systems is NP-complete for both node scheduling [82] and link scheduling [52]; in multi-channel systems, link scheduling can be solved in polynomial time [83].

Real-life synchronized TDMA ad hoc networks use the slotted time model. The network operates according to a TDMA schedule of period equal to T_{system} slots. Each slot can carry a certain amount of bits; demands for each entity given in bits/sec are translated in a number of slots. In slotted time, node scheduling has been addressed in [84][85] for single channel systems; link scheduling has been considered in [52] for single channel systems and in [53] for multi-channel systems. Unfortunately, all problem instances in slotted time are NP-complete.

References [84][55][54][56] propose efficient heuristics for the TDMA optimization problem in slotted time. In [54], Silvester proposes such a heuristic for link scheduling in single channel systems. Post, Sarachik and Kerschenbaum address link scheduling for both single-channel and multi-channel systems [55]. Broadcast (node) scheduling is considered in [84]. A unified framework is presented in [56]. The optimal scheduling problem is first parametrized with respect to scheduled entities (links or nodes) and interference constraints and then further abstracted to a generic graph coloring problem. This problem is addressed by a greedy heuristic of polynomial complexity. Alterna-

tively, optimal solutions exist for restricted topologies. For single channel systems, tree topologies can be optimally scheduled [86]. For multi-channel systems, scheduling links is equivalent to coloring edges in a multi-graph where the multiple edges between two node endpoints map to the slot requirement of the corresponding link. If the network topology is bipartite the optimal solution can be reached using minimum edge-coloring algorithms for bipartite multi-graphs [87].

Synchronized TDMA can be viewed as a special case of asynchronous TDMA if all link phases in the network are set to zero. Hence, the optimal link scheduling in asynchronous TDMA is NP-complete in its general form. Existing heuristics or optimal solutions for special cases for synchronized systems are not straightforward to apply to asynchronous TDMA. First, the problem cannot be captured by the generic formulation of equations (3.1) and (3.2)–the notion of activation sets implies existence of slot synchronization. Second, graph coloring techniques are not readily applicable. For example, in multi-channel systems there exists no one-to-one mapping of the slot demand per node pair in the network topology graph to multiple edges for this node pair in the corresponding multi-graph: in the asynchronous system, each link slot demand is the number of slots that should be allocated in the local schedule of the master endpoint; however the slave endpoint must allocate additional slots in its local schedule for time reference alignment. Also, as will be evident later in the discussion, the number of additional slots required in the slave local schedules depends on the order links are activated in the local schedules of the masters.

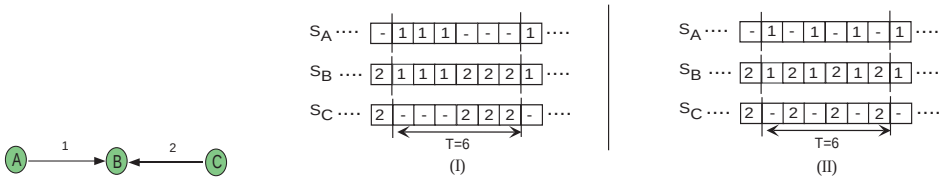
Our approach is based on the observation that the additional slots needed by the slaves yield an increase in period with respect to the minimum period of a perfectly synchronized system. This period increase is an overhead induced by the system asynchronicity. The link schedule optimization problem for the asynchronous system is

translated to an overhead minimization problem. First, a synchronized link schedule that realizes the demand allocation is computed. Using this schedule as a reference we seek an asynchronous schedule of minimum overhead. If a reference synchronized schedule of minimum period can be found, a minimum overhead asynchronous schedule is a minimum-period asynchronous schedule. When the reference schedule period is sub-optimal, a minimum-overhead asynchronous schedule is still useful: the resulting period will be compared to T_{system} for determining feasibility of the demand allocation at hand. Therefore, minimum-overhead schedules allow detection of a greater number of feasible allocations.

The amount of overhead depends on the ordering of link activations in the reference synchronized schedule. Consider the 3-node line configuration of Figure 3.2 where node B is slave to both nodes A and C and where the demand allocation is 3 slots for each link.

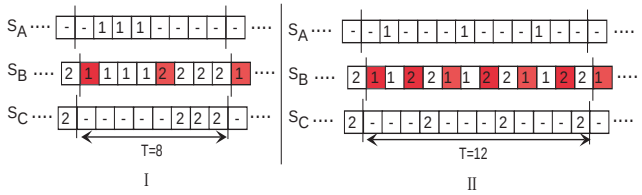
First, let us assume existence of slot synchronization. Since each node can communicate to only a single link at a time, the demand allocation can be realized by a minimum-period schedule of 6 slots. In this schedule, each link is activated 3 times by assigning concurrent slots in the endpoints' local schedules. Figure 5.8(b) illustrates two possible instances of the minimum-period schedule, each using a different ordering of link activations.

Figures 3.2(c)-I and 3.2(c)-II are two asynchronous schedules where links are activated in the order of Figures 5.8(b)-I and 5.8(b)-II, respectively. Both asynchronous schedules need a period greater than 6 slots to realize the demand allocation: in Figure 3.2(c)-I node B switches time reference only once per link yielding a period of 8 slots; in Figure 3.2(c)-II node B is forced to switch time reference every slot, yielding a period of 12 slots.



(a) Network configuration: B is a slave to both A and C.

(b) Synchronized system: Two possible synchronized schedule instances realizing slot allocation $(\tau_1, \tau_2) = (3, 3)$ in a minimum period of 6 slots



(c) Asynchronous system: Depending on the order of link activations, slot allocation $(\tau_1, \tau_2) = (3, 3)$ is realized by schedules of different minimum periods.

Figure 3.2: An example of the asynchronicity overhead

In the example of Figure 3.2, it is possible to determine by inspection the link ordering and asynchronous schedule that yield minimum overhead (Schedule 3.2(c)-I). However, for arbitrary configurations and demand allocations a systematic approach is needed. We first introduce an algorithm that finds a minimum overhead asynchronous schedule for a fixed ordering of link activations in the reference synchronized schedule. This algorithm can be used to determine the minimum-overhead schedule over all possible orderings via exhaustive search. The following sections describe in detail our approach for the solution of this problem.

3.3 Equivalent schedules

A *link activation set* consists of links that can simultaneously transmit without conflicts to the intended receivers. A *synchronized link schedule* $\widetilde{\boldsymbol{S}}$ of period $\widetilde{T}$ is a collection of link activation sets $\{A_k : 1 \leq k \leq \widetilde{T}\}$. A *synchronized schedule instance* $\widetilde{S}^{(\pi)}$ is a periodic sequence of a specific ordering π of the link activation sets of $\widetilde{\boldsymbol{S}}$:

$$\widetilde{S}^{(\pi)} = (A_{\pi(1)}, ..., A_{\pi(\widetilde{T})}). \tag{3.3}$$

where π is a mapping of the indices $\{1, ..., \widetilde{T}\} \rightarrow \{1, ..., \widetilde{T}\}$.

Let $\widetilde{\boldsymbol{S}}^{(\pi)}$ be a synchronized schedule instance realizing allocation $\boldsymbol{\tau}$. For the ordering of link activations in $\widetilde{\boldsymbol{S}}^{(\pi)}$, allocation $\boldsymbol{\tau}$ can be realized by more than one asynchronous schedules, each having a different period.

Consider the synchronized schedule instance of Fig. 5.8(b)-I that realizes allocation $(\tau_1, \tau_2) = (3, 3)$ by activating each link in 3 consecutive slots. For this ordering of link activations, the asynchronous schedule of Fig. 3.2(c)-I realizes the same allocation using a period of 8 slots. If slave B spent 5 slots instead of 4 listening on link 1, the same demand allocation would also be realized with this ordering of link activations but the overall period of the resulting asynchronous schedule would be 9 slots instead of 8.

We define an asynchronous schedule $\boldsymbol{S}^{(\pi)}$ to be *equivalent* to a synchronized schedule instance $\widetilde{\boldsymbol{S}}^{(\pi)}$ if the following conditions hold:

- **(E.1):** Every node activates its adjacent links in $\boldsymbol{S}^{(\pi)}$ in the same order as in $\widetilde{\boldsymbol{S}}^{(\pi)}$.
- **(E.2):** $\boldsymbol{S}^{(\pi)}$ realizes the same allocation as $\boldsymbol{S}^{(\pi)}$.
- **(E.3):** $\boldsymbol{S}^{(\pi)}$ satisfies $(E.1)$ and $(E.2)$ in minimum period.

Thus, an equivalent schedule $\boldsymbol{S}^{(\pi)}$ of a synchronized schedule instance $\widetilde{S}^{(\pi)}$ is an asynchronous schedule that yields minimum overhead for the ordering of link activations in $\widetilde{S}^{(\pi)}$.

We now present an algorithm called EQUIVALENT that takes a network configuration and a reference synchronized schedule instance $\widetilde{\boldsymbol{S}}^{(\pi)}$ as input and outputs the equivalent asynchronous schedule $\boldsymbol{S}^{(\pi)}$ of $\widetilde{\boldsymbol{S}}^{(\pi)}$.

EQUIVALENT constructs $\boldsymbol{S}^{(\pi)}$ incrementally by iterating over the link activation sets of $\widetilde{\boldsymbol{S}}^{(\pi)}$. During iteration k, let l be a link in activation set $A_{\pi(k)}$ and i and j be its master and slave endpoints. Also let $p_i^{(k-1)}$ and $p_j^{(k-1)}$ be the last assigned slot positions in the local schedules $\boldsymbol{S}_i^{(\pi)}$ and $\boldsymbol{S}_j^{(\pi)}$, respectively ($p_n^{(0)} = 0, \forall n \in N$).

First, master i determines the earliest possible slot $p_i^{(k)}$ to be assigned to link l in $\boldsymbol{S}_i^{(\pi)}$. There are three possible cases:

- **Case A: Link l was activated in iteration $k-1$:** The local schedules are "in synch" and node i can allocate to link l the next slot:

$$p_i^{(k)} = p_i^{(k-1)} + 1 \tag{3.4}$$

- **Case B: Link l was not activated in iteration $k-1$ and $p_i^{(k-1)} > p_j^{(k-1)}$:** The master's local schedule is considered forward in time with respect to the slave's local schedule. The earliest slot is again:

$$p_i^{(k)} = p_i^{(k-1)} + 1 \tag{3.5}$$

- **Case C: Link l was not activated in iteration $k-1$ and $p_j^{(k-1)} \geq p_i^{(k-1)}$:** The slave's local schedule is considered forward in time with respect to the master, so the master must find the earliest unassigned slot in $\boldsymbol{S}_i^{(\pi)}$ whose start time exceeds

the end time of slot $p_j^{(k-1)}$ in $\boldsymbol{S}_j^{(\pi)}$:

$$p_i^{(k)} = p_j^{(k-1)} + \frac{\phi_l^2 - \phi_l + 2}{2},\ \phi_l \in \{1, 0, -1\} \tag{3.6}$$

Then i assigns slot $p_i^{(k)}$ to link l. If any intermediate unassigned slots exist between $p_i^{(k-1)}$ and $p_i^{(k)}$, they are assigned as idle in $\boldsymbol{S}_i^{(\pi)}$.

Once the master updates its local schedule, slave j determines $p_j^{(k)}$ as the earliest unassigned slot in $\boldsymbol{S}_j^{(\pi)}$ whose end time exceeds the end time of $p_i^{(k)}$ in $\boldsymbol{S}_i^{(\pi)}$. Depending on the link phase ϕ_l the position of this slot is computed as:

$$p_j^{(k)} = p_i^{(k)} + \frac{\phi_l(1 + \phi_l)}{2},\ \phi_l \in \{1, 0, -1\} \tag{3.7}$$

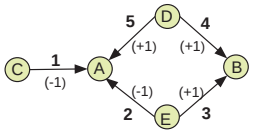
If there are any unassigned slots between $p_j^{(k-1)}$ and $p_j^{(k)}$, they are assigned to link l in $\boldsymbol{S}_j^{(\pi)}$.

The same assignment steps are performed for every link l in $A_{\pi(k)}$. For every node n not considered during iteration k, $p_n^{(k)} = p_n^{(k-1)}$. At the end of iteration k, the *forward progress* $f(k)$ is defined as:

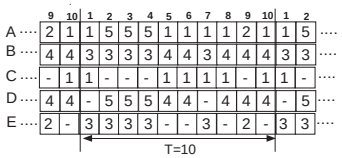
$$f(k) = \max_{n \in N} \{p_n^{(k)}\} \tag{3.8}$$

After $\widetilde{T}$ iterations, the asynchronous schedule period $T^{(\pi)}$ is set to the forward progress $f(\widetilde{T})$. Then, starting again from $A_{\pi(1)}$, a few extra iterations are performed until all nodes assign their local schedules up to slot $T^{(\pi)}$. Upon termination, all nodes use the first $T^{(\pi)}$ slots in their local schedules to form an asynchronous schedule with this period. An example of the algorithm operation is illustrated in Figure 3.3; the algorithm pseudocode can be found in Chapter Appendix 3.

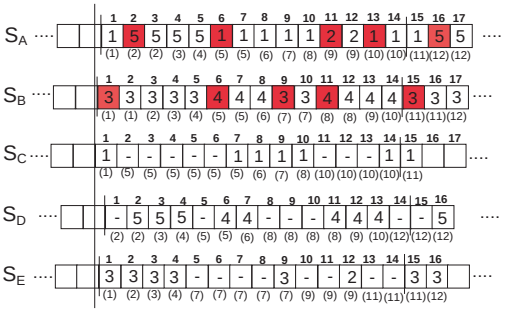
Proposition 3.3.1 *The computational complexity of EQUIVALENT is* $O(N\widetilde{T})$.



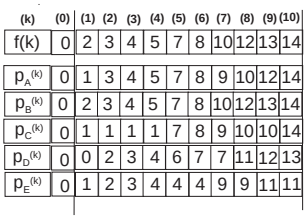
(a) Network configuration



(b) Reference synchronized schedule instance of period $\widetilde{T} = 10$ slots, realizing allocation $(\tau_1, \tau_2, \tau_3, \tau_4, \tau_5) = (6, 1, 5, 5, 3)$.



(c) Numbers in parentheses indicate iteration where the slot was assigned on each node's local schedule. Switching slots are shaded. The equivalent schedule period (=14) is determined at the 10^{th} iteration. Two additional iterations are performed so that all nodes assign their local schedules up to this period.



(d) Evolution of $p_n^{(k)}$ and $f(k)$.

Figure 3.3: An example of the EQUIVALENT algorithm execution

Proof During iteration k of EQUIVALENT the link activation set $A_{\pi(k)}$ is added to the asynchronous schedule. Addition of each link l of $A_{\pi(k)}$ requires a constant number of arithmetic operations:

- Checking whether link $l = (i, j)$ is in $A_{\pi(k-1)}$: This operation can be performed by inspecting if slots $p_i^{(k-1)}$ and $p_j^{(k-1)}$ have been assigned to l in the local schedules $\boldsymbol{S}_i^{(\pi)}$ and $\boldsymbol{S}_j^{(\pi)}$, respectively. (two comparisons).
- Comparing $p_i^{(k-1)}$ with $p_j^{(k-1)}$ (one comparison).
- Updating $p_i^{(k)}$ and $p_j^{(k)}$ (two additions).

Since $A_{\pi(k)}$ is a matching in the network topology graph, it consists of at most $N/2$ links, the size of a perfect matching. Therefore, insertion of a link activation set $A_{\pi(k)}$ requires $O(N)$ operations. EQUIVALENT requires $\widetilde{T}$ iterations to determine the period of the asynchronous schedule $T^{(\pi)}$, as well as a certain number of additional iterations until all nodes fill their local schedules up to $T^{(\pi)}$. Due to the schedule periodicity, there will be no more than $\widetilde{T}$ extra iterations. Therefore, EQUIVALENT requires at most $2\widetilde{T}$ iterations ($O(\widetilde{T})$). Since each iteration requires $O(N)$ operations, the complexity of EQUIVALENT is $O(N\widetilde{T})$. ■

For any network configuration and any link activation ordering π, EQUIVALENT possesses two important properties, summarized by the following theorems:

Theorem 3.3.2 *The asynchronous schedule $\boldsymbol{S}^{(\pi)}$ derived by EQUIVALENT incurs minimum overhead for the link activation ordering corresponding to $\widetilde{\boldsymbol{S}}^{(\pi)}$.*

Proof We need to show that the reference synchronized schedule $\widetilde{\boldsymbol{S}}^{(\pi)}$ and the derived asynchronous schedule $\boldsymbol{S}^{(\pi)}$ satisfy the following conditions:

1. Nodes activate the links in the same order in both schedules.

2. Both schedules realize the same slot allocation.

3. Schedule $\boldsymbol{S}^{(\pi)}$ is conflict-free and has the minimum possible period for the ordering π of link activations.

Condition 1 is satisfied because the link activation set instances are added to $\boldsymbol{S}^{(\pi)}$ in a sequential manner. Also, when a link $l = (i, j)$ is added at iteration k, the master i assigns only one slot to link l. Thus the link masters assign in their local schedules a number of slots equal to the number of slots assigned to l in the synchronized schedule. Since a slot allocation of an asynchronous schedule is defined as the number of conflict-free slots in the local schedules of the master node endpoints, condition 2 also holds.

Regarding condition 3, when a link l is considered on iteration k, equations (3.6) and (3.4) for $p_i^{(k)}$ ensure that the master i assigns the earliest possible slot in its local schedule that does not overlap in time with the last assigned slot $p_j^{(k-1)}$ of slave j. Then, equation (3.7) for $p_j^{(k)}$ ensures that the slave will assign the smallest possible number of time overlapping slots with respect to $p_i^{(k)}$. Similarly, every other endpoint node for a link of iteration k progresses in its local schedule by the minimum number of slots that guarantee a conflict-free transmission. Thus, at every step k, the forward progress $f(k) = \max_{n \in N}\{p_n^{(k)}\}$ is the minimum possible. Since this property holds for all steps k, it also holds for $f(\widetilde{T})$ which is, by definition, the period of the resulting asynchronous schedule. ■

Theorem 3.3.3 *If $\widetilde{T}$ is the period of the reference synchronized schedule, the period $T^{(\pi)}$ of* any *equivalent asynchronous schedule is upper bounded by $2\widetilde{T}$.*

Proof To prove Theorem 3.3.3, we first establish the following lemmae:

Lemma 3.3.4 *For every master-slave link (i,j) let $L_{ij}^{(k)} = max\{p_i^{(k)}, p_j^{(k)}\}$. Then the following inequalities hold:*

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} \geq 0,\ \forall k = 1,2,..,\widetilde{T}. \tag{3.9}$$

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} \leq 2,\ \forall k\ where\ link\ (i,j)\ is\ activated. \tag{3.10}$$

Proof When link (i,j) is activated in iteration k, both nodes i and j assign slots in their local schedule and therefore $L_{ij}^{(k)} > L_{ij}^{(k-1)}$. If nodes i and j are not involved in any link activation during iteration k, then $L_{ij}^{(k)} = L_{ij}^{(k-1)}$ since p_i and p_j are not updated. Therefore in general $L_{ij}^{(k)} \geq L_{ij}^{(k-1)}$.

We now prove the upper bound. Let link (i,j) where master is i and slave is j be activated in iteration k. If this is the case then due to equation (3.7), $p_j^{(k)} \geq p_i^{(k)}$ and therefore $L_{ij}^{(k)} = p_j^{(k)}$. We now distinguish 3 different cases that arise when the link (i,j) is activated in iteration k:

- **Link (i,j) was activated in iteration $k-1$**: Equation (3.7) was used in iteration k-1 and therefore $p_j^{(k-1)} = p_i^{(k-1)} + \frac{\phi_l(1+\phi_l)}{2} \geq p_i^{(k-1)}$. Therefore $L_{ij}^{(k-1)} = p_j^{(k-1)}$. From equations (3.6) and (3.7), $p_j^{(k)} = p_i^{(k-1)} + 1 + \frac{\phi_l(1+\phi_l)}{2}$. Since $L_{ij}^{(k)} = p_j^{(k)}$, we finally have that

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 1 \leq 2. \tag{3.11}$$

- **Link (i,j) was not activated in iteration $k-1$ and $p_i^{(k-1)} > p_j^{(k-1)}$**: In this case $L_{ij}^{(k-1)} = p_i^{(k-1)}$. Also from equations (3.6) and (3.7) we have that $L_{ij}^{(k)} = p_j^{(k)} = p_i^{(k-1)} + 1 + \frac{\phi_l(1+\phi_l)}{2}$. Therefore,

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 1 + \frac{\phi_l(1+\phi_l)}{2} \leq 2. \tag{3.12}$$

- **Link (i,j) was not activated in iteration $k-1$ and $p_j^{(k-1)} \geq p_i^{(k-1)}$**: In this case $L_{ij}^{(k-1)} = p_j^{(k-1)}$. Application of equations (3.6) and (3.7) yields $L_{ij}^{(k)} = p_j^{(k)} =$

$p_j^{(k-1)} + 2$ and then:

$$L_{ij}^{(k)} - L_{ij}^{(k-1)} = 2 \leq 2. \tag{3.13}$$

For all cases $L_{ij}^{(k)} - L_{ij}^{(k-1)} \leq 2$. ■

Lemma 3.3.5 *The following property holds for the forward progress $f(k)$ for every iteration k:*

$$0 \leq f(k) - f(k-1) \leq 2, \; \forall k = 1, 2.., \widetilde{T} \tag{3.14}$$

Proof We use contradiction. Suppose there is an iteration k for which $f(k) - f(k-1) > 2$. Since $f(k)$ is strictly greater than $f(k-1)$ the increase in the forward progress was contributed by at least one link $l = (i, j)$ in the link set $A_{\pi(k)}$ that was activated during this iteration. This means that $L_{ij}^{(k)} = f(k)$. From Lemma 3.3.4 it holds that:

$$\begin{aligned} L_{ij}^{(k-1)} &\geq L_{ij}^{(k)} - 2 \Leftrightarrow \\ L_{ij}^{(k-1)} &\geq f(k) - 2 \end{aligned} \tag{3.15}$$

and from the hypothesis we have that $f(k-1) < f(k) - 2$. Therefore it must be that $L_{ij}^{(k-1)} > f(k-1)$. We arrive at a contradiction since by the definition of these quantities this implies that $max\{p_i^{(k-1)}, p_j^{(k-1)}\} > \max_{n \in N}\{p_n^{(k-1)}\}$. ■

We are now ready to prove Theorem 3.3.3. Starting from Lemma 3.3.5:

$$\begin{aligned} \sum_{k=1}^{\widetilde{T}} (f(k) - f(k-1)) &\leq \sum_{k=1}^{\widetilde{T}} (2) \overset{f(0)=0}{\Longleftrightarrow} \\ f(\widetilde{T}) &\leq 2\widetilde{T} \overset{T^{(\pi)} \triangleq f(\widetilde{T})}{\Longleftrightarrow} \\ T^{(\pi)} &\leq 2\widetilde{T} \end{aligned}$$

■

Theorem 3.3.3 states that the maximum overhead of an equivalent schedule is $\widetilde{T}$ slots. This leads to the following statement for feasibility of allocations in asynchronous TDMA ad hoc networks:

Corollary 3.3.6 *Consider an asynchronous TDMA ad hoc network operating with a period T_{system} and a demand allocation $\boldsymbol{\tau}$. If $\boldsymbol{\tau}$ can be realized by a* synchronized *schedule of period $\widetilde{T} \leq \lfloor T_{system}/2 \rfloor$, then $\boldsymbol{\tau}$ is feasible by the asynchronous system.*

Proof From Theorem 3.3.3, for any permutation π:

$$\begin{aligned} T^{(\pi)}(\boldsymbol{\tau}) &\leq 2\widetilde{T}(\boldsymbol{\tau}) \\ &\leq 2(\lfloor T_{system}/2 \rfloor) \\ &\leq T_{system} \end{aligned}$$

Theorem 3.3.2 states that $T^{(\pi)}(\boldsymbol{\tau})$ is the minimum period that can be generated by link activation ordering π. Since the minimum period is less than or equal to the system period, the allocation $\boldsymbol{\tau}$ is feasible. ■

Corollary 3.3.6 asserts that EQUIVALENT can realize at least half the allocations that are feasible under perfect synchronization. If the condition $\widetilde{T} \leq \lfloor T_{system}/2 \rfloor$ holds for a demand allocation, any reference synchronized schedule instance can be used to generate an asynchronous schedule realizing this allocation. Otherwise, we must solve the optimization problem addressed next.

3.4 Computing optimal asynchronous schedules

3.4.1 Optimal algorithm

The optimal asynchronous schedule can be determined by executing EQUIVALENT for all $\widetilde{T}!$ synchronized schedule instances $\widetilde{\boldsymbol{S}}^{(\pi)}$ and selecting the equivalent schedule of

minimum overhead. Such an exhaustive search is prohibitive even for small values of $\widetilde{T}$.

A link activation set may appear multiple times in the reference synchronized schedule. The search space can be reduced if we consider only reference schedules where all instances of each link activation set are scheduled in consecutive slots–no switching slots are generated by EQUIVALENT when $A_{\pi(k-1)} = A_{\pi(k)}$; the overhead is zero during such a transition. If $M(\widetilde{\boldsymbol{S}})$ is the set of *distinct* link activation sets appearing in the reference schedule, we only need to search $|M(\widetilde{\boldsymbol{S}})|!$ schedule instances instead of $\widetilde{T}!$. Unfortunately, even $|M(\widetilde{\boldsymbol{S}})|!$ can be prohibitively large for exhaustive searches. In this case we resort to the heuristic algorithm introduced in the next section.

3.4.2 MIN_PROGRESS

MIN_PROGRESS is a heuristic for overhead minimization that consists of two phases. Phase I determines an ordering π_h of the distinct link activation sets in $M(\widetilde{\boldsymbol{S}})$. Phase II involves two steps: first, a synchronized schedule instance is formed, where distinct link activation sets are ordered according to π_h and the instances of each set are activated in consecutive slots. Second, this synchronized schedule instance is input to EQUIVALENT to generate the final asynchronous schedule.

We now describe Phase I that selects π_h. An asynchronous schedule is constructed using only the distinct link activation sets instead of all their instances. The sets are added to the asynchronous schedule in the same way as instances are added in EQUIVALENT. Upon initialization, an arbitrary link activation set of the set $M(\widetilde{\boldsymbol{S}})$ is added to the asynchronous schedule. Let $U^{(k-1)}$ be the set of all unassigned link activation sets at the start of iteration k ($U^{(0)} = M(\widetilde{\boldsymbol{S}})$). The addition of each set M^{α} of $U^{(k-1)}$ will generate a forward progress $f(\alpha, k)$ for the asynchronous schedule. The algorithm se-

lects the link activation set yielding minimum forward progress, with ties being broken arbitrarily. Let M^{α_k} be the selected set. Then the k-th entry of π_h is set to α_k and set M^{α_k} is removed from the U-set. The same steps are repeated until the U-set becomes empty after $|M(\widetilde{\boldsymbol{S}})|$ iterations.

Phase I can be extended to select and insert multiple link activation sets per iteration, according to a horizon parameter h. During iteration k, all possible h-set blocks in the U-set and all possible orderings ($h!$) of the link activation sets within each h-set block are considered. The block and ordering that yields minimum forward progress is selected and added to the asynchronous schedule. The selected block is removed from the U-set and the next iteration is performed. Depending on whether h divides $|M(\widetilde{\boldsymbol{S}})|$ or not, the algorithm will terminate in $\lfloor \frac{|M(\widetilde{\boldsymbol{S}})|}{h} \rfloor$ or $\lfloor \frac{|M(\widetilde{\boldsymbol{S}})|}{h} \rfloor + 1$ iterations, respectively. The algorithm pseudocode can be found in Chapter Appendix 3.B.

For the minimum horizon value ($h = 1$), each block consists of a single activation set. During iteration k, the remaining $|M(\widetilde{\boldsymbol{S}})| - k$ activation sets in he U-set are tested. Therefore, only $\sum_{k=1}^{|M(\widetilde{\boldsymbol{S}})|} k = \frac{(|M(\widetilde{\boldsymbol{S}})|)(|M(\widetilde{\boldsymbol{S}})|+1)}{2}$ tests or, equivalently, $O(N|M(\widetilde{\boldsymbol{S}})|^2)$ operations are performed in this case. Increasing the horizon h is expected to improve performance because more orderings are tested per iteration. This, however, comes at an expense of computational complexity. For the maximum horizon value ($h = |M(\widetilde{\boldsymbol{S}})|$) MIN_PROGRESS is essentially the optimal algorithm–it includes a single iteration where a block of $|M(\widetilde{\boldsymbol{S}})|!$ orderings must be exhaustively tested.

The dependence of complexity on h is summarized by the following proposition:

Proposition 3.4.1 *For $h > 0$ and fixed, the computational complexity of MIN_PROGRESS is $O(N|M(\widetilde{\boldsymbol{S}})|^{h+1})$.*

Proof Let M be the number of distinct activation sets in the reference synchronized

schedule ($M = |M(\widetilde{\boldsymbol{S}})|$). The complexity of MIN_PROGRESS is determined by the complexities of Phases I and II:

1) Complexity of Phase I: During iteration k, $\binom{M-(k-1)h}{h}$ blocks are considered and, for each block, $h!$ orderings of activation sets are tested. Depending on whether h divides M or not, the last iteration will consist of a single block of h or $(M \ mod \ h)$ activation sets, respectively. Testing each ordering of activation sets involves insertion of h activation sets to the asynchronous schedule. Therefore, the total number of insertions C_I throughout the execution of Phase I is given by:

$$C_I = \sum_{k=1}^{\lfloor \frac{M}{h} \rfloor} \binom{M-(k-1)h}{h} \cdot h! \cdot h + r(M,h) \cdot h \tag{3.16}$$

where

$$r(M,h) = \begin{cases} (M \ mod \ h)! & \text{if } M \ mod \ h \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.17}$$

After some algebraic manipulations, equation (3.16) yields:

$$C_I = h \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor - 1} \prod_{i=0}^{h-1} ((M-i) - hk) + r(M,h) \cdot h \tag{3.18}$$

An upper bound to C_I is:

$$\begin{aligned} C_I &< h \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor - 1} \prod_{i=0}^{h-1} (M - hk) + r(M,h) \cdot h \\ &= h \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor - 1} (M - hk)^h + r(M,h) \cdot h \\ &= h \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor - 1} \sum_{i=0}^{h} (-1)^i a_i M^{h-i} (hk)^i + r(M,h) \cdot h \\ &= h \sum_{i=0}^{h} (-1)^i a_i h^i M^{h-i} \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor - 1} k^i + r(M,h) \cdot h \\ &= h \sum_{i=0}^{h} (-1)^i a_i h^i M^{h-i} \sum_{k=1}^{\lfloor \frac{M}{h} \rfloor - 1} k^i + r(M,h) \cdot h \end{aligned} \tag{3.19}$$

where a_i are positive integers. The term $r(M,h)\cdot h$ is constant, since $(M \ mod \ h) < h$. The term $\sum_{k=1}^{\lfloor \frac{M}{h} \rfloor -1} k^i$ is $\Theta(M^{i+1})$ because the power sum $\sum_{k=1}^{n} k^i$ can be expanded in polynomial form as $\sum_{j=0}^{i+1} b_j n^j$, where b_j are integers. Multiplying this term with M^{h-i} in (3.19), causes the entire term in (3.19) to be $\Theta(M^{h+1})$.

A lower bound to C_I is:

$$C_I > h \sum_{k=0}^{\lfloor \frac{M}{h} \rfloor -1} \prod_{i=0}^{h-1} ((M-(h-1)) - hk) + r(M,h)\cdot h \tag{3.20}$$

and can be shown to be $\Theta(M^{h+1})$ in a similar way as (3.19). Therefore, C_I is $\Theta(M^{h+1})$.

From the proof of Proposition 1, the insertion of each activation set requires $O(N)$ operations. Thus, the number of operations needed by Phase I is $O(NM^{h+1})$.

2) Complexity of Phase II: Given the ordering π_h computed by Phase I, Phase II uses EQUIVALENT to generate the corresponding minimum-overhead asynchronous schedule. According to Proposition 1, EQUIVALENT requires $O(NM)$ operations for inserting M blocks of activation sets.

From 1) and 2), we conclude that the complexity of MIN_PROGRESS is dominated by the complexity of Phase I and is $O(NM^{h+1})$. ■

Given a specific input reference schedule, the horizon h must be carefully selected for tractability. According to MIN_PROGRESS, the maximum number of $\binom{|M(\widetilde{\boldsymbol{S}})|}{h}$ blocks must be considered in the first iteration. The horizon h must be selected small enough to allow exhaustive enumeration of this number, as well as exhaustive enumeration of $h!$ orderings per block. The algorithm performance with respect to h will be investigated next in the experiments section.

3.5 Performance Evaluation

3.5.1 Factors affecting the overhead

We are interested in evaluating performance in view of the factors that affect the asynchronicity overhead. The overhead is first related to the topology structure. In general, denser topologies are expected to produce higher overhead because more links will translate to a higher number of time reference switches. Performance is also affected by the master-slave role assignments. In the example of Figure 3.2, if node B is assigned as master to nodes A and C, the overhead is zero due to the single time reference in the system.

For a specific network configuration the overhead also depends on the demand allocation at hand. A parameter specific to the demand allocation is the ratio $|M(\widetilde{S})|$ of distinct link activation sets to the period $\widetilde{T}$ of the optimal reference schedule. A small ratio is desirable because overhead is generated only during the transitions between distinct activation sets in the synchronized schedule. Another related parameter is the period $\widetilde{T}$ of the synchronized schedule. Larger periods may allow for smaller $|M(\widetilde{S})|/\widetilde{T}$ ratios and, therefore, less generated overhead.

3.5.2 Experimental setting

Performance must be evaluated for a variety of network configurations and optimal reference synchronized schedules. As mentioned in Section 3.2, determination of optimal synchronized schedules is in general a NP-complete problem. However, for bipartite topologies in multi-channel systems, the minimum period equals the maximum node

utilization:

$$\widetilde{T}(\boldsymbol{\tau}) = \max_{i \in N} \sum_{l \in L(i)} \tau_l. \tag{3.21}$$

where $L(i)$ is the set of adjacent links to node i. Thus, in this case, optimal reference synchronized schedules of period $\widetilde{T}$ can be constructed by generating arbitrary conflict-free schedules where at least one node transmits during the entire period.

In our experiments we consider $|N|$-node multi-channel bipartite networks with $|N|/2$ nodes per bipartite set. This provides a baseline topology of $|N|^2/4$ links. We use the restrictive parameters B_{max} and f to generate various topologies from the baseline. The channel degree parameter B_{max} is an upper bound on the number of channels a node can participate as slave. Such a constraint would arise in practice to avoid excessive overhead. We also restrict the number of links where a node can act as master to 7. This restriction is specific to Bluetooth, a multi-channel asynchronous TDMA system. Combined with B_{max}, this provides an upper bound of $B_{max} + 6$ to the overall link degree of each node in the topologies we consider. The density parameter f ($0 \leq f \leq 1$) generates topologies where an arbitrary $f \times 100\%$ links of the baseline topology remain intact while the rest have been removed.

Given a topology constructed as above, asynchronicity is introduced by 1) master-slave role assignments on the links and 2) arbitrary phase differences in the hardware clocks of the nodes in the network. According to the link role assignments, a node may act either as master to all its adjacent links (master) or as slave to all its adjacent links (S/S bridge), or as master to some links and slave to others (M/S bridge).

3.5.3 Performance of MIN_PROGRESS with respect to optimal

Six 20-node bipartite topologies (10 masters and 10 S/S bridges) of varying density are considered in this experiment. For each topology we randomly generate 100 reference

synchronized schedules of period $\widetilde{T} = 7$. This period allows exhaustive search and determination of the optimal asynchronous schedule. Figure 3.4 compares the resulting optimal and MIN_PROGRESS periods. For each topology, the periods are averaged over all reference schedules. Using a horizon $h = 1$, MIN_PROGRESS exceeds the optimal by less than one slot on the average, while in topology 5 it exceeds the optimal by 1.3 slots on the average.

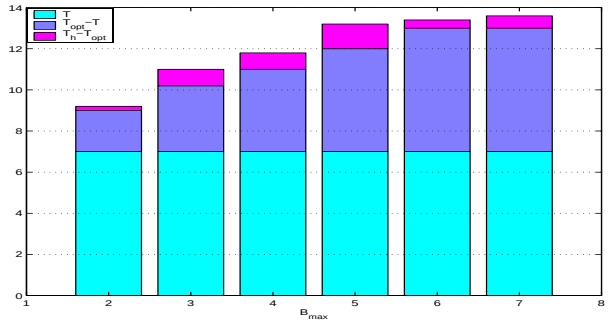


Figure 3.4: MIN_PROGRESS vs. optimal. Each bar graph corresponds to a different 20-node bipartite network configuration where density increases by varying B_{max} from 2 to 7. The reference synchronized schedule period is 7 slots. The optimal period T_{opt} and the MIN_PROGRESS ($h = 1$) period T_h of each bar are averages of 100 reference synchronized schedules.

The optimal and MIN_PROGRESS periods increase with B_{max} and for $B_{max} = 7$ they both approach 14 slots, the upper bound of EQUIVALENT. The high overhead stems from B_{max} being equal to the small reference period $\widetilde{T}$: S/S bridges with such a channel degree need to switch time reference at almost every slot regardless of the link activation order in the reference schedule.

3.5.4 Performance of MIN_PROGRESS for large problem sizes

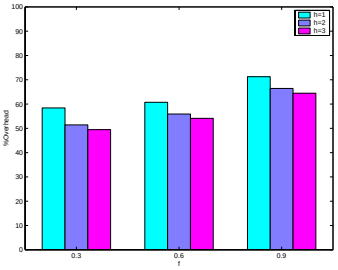
For each parameter set $(N, B_{max}, f, \widetilde{T})$ we generate 10 topologies and, for each topology, 100 arbitrary reference synchronized schedules. For each $(N, B_{max}, f, \widetilde{T})$, the overhead is averaged over the corresponding topologies and reference schedules and is plotted as the %increase in the reference period $\widetilde{T}$. If T_h is the period computed by MIN_PROGRESS, this quantity is equal to $\frac{T_h - \widetilde{T}}{\widetilde{T}}$, with 100% denoting that MIN_PROGRESS yields period $2\widetilde{T}$, the upper bound of EQUIVALENT. We proceed by investigating the various factors that affect the performance of MIN_PROGRESS.

Effect of horizon

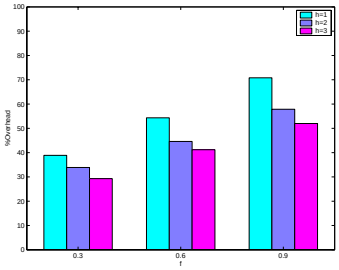
In this set of experiments, we use 20-node bipartite topologies (10 masters and 10 S/S bridges) and vary the density parameter f ($B_{max} = 7$) and reference period $\widetilde{T}$. Figure 3.5 plots the overhead of MIN_PROGRESS using up to 3 activation sets per block (h=1 to h=3).

For all scenarios, the overhead decreases as h increases. The improvement is always more drastic from $h = 1$ to $h = 2$ than from $h = 2$ to $h = 3$. Using $h = 2$ instead of $h = 1$ appears beneficial for larger periods and densities (bar graphs $f = 0.6, 0.9$ in Fig. 3.5(c) and Fig. 3.5(c)), with a maximum overhead reduction of 13% at $\widetilde{T} = 112$ and $f = 0.9$.

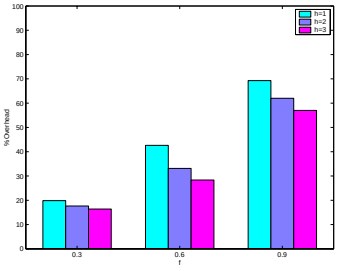
For the lowest density considered, MIN_PROGRESS performs similarly for all h. Overall, a horizon $h = 2$ seems to provide a good performance/complexity trade-off at higher reference periods and topology densities, while a horizon $h = 1$ appears sufficient at low topology densities.



(a) $\widetilde{T} = 28$ slots, $B_{max} = 7$, f varies.



(b) $\widetilde{T} = 112$ slots, $B_{max} = 7$, f varies.



(c) $\widetilde{T} = 448$ slots, $B_{max} = 7$, f varies.

Figure 3.5: Effect of the choice of horizon for varying topology densities and reference periods ($N = 20$, $B_{max} = 7$).

Effect of phase and role assignments

Consider a topology graph G(N,E). Since for every link l, ϕ_l can be -1, 0, or 1, there are $3^{|E|}$ possible link phase assignments in the network. Also, there are $2^{|E|}$ possible master-slave link role assignments.

In this experiment, we consider 20-node bipartite topologies. For a specific topology and reference synchronized schedule, we measure the standard deviation of the generated overhead of MIN_PROGRESS for a sample of 1000 arbitrary phase (or role) assignments. Then, for each parameter set (f,$\widetilde{T}$) we plot the average standard deviation over the corresponding topologies and reference schedules.

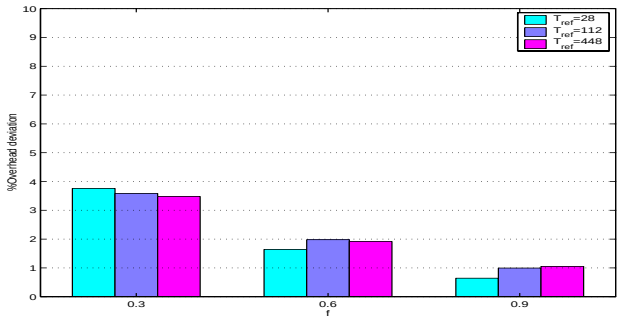


Figure 3.6: Average overhead standard deviation due to link phase variability for 20-node networks and various values of f and $\widetilde{T}$ ($h = 1$, fixed link roles per topology)

.

For every (f,$\widetilde{T}$), role variability (Fig 3.7) produces higher standard deviation than phase variability (Fig 3.6)–the difference never exceeds 1%. Apart from this difference, both figures have similar properties: For a fixed density the standard deviation appears insensitive to $\widetilde{T}$–less than 0.5% changes are observed. However, for every $\widetilde{T}$, the standard deviation decreases as the density increases. This indicates that the overhead deviates less from a certain mean as the number of links per locality increases; therefore variability in phase and role assignments affect the algorithm performance to

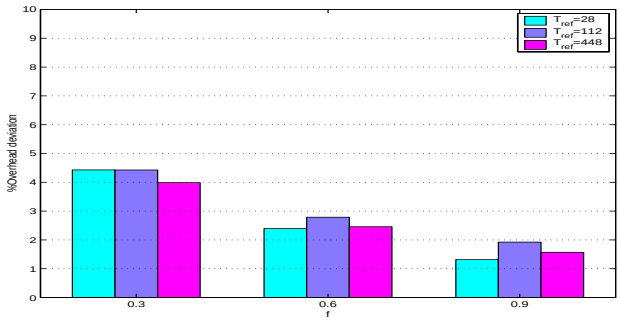


Figure 3.7: Average overhead standard deviation due to link role assignment variability for 20-node networks and various values of f and $\widetilde{T}$ ($h = 1$, fixed link phases per topology).

a lesser extent in this case.

Effect of density

Here, a 100-node (50 masters, 50 S/S bridges) baseline bipartite topology is used. Figure 3.8 illustrates the effect of B_{max} on the overhead of MIN_PROGRESS. For fixed $\widetilde{T}$ the overhead consistently increases with B_{max}. At $\widetilde{T} = 28$, the overhead is 15% when $B_{max} = 2$ but reaches 60% when $B_{max} = 7$. The overhead decreases as the reference period increases. At $B_{max} = 7$ the overhead reduces to 30% for $\widetilde{T} = 896$ slots. While this decrease is more drastic for transitions between smaller periods (e.g. from 28 to 56 slots), it is less for larger periods (e.g. from 448 to 896 slots). This indicates that a non-negligible overhead may still exist even if the system uses a large period. Similar trends arise in Figure 3.9 where $B_{max} = 7$ and only parameter f is used to vary the topology density. The overhead increases with network density regardless of the number of time references in which each node participates.

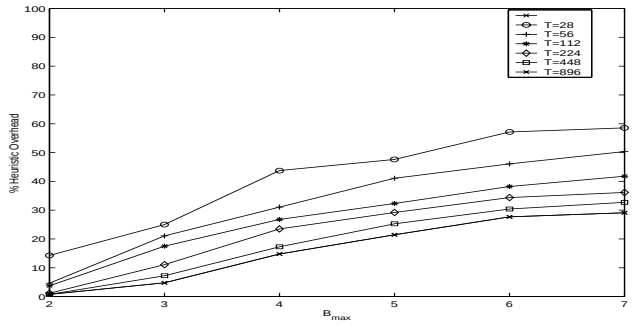


Figure 3.8: Overhead of MIN_PROGRESS ($h = 1$) for 100-node networks as B_{max} and $\widetilde{T}$ vary ($f = 1.0$)

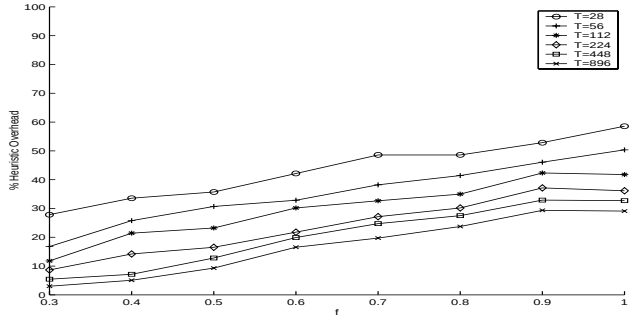


Figure 3.9: Overhead of MIN_PROGRESS($h = 1$) for 100-node networks as f and $\widetilde{T}$ vary ($B_{max} = 7$).

Effect of demand allocation

The previous experiments investigated the algorithm performance averaged over arbitrary demand allocations and topologies. A natural question that arises next is whether there exists a network configuration and demand allocation for which the generated overhead is maximized. In this section we make a first attempt to informally classify such worst-case instances and then test our intuition through simulations.

Let the topology be bipartite and $\Psi^{(\widetilde{T})}$ be the set of all allocations realized by a synchronized schedule of minimum period $\widetilde{T}$. For any allocation $\boldsymbol{\tau}$ in $\Psi^{(\widetilde{T})}$, let $BN(\boldsymbol{\tau})$ be the set of nodes that receive maximum utilization $\widetilde{T}$ under $\boldsymbol{\tau}$.

$$BN(\boldsymbol{\tau}) = \{n : arg\max_{i \in N} \sum_{j \in N(i)} \tau_{ij}\}. \tag{3.22}$$

We conjecture that maximum overhead will be generated if the following conditions hold for a demand allocation $\boldsymbol{\tau}^{max}$ in $\Psi^{(\widetilde{T})}$ and at least one of the bottleneck nodes in $BN(\tau^{max})$:

- **P1:** In addition to maximum utilization, the node has maximum link degree.
- **P2:** The node is a S/S bridge.
- **P3:** Allocation $\boldsymbol{\tau}^{max}$ is such that the node is requested to allocate an equal number of slots to its adjacent links.

A maximum utilization node will be considered at every iteration of an overhead minimization algorithm. Also, since this is a node of maximum degree and acts as a S/S bridge, it will visit the maximum possible number of time references (B_{max}) as slave. If link demands are equal for this node, we can show that the overhead will be maximized under the worst ordering of link activations.

A maxmin fair allocation in a synchronized multi-channel wireless ad hoc network maximizes utilization of the nodes with maximum link degree [88, 89]. If at least one of these nodes is also assigned as a S/S bridge then conditions P1-P3 will hold.

Figure 3.10 compares the MIN_PROGRESS overhead resulting from a maxmin fair reference schedule and the average MIN_PROGRESS overhead over 100 arbitrary schedules. (The algorithm in [88] is used to compute the reference maxmin fair schedules).

The average MIN_PROGRESS overhead decreases as the system period increases. The overhead for the maxmin fair schedule however, does not change significantly– in the order of 80% for all cases. This indicates that the overhead can be very high for the allocations we identified even if an overhead minimization algorithm such as MIN_PROGRESS is used. Counterintuitively, the overhead remains high even if the reference period increases. Nevertheless, it is always less than the upper bound given by EQUIVALENT.

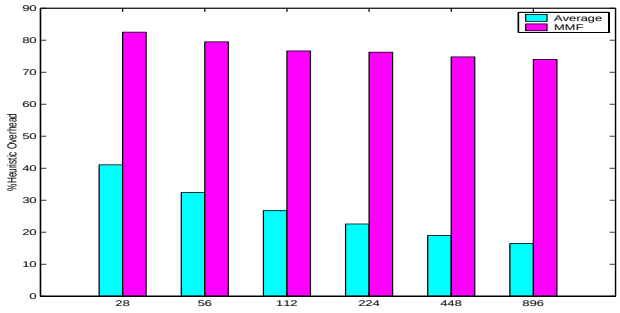


Figure 3.10: MIN_PROGRESS overhead for maxmin fair allocations vs. average MIN_PROGRESS overhead. For each reference period, both quantities are averaged over all topologies considered in Figures 3.8 and 3.9

3.6 Conclusions

In this chapter, we addressed the problem of minimizing overhead in TDMA wireless ad hoc networks that use multiple local time slot references instead of a single global time slot reference. This overhead arises due to slots wasted when nodes synchronize to the different local time slot references and manifests as loss of supported allocations with respect to a perfectly synchronized system. The problem was cast and addressed using a generic framework; the results can be directly applied to Bluetooth, a wireless technology operating according to the asynchronous TDMA communication paradigm.

It was demonstrated that the overhead can significantly affect the ability of a network to allocate bandwidth if no measures are taken to minimize it. We introduced two scheduling algorithms that aim to minimize overhead while ensuring that the generated overhead has an upper bound regardless of the network configuration or demand allocation at hand. The first algorithm reaches the optimal solution but cannot be applied to large problem sizes because it relies on exhaustive search. For such cases an efficient heuristic was devised. We also identified and verified, through simulations, certain conditions on demand allocations and network configurations for which the overhead can be high even if an overhead minimization algorithm is run. Further investigation of the exact nature of such conditions is an interesting research direction.

Both optimal and heuristic algorithms are centralized and can operate in settings where global information is available. More importantly, they can be used to provide design insights and serve as a reference performance measure for any overhead-aware distributed approaches that will follow.

Chapter Appendix 3−Pseudocodes of EQUIVALENT and MIN_PROGRESS algorithms

Procedure EQUIVALENT

input : $G(N,E), \mathbf{\Phi}=[\phi_l],\ \widetilde{\boldsymbol{S}}^{(\boldsymbol{\pi})}=[A_{\pi(k)}],\ \boldsymbol{\pi},\ \widetilde{T}$

output : $\boldsymbol{S}_{eq}=[\boldsymbol{S}_n],\ n\in N$: The asynchronous equivalent schedule of $\widetilde{\boldsymbol{S}}^{(\boldsymbol{\pi})}$

T_{eq}: The period of $\boldsymbol{S}_{eq}$.

local : $\boldsymbol{p}=[p_n^{(k)}],\ \boldsymbol{f}=[f(k)],\ k$

Initialization: $f(0)=0, p_n^0=0,\ \forall n\in N$;

begin

1 **for** $k=1$ *to* $\widetilde{T}$ **do**

AddLinkActivationSet($G,\ \mathbf{\Phi},\ k,\ A_{\pi(k)},\ \boldsymbol{p},\ f(k), \boldsymbol{S}_{eq}$);

end

$T_{eq}=\boldsymbol{f}(\widetilde{T})$;

$q=1$;

$k=\widetilde{T}+1$;

repeat

AddLinkActivationSet($G,\ \mathbf{\Phi},\ k,\ A_{\pi(q)},\ \boldsymbol{p},\ f(k),\ \boldsymbol{S}_{eq}$);

$k=k+1$;

$q=q+1$;

until $(p_n^{(k)}=T_{eq}, \forall n\in N)$;

end

Function `AddLinkActivationSet`

Add a set of links to the asynchronous schedule $\boldsymbol{S}$

input : $G(N,E)$, $\boldsymbol{\Phi}$, k, $LINKSET$, $\boldsymbol{p}^{(k-1)}$, $\boldsymbol{S}$

output : $\boldsymbol{p}^{(k)}$, $f(k)$, $\boldsymbol{S}$

local : $ACTIVATED\_NODES\_SET = \{\ \}$

begin

1 **for** *every link* l *in* $LINKSET$ **do**

$i = l.master,\ j = l.slave$;

Add i and j in $ACTIVATED\_NODES\_SET$;

if $(p_i^{(k-1)} == 0$ *AND* $p_j^{(k-1)} == 0)$ **then**

/*This is the first activation for both nodes*/;

$p_i^{(k)} = p_i^{(k-1)} + 1,\ \ \boldsymbol{S}_i(p_i^{(k)}) = l$;

$p_j^{(k)} = p_i^{(k)} + \frac{\phi_l(1+\phi_l)}{2},\ \ \boldsymbol{S}_j(p_j^{(k)}) = l$;

end

if $(\boldsymbol{S}_i(p_i^{(k-1)}) == l$ *AND* $\boldsymbol{S}_j(p_j^{(k-1)}) == l)$ **then**

/*Case A*/ ;

$p_i^{(k)} = p_i^{(k-1)} + 1$;

end

else

if $(p_i^{(k-1)} > p_j^{(k-1)})$ **then**

/*Case B*/ ;

$p_i^{(k)} = p_i^{(k-1)} + 1$;

end

else

/*Case C*/ ;

$p_i^{(k)} = p_j^{(k-1)} + \frac{\phi_l^2 - \phi_l + 2}{2}$;

end

end

$\boldsymbol{S}_i(p_i^{(k)}) = l$;

for *any unassigned* $t \in p_i^{(k-1)} + 1, ..., p_i^{(k)} - 1$ **do**

$\boldsymbol{S}_i(t) = idle$;

end

$p_j^{(k)} = p_i^{(k)} + \frac{\phi_l(1+\phi_l)}{2}$;

$\boldsymbol{S}_j(p_j^{(k)}) = l$;

for *any unassigned* $t \in p_j^{(k-1)} + 1, ..., p_j^{(k)} - 1$ **do**

$\boldsymbol{S}_j(t) = l$;

end

end

for *every* n *not in* $ACTIVATED\_NODES\_SET$ **do**

$p_n^{(k)} = p_n^{(k-1)}$;

end

$f(k) = \max_{n \in N}\{p_n^{(k)}\}$;

end

Function `GetForwardProgress`

Compute forward progress on $\boldsymbol{S}$ due to $LINKSET$

input : $\boldsymbol{\Phi}$, k, $\boldsymbol{p}^{(k-1)}$, $LINKSET$, $\boldsymbol{S}$

local : f, $\boldsymbol{p} = [p_n]$, $n = 1, .., N$,
$ACTIVATED\_NODES\_SET = \{\ \}$

begin

1 **for** *every link* l *in* $LINKSET$ **do**

$i = l.master$, $j = l.slave$;

Add i and j in $ACTIVATED\_NODES\_SET$;

if $(p_i^{(k-1)} == 0\ AND\ p_j^{(k-1)} == 0)$ **then**

$p_i = p_i^{(k-1)} + 1,\ p_j = p_i + \frac{\phi_l(1+\phi_l)}{2}$;

end

if $(\boldsymbol{S}_i(p_i^{(k-1)}) == l\ AND\ \boldsymbol{S}_j(p_j^{(k-1)}) == l)$ **then**

$p_i = p_i^{(k-1)} + 1$;

end

else

if $(p_i^{(k-1)} > p_j^{(k-1)})$ **then**

$p_i = p_i^{(k-1)} + 1$;

end

else

$p_i = p_j^{(k-1)} + \frac{\phi_l^2 - \phi_l + 2}{2}$;

end

end

$p_j^{(k)} = p_i^{(k)} + \frac{\phi_l(1+\phi_l)}{2}$;

end

for *every* n *not in* $ACTIVATED\_NODES\_SET$ **do**

$p_n = p_n^{(k-1)}$;

end

$f = \max_{n \in N}\{p_n\}$;

return f;

end

Function `FindHperm`

Phase I of MIN_PROGRESS that finds permutation $\boldsymbol{\pi}_h$

input : $\boldsymbol{\Phi}$, $\widetilde{\boldsymbol{S}} = \{M^{\alpha}, \lambda^{\alpha} : \alpha = 1, .., |M(\widetilde{\boldsymbol{S}})|\}$, $\widetilde{T}$

output : $\boldsymbol{\pi}_h$: $\boldsymbol{\pi}_h(k)$ contains the activation set index selected at iteration k

local : $\boldsymbol{U}$, $currmin$, f, $\boldsymbol{p} = [p_n^{(k)}]$

$\boldsymbol{S}$: $dummy\ asynchronous\ schedule$

Initialization: $\boldsymbol{U} = \{M^1, .., M^{|M(\widetilde{\boldsymbol{S}})|}\}$;

begin

1 **for** $k = 1$ *to* $|M(\widetilde{\boldsymbol{S}})|$ **do**

$currmin = 3|M(\widetilde{\boldsymbol{S}})|$;

for *every set* $M^g \in \boldsymbol{U}$ **do**

f=GetForwardProgress($\boldsymbol{\Phi}$, k, $\boldsymbol{p}^{(k-1)}$, M^g, $\boldsymbol{S}$);

if *(*$currmin < f$*)* **then**

$\boldsymbol{\pi}_h(k) = g$;

currmin = f ;

end

end

AddLinkActivationSet(G, $\boldsymbol{\Phi}$, k, $M^{\boldsymbol{\pi}_h(k)}$, $\boldsymbol{p}$, f, $\boldsymbol{S}$);

$\boldsymbol{U} = \boldsymbol{U} - \{M^{\boldsymbol{\pi}_h(k)}\}$;

end

end

Procedure MIN_PROGRESS

input : $G(N, E), \mathbf{\Phi}, \widetilde{\boldsymbol{S}} = \{M^{\alpha}, \lambda^{\alpha}\}, \widetilde{T}$

output : $\boldsymbol{S}$: The asynchronous schedule computed by the heuristic

local : $\boldsymbol{\pi}_h$: permutation of the activation sets M^{α}

begin

1 /*Phase I*/ ;

FindHperm($\mathbf{\Phi}, \widetilde{\boldsymbol{S}}, \widetilde{T}, \boldsymbol{\pi}_h$);

Form $\widetilde{\boldsymbol{S}}^{(\boldsymbol{\pi}_h)}$ from $\widetilde{\boldsymbol{S}}$ using $\boldsymbol{\pi}_h$ for the ordering

of sets M^{α} and activating the λ^{α} instances of

each set M^{α} in consecutive slots.;

/*Phase II*/ ;

EQUIVALENT($G, \mathbf{\Phi}, \widetilde{\boldsymbol{S}}^{(\boldsymbol{\pi}_h)}, \boldsymbol{\pi}_h, \boldsymbol{S}$);

end

Chapter 4

A distributed asynchronous TDMA protocol

The main advantages of TDMA over random access are conflict-free transmissions and strict bandwidth allocation guarantees. However, implementing TDMA in the distributed ad hoc network setting has been a notoriously challenging task. As a result most TDMA protocols rely on one or more of the following global assumptions: network-wide slot synchronization, universal slot enumeration or a priori knowledge of the number of nodes in the network. In addition, TDMA protocols providing bandwidth allocation guarantees typically require knowledge the network topology in its entirety.

In this chapter we introduce a TDMA protocol that does not rely on any global assumptions. The protocol is completely distributed, asynchronous and traffic adaptive: in response to asynchronous local events such as traffic or topology changes, nodes reassign slots to their adjacent links using only local information. The protocol can be executed simultaneously in different parts of the network. It ensures that the network TDMA schedule remains free of transmission conflicts despite the concurrent slot reassignments.

Being TDMA-based, the protocol can potentially be used to provide rate guarantees to the network links. This involves computation of a TDMA schedule that realizes

a given set of link slot demands. The demands input to the TDMA protocol must be feasible–there must exist a network TDMA schedule of length less than the system period that can realize them. As we saw in Chapter 3, feasibility determination of an arbitrary demand allocation is a NP-complete problem, even if global information is available. Our approach is based on the fundamental observation that, in practice, the link demands will not be arbitrary but will be locally generated by bandwidth allocation algorithms running at the nodes. We are therefore interested in a subset of feasible allocations that can be characterized by a set of local conditions. The local conditions give rise to the node QoS utilization parameter–a sufficient number of slots each node can provide as demands to its adjacent links in order for feasibility to be ensured at all times. It turns out that the QoS utilization parameter depends on both the existence (or lack thereof) of slot synchronization and the degree of topology control. This introduces an interesting trade-off between topology restrictions and the fraction of feasible allocations that can be captured by the local conditions.

The rest of the chapter is organized as follows: Section 6.4 provides an overview of state of the art distributed TDMA protocols for ad hoc networks and identifies the most important issues involved in their design. Section 4.2 introduces the multi-channel TDMA access architecture and control structure used by the distributed TDMA protocol. The protocol is presented and analyzed in Section 4.3. Section 4.4 introduces the dynamic link scheduling problem and elaborates on the determination of local feasibility conditions in various settings. Section 4.5 concludes.

4.1 Related work

Early efforts for traffic-aware TDMA protocols aimed to realize a given set of slot demands on nodes or links. Since the problem of computing the minimum-length schedule realizing the demands is NP-complete, these protocols rely on heuristics that compute sub-optimal schedules [54][55][84][56]. These heuristics require global knowledge of network topology and traffic requirements. They can be incorporated in a TDMA protocol for a (slot-synchronized) wireless ad hoc network as follows: every node sends its connectivity or link demand changes to a central controller. Consequently, the controller computes and distributes the new TDMA schedule back to the nodes. In absence of a central controller each node can broadcast the locally observed changes to the entire network. Each node then uses an identical copy of the centralized algorithm to compute a TDMA schedule for the new demand allocation and then communicate on the part of the schedule that corresponds to its own locality. This idea was utilized in [90] to produce such a "distributized" version of the centralized heuristic in [55].

Two drawbacks are associated with centralized TDMA protocols in ad hoc networks. First, an often unnecessarily high communication and computation overhead are incurred: a single change in topology or demand triggers network-wide broadcasts and global schedule recomputation from scratch. Second, the TDMA protocol uses a variable system period–equal to the computed schedule length for the demands at hand[1]. Hence, each time a change occurs, communications in the network must be suspended until the new system period and TDMA schedule are determined. Both problems become more acute as the network size increases and as changes become more frequent: If network dynamics occur too fast for the system to react the result is excessive com-

[1]If a fixed system period of T_{system} slots were used, the system should support an abort mechanism in case the computed schedule length exceeds T_{system} slots.

munication overhead and extended network downtime.

A parallel line of research focuses on TDMA protocols where a fixed system period is used and nodes coordinate transmissions using only a local view of the network. These distributed protocols do not attempt to realize specific traffic demands. Instead, emphasis is placed on constructing a conflict-free network TDMA schedule. This is a challenging task because nodes have access to local information and may independently assign conflicting slots during this process. Equally challenging to the schedule construction is the maintenance of its conflict-free property in the face of topology changes.

One of the first distributed protocols in this family is the Link Activation Algorithm (LAA) [91] developed for link scheduling in multi-channel ad hoc networks. LAA is executed in two N-slot control frames: during slot i of the first frame, node i constructs its own conflict-free link schedule by taking into account the schedules previously broadcast by lower indexed nodes within range; it then broadcasts this schedule to its neighbors. During the second N-slot frame the nodes resolve any scheduling conflicts that occurred when they set up their schedules independently during the first frame. LAA is simple fast and robust but rather inflexible–the established schedule depends only on the relative order of node identities in the network. A more sophisticated algorithm was later proposed in [92] where the nodes periodically reorganize the TDMA schedule based on local traffic observations. Traffic-adaptive distributed TDMA protocols have also been proposed for single-channel systems for both link scheduling [93] as well as broadcast (node) scheduling [94][62].

Keeping the network TDMA schedule free of transmission conflicts may result in excessive control overhead. Chlamtac and Farago [95] and Ju and Li [96] relax on the conflict-free requirement and develop TDMA protocols that are topology-transparent. Each node uses a precomputed TDMA node schedule of period T_{system} slots ($T_{system} <$

N). The schedule of each node is unique to the node's identity and has been computed using estimates on the maximum projected number of one-hop neighbors per node as well as number of nodes in the network. Although conflicts are allowed, the schedule computation guarantees a node to transmit conflict-free on at least one slot within a period of T_{system} slots regardless of the current topology structure. Since the schedules are precomputed, no control overhead exists for schedule maintenance. However, the achieved throughput is sensitive to the choice of parameters (maximum node degree and maximum number of nodes) and can be very low for certain scenarios. The performance of the TDMA protocol in [95] (called Time Spread Multiple Access (TSMA)) was experimentally compared to Carrier Sense Multiple Access (CSMA) in [97]. Interestingly, CSMA, a pure random access protocol, outperformed TSMA in terms of both throughput and delay in most tested scenarios. This gives an indication that topology transparent TDMA protocols are not necessarily better than pure random access protocols.

All the TDMA protocols mentioned above assume a mechanism that maintains network-wide slot synchronization. The emergence of Bluetooth-based ad hoc networks (termed "scatternets") created the need for distributed TDMA link scheduling protocols that do not rely on this assumption. The approaches for scatternet scheduling can be categorized into hard and soft coordination protocols. Hard coordination protocols [81] attempt to establish perfectly conflict-free link schedules. The advantage is that they can provide strict bandwidth allocation guarantees since no transmission conflicts exist. However, maintenance of the conflict-free property may come at the expense of communication overhead when there are dynamic changes in the network. On the other hand, soft coordination schemes [79][80] are the link scheduling analog of the topology-transparent protocols in [95] and [96]: they trade off perfectly conflict-free transmissions for lower complexity. The downside: occasional transmission conflicts and lack of abil-

ity to provide strict bandwidth allocation guarantees; hence loss of the main advantages of TDMA over random access.

The above discussion provides some practical guidelines for design of TDMA protocols for ad hoc networks:

- To avoid network downtime, the period of the TDMA schedule must be fixed.
- Since our goal is provision of QoS guarantees, the network TDMA schedule must be free of transmission conflicts. In this case, care must be taken to ensure the control overhead for schedule maintenance is minimized.
- It is desirable for the protocol to not require global knowledge including network-wide slot synchronization, universal slot enumeration or a priori knowledge of the number of nodes in the network.
- During protocol operation nodes should have access to only local information.

4.2 TDMA architecture

4.2.1 Signaling and local TDMA schedule structure

The system uses multiple channels for communication and a universal channel for neighborhood discovery. Each channel can be implemented as a distinct frequency band or spread spectrum code (Frequency Hopping (FH) sequence or Direct Sequence (DS) code).

Since neighborhood discovery is not communication-intensive we will assume it is implemented using a simple random access protocol (e.g. ALOHA) on a separate low-

cost transceiver[2]. Hence, each wireless node has two transceivers: one dedicated to neighborhood discovery; the other to communications.

When two nodes within range discover each other they may decide to establish a communication link. This decision can be the result of a simple policy (e.g. establishment of every discovered link or establishment of up to D_{max} links per node) or the result of a distributed topology control protocol that ensures certain global network topology properties (e.g. bipartite or tree structures).

When established, links must be assigned communication channels such that no secondary conflicts exist in the network. One way to achieve this is to associate every node with a unique channel; if each link is assigned the channel of one of the node endpoints, then, all transmissions satisfying the primary interference constraints will occur in different channels. Bluetooth implements this method using spread spectrum signaling. Each node is associated with a unique frequency hopping (FH) sequence derived from its unique MAC address. Upon link establishment, one of the node endpoints is assigned as master and the other as slave. The link is assigned the FH sequence of the master. Although not orthogonal, Bluetooth FH sequences have been shown to perform well in practice [46]. Interference can be further mitigated using distributed assignment mechanisms that minimize the number of FH channels per locality [42][45][25]. Secondary interference can also be avoided if nodes within two wireless hops of each other are assigned orthogonal channels–if D_{max} is an upper bound on the intended adjacent links per node, a total of $2D_{max}(D_{max}-1)+1$ (instead of N) channels are needed [23]. References [23][47] propose distributed dynamic algorithms performing such assignments.

Although secondary interference can be avoided using one of the above techniques,

[2] Alternatively, the discovery and communication functions can be integrated using a single transceiver at the expense of increased scheduling complexity.

primary interference is always present: since each wireless node has a single communication transceiver it can transmit or receive to only a single channel and link at a time. Each node uses a local T_{system}-slot periodic schedule to coordinate transmissions on its adjacent links. The system supports both slot-synchronized and asynchronous modes of operation. In the synchronized mode all local schedules are perfectly slot-aligned using either additional hardware (e.g. GPS clocks) or a separate protocol (e.g. the Network Time Protocol (NTP)). In the asynchronous mode the local schedules are not slot aligned; time slot reference for communication on a link is provided upon link establishment by the master node endpoint. Each communication slot consists of two mini-slots that support that support full-duplex transfer–one for master-to-slave transmission and the other from slave-to-master transmission.

The nodes use a distributed TDMA protocol to reassign slots to their adjacent links in response to locally observed asynchronous events such as changes in topology or traffic requirements. Due to asynchronous nature of the events the protocol can be executed simultaneously in different parts of the network. Since the nodes decide to reassign slots based on only local information, the protocol needs to 1) coordinate initiation of slot reassignments on a link by both node endpoints and 2) ensure the network TDMA schedule remains conflict-free despite the simultaneous slot reassignments. The protocol is also used to assign an initial number of conflict-free slots to a link that has just been discovered and needs to be established.

4.2.2 Exchanging control information

When nodes execute the protocol they must exchange control messages to keep their local schedules consistent and hence preserve the conflict-free property of the entire TDMA schedule. According to the previous section, the predominant method for ex-

changing control information is by splitting the TDMA schedule of T_{system} slots into control and data parts of $T_{control}$ and T_{data} slots, respectively.

Apart from the issues regarding global slot synchronization and universal slot enumeration mentioned in Chapter 1, deciding on the relative sizing of the control and data parts is not trivial. Clearly, setting $T_{control} = N$ and using TDMA on the control part does not scale with network size. Alternatively, $T_{control} < N$ and the control part is shared using random access, $T_{control}$ must be chosen large enough to minimize the control overhead and small enough to ensure timely delivery of control messages. This design problem has been considered in [62].

Note that these approaches aim at each node sending conflict-free only a single control packet during the control part. If the coordination protocol requires multiple control messages per link rate adjustment, multiple system periods will be required. This increases the protocol response to network dynamics.

An alternative approach to splitting the TDMA communication schedule in control and data part is to use a separate transceiver and channel, as well as a simple access mechanism (e.g. ALOHA) for the exchange of control information [98]. However, random access cannot guarantee packet delivery in a timely manner–the protocol may respond very slowly to the network dynamics.

According to our approach, each node uses its current local TDMA communication schedule for the exchange of control information. However, the local schedule is not split in a control and data part. Instead any slot can be used for transmission of either a control packet or a data packet. To speed up the schedule modification process, control packets are given transmission priority over data packets.

4.3 The distributed TDMA protocol

4.3.1 Overview

The protocol allows each node to be involved in at most one link slot reassignment at a time. A node conveys its busy status to its neighbors using a local one-bit variable, called BUSY_BIT. The current value of BUSY_BIT is copied to the corresponding field of every outgoing control or data packet. In addition to its own BUSY_BIT, each node maintains the BUSY_BIT of its neighbors using a local variable, called NEIGHBOR_BBIT_VEC.

Rate adjustment on a link l can be initiated when none of its node endpoints is currently busy on a rate adjustment of other links. Upon initiation, both endpoints set their BUSY_BIT variables. Then, they exchange their current local schedules using SC_INFO control packets. This information aids one of the endpoints to determine the new set of slot positions to be assigned to this link.

Each endpoint stores the new slots for link l in a variable called LOCK_VEC. Some of these slots may be currently assigned to other links adjacent to the node endpoints and need to be canceled.

Each endpoint signals schedule modifications to all its affected neighbors using SC_UPD packets. A SC_UPD packet transmitted on a link contains the new slot positions to refresh the old ones for this link in the recipient's local schedule. The recipient of a SC_UPD packet updates its local schedule accordingly and acknowledges with a SC_UPD_ACK packet.

After all affected neighbors acknowledge their schedule modifications, the endpoints update their own local schedules by assigning the new slot positions (stored in their LOCK_VEC variables) to link l. Finally, the endpoints become available for rate adjust-

ment on other links by clearing their BUSY_BIT and LOCK_VEC variables.

4.3.2 Detailed operation

Consider a node u that receives a request for rate adjustment for link $l = (u, v)$. Such a request can be triggered either by a timer expiration or by an explicit higher layer notification.

First, node u sets its BUSY_BIT and becomes unavailable for rate adjustment on other adjacent links. Then, it inspects NEIGHBOR_BBIT_VEC(l) (constantly updated by incoming packets from v). If NEIGHBOR_BBIT_VEC(l)=1, then v is currently engaged on a rate adjustment on another of its own adjacent links. Node u must wait until that rate adjustment ends. Then, the following steps are performed:

1. Node u computes a rate estimate (in # of slots) for link (u, v) and sends a SC_INFO packet to node v containing its rate estimate and current local schedule.

2. Upon reception of the SC_INFO packet, node v sets its own BUSY_BIT and computes its own rate estimate for the link. It then uses a local election rule to determine whether it will be the ASSIGNER or ASSIGNEE for this rate adjustment. The local rule can be based on the rate estimates of the SC_INFO packets, the endpoint addresses, or even the parent-child relationship of the endpoints (in the case of a tree topology). If node v determines itself to be the ASSIGNEE, it replies with a SC_INFO packet to node u; otherwise it performs the ASSIGNER action described next.

3. Using the information in the SC_INFO packet of the ASSIGNEE, the ASSIGNER node a) decides the new rate for link (u, v) and b) performs a **slot assignment**

algorithm that determines the new slot positions for link (u, v) in the endpoints' local schedules that will realize the new rate .

4. The ASSIGNER stores the new slot positions in its LOCK_VEC. Since the local schedules of the node endpoints are not generally slot-synchronized, the ASSIGNER computes a set of time-overlapping slot positions with respect to the slot offset of the ASSIGNEE. It then sends the translated slot positions to the ASSIGNEE via a SC_UPD packet and waits for acknowledgement.

5. Upon reception of the SC_UPD packet from the ASSIGNER, the ASSIGNEE stores the new slot positions in its own LOCK_VEC. Some of the new positions may indicate that certain slots currently assigned to other adjacent links must now be assigned to link l in the local schedule of the ASSIGNEE. In this case, the ASSIGNEE sends SC_UPD packets on the affected links and waits for acknowledgements. Finally, the ASSIGNEE replies to the ASSIGNER with a SC_UPD_ACK packet.

6. Upon reception of the SC_UPD_ACK packet from the ASSIGNEE, the ASSIGNER sends SC_UPD packets to its own affected neighbors (the ASSIGNEE excluded) and waits for acknowledgements.

7. For each SC_UPD_ACK packet received by an affected neighbor, each endpoint sets the corresponding slots as idle in its own local schedule. Without loss of generality, let u be the first node endpoint of link (u, v) that receives all acknowledgements from its affected neighbors. Node u sends a COMMIT_REQ packet to node v and waits for acknowledgment.

8. When node v receives the COMMIT_REQ packet, as well as *all* SC_UPD_ACK packets from its affected neighbors:

(a) it assigns the new slot positions (stored in $LOCK\_VEC$) to link (u, v),

(b) it sends a COMMIT_ACK packet to node u, and

(c) waits for acknowledgement.

9. Upon reception of the COMMIT_ACK packet, node u assigns to link (u, v) the new slot positions (stored in its own $LOCK\_VEC$), and sends a COMMIT_ACK packet to v. Then, it clears its BUSY_BIT and LOCK_VEC variables, thus becoming available for rate adjustment on other adjacent links.

10. Upon reception of the COMMIT_ACK packet from u, node v clears its own BUSY_BIT and LOCK_VEC variables and becomes available for rate adjustment on other adjacent links.

11. Rate adjustment of link (u, v) is complete.

4.3.3 Properties

The protocol has the following properties:

Property 4.3.1 *The network TDMA schedule is always free of transmission conflicts, despite the simultaneous rate adjustments.*

Proof We define a link as *busy* with respect to rate adjustment if both its node endpoints are currently busy for rate adjustment on this link. According to the protocol, each node can be busy at only one adjacent link at a time. In addition, the BUSY_BIT precludes the neighbors of each endpoint to initiate a rate adjustment on a non-busy link. Hence, at any time no busy links have common node endpoints–the set of busy links is a matching on the network topology graph.

Let $B^{(t)}$ be the set of busy links in the network at time t and $NB^{(t)}$ be the set of non-busy links adjacent to the node endpoints of the links in $B^{(t)}$. Depending on the slot reassignments, the rates of the links in $B^{(t)}$ may either increase or decrease; however, the rates of the links in $NB^{(t)}$ can never increase–they may either remain intact or decrease due to their slots being reassigned to their adjacent busy links. Hence the protocol allows *simultaneous* re-assignments of *concurrent* slots only on a matching in the topology graph; hence the TDMA schedule conflict-free property is maintained.

Note it is possible for a node u currently busy on link (u, v) to receive a SC_UPD packet cancelling slots on a non-busy link (u, x) due to a rate adjustment on another link (x, y) (both $x, y \neq v$). Since a slot cancellation request on a non-busy link does not affect slot re-assignments on busy link (u, v), node u sets these slots idle in its local schedule and responds to node x with a SC_UPD_ACK packet. In addition, upon reception of a SC_UPD_ACK on a non-busy link it has requested slot cancellation, busy node x can immediately set these slots as idle in its local schedule. Stored in the LOCK_VEC of x, these slots will be re-assigned to busy link (x, y) once its rate adjustment is complete.

Finally, according to the protocol, the node endpoints reassign slots on their busy link only after having received acknowledgements from the all its affected neighbors. Thus, it is ensured that all local schedules affected by the slot reassignments on the busy link will be conflict-free and consistent after the update. ■

Property 4.3.2 *If the network degree is D_{max} the maximum number of control packets needed for each link rate adjustment is $5 + 2D_{max}$.*

Proof The protocol needs at most 2 SC_INFO packets, 2 packets for the SC_UPD-SC_UPD_ACK packet exchange on the busy link, 1 COMMIT_REQ packet, and 2 COMMIT_ACK packets. If both node endpoints have degree D_{max} and if all their

neighbors are affected by the slot assignments on the busy link, $2(D_{max} - 1)$ packets are needed for the SC_UPD-SC_UPD_ACK packet exchanges on the non-busy links.

■

Property 4.3.3 *If no packets are lost due to channel errors, the maximum duration of a link rate adjustment is $5T_{system}$ slots.*

Proof We focus on a busy link (u, v) and its one-hop neighborhood as shown in Figure 4.1. Without loss of generality, let the system be slot-synchronized. We assume that slot

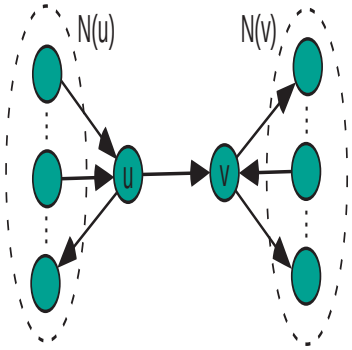


Figure 4.1: The busy link (u, v) and its one-hop neighborhood. The one-hop neighbors of u and v are denoted by $N(u)$ and $N(v)$, respectively. Arrows denote master-slave relationships.

reassignments are such that each link is always assigned at least one slot in the local TDMA schedules of its node endpoints. This condition can be maintained because the ASSIGNER is aware of the local schedule of the ASSIGNEE when it determines the new slot positions for the busy link. To derive the maximum duration of the link rate adjustment, we will assume that each link is assigned exactly once within a period of the TDMA schedule.

The rate adjustment starts when the first SC_INFO packet is sent on the busy link and terminates when the last COMMIT_ACK packet is received on the busy link.

Without loss of generality, let the busy link (u, v) be assigned slot 0 and the rest of its adjacent links be assigned subsequent slots (one slot each) within the TDMA schedule period. Recall that each slot in the TDMA schedule is full-duplex–in the first part the master polls; in the second part the slave responds. The duration of the link rate adjustment will depend on the master-slave role assignment on the busy link (u, v) as well as the master-slave role assignments on the non-busy links adjacent to nodes u and v.

We derive the maximum duration by constructing a scenario that maximizes the delay of each stage of the protocol execution. Since we have assumed that each link is assigned one slot within a system period, the protocol execution will require multiple TDMA cycles:

Cycle 1**:** Let u initiate the link rate adjustment by sending the first SC_INFO packet at slot 0. If u is master on link (u, v), the SC_INFO packet will be sent in the first half of slot 0 and v will reply in the second half; if u is master on link (u, v), v will reply in the the first part of slot 0 of the next cycle. Hence, to maximize delay, we will assume that u is slave on link (u, v).

Cycle 2**:** Having received the SC_INFO packet from node u, node v determines its ASSIGNEE or ASSIGNER role. In the first case, it will delegate the ASSIGNER responsibility to node u by replying with a SC_INFO packet; in the second case, it will act as ASSIGNER and initiate the slot reassignments. Clearly, the first case maximizes delay and will be assumed in the steps that follow. Upon receiving the SC_INFO packet in the first half of slot 0, (slave) node u will act as ASSIGNER and send a SC_UPD packet to v in the second half of slot 0.

When v receives the SC_UPD packet, it will notify the set of its affected one-hop neighbors $A(v)$ ($A(v) \subseteq N(v)$) about the schedule changes using SC_UPD packets. If

v is master to *all* its affected adjacent links, it will receive all acknowledgments within this cycle. If v is slave in at least one affected link, all acknowledgements will arrive by the next TDMA cycle. Since we seek the maximum delay, we will assume that node v is slave to at least one link and this link has been affected by the slot reassignment on busy link (u, v).

Cycle 3**:** Node v replies to node u with a SC_UPD_ACK packet. Then, node u sends SC_UPD packets to all its affected neighbors. To maximize delay will also assume that node u is slave to at least one link and this link has been affected by the slot reassignment on busy link (u, v). In addition, node v receives all SC_UPD_ACK packets from its affected neighbors by the end of this cycle.

Cycle 4**:** Node v sends a COMMIT_REQ packet to u at slot 0. In addition, node u receives all SC_UPD_ACK packets from its affected neighbors by the end of this cycle.

Cycle 5**:** Once the COMMIT_REQ and all SC_UPD_ACK packets have been received, node u sends a COMMIT_ACK packet to v at the second half of slot 0.

Cycle 6: Node v completes the rate adjustment by sending a COMMIT_ACK packet to node u at the first half of slot 0.

In the above scenario, the rate adjustment starts at the second half of slot 0 of cycle 1 and terminates at the first-half of slot 0 of cycle 6. Hence, the overall delay equals $5 \cdot T_{system}$ slots. The delay is maximum because the scenario was constructed by considering the maximum delay case at every stage of the protocol execution. ■

4.3.4 Design considerations

Storage requirements

Let D_{max} be the maximum number of adjacent links per node. Given D_{max}, a node can distinguish its adjacent links using $\lceil log_2 D_{max} \rceil$ bits. Hence, the local schedule can be

	slot 0		slots 1-$T_{system}-1$			
cycle#	$v->u$	$u->v$	$N(v)->v$	$v->N(v)$	$N(u)->u$	$u->N(u)$
1		SC_INFO				
2	SC_INFO	SC_UPD		SC_UPD		
3	SC_ACK		SC_ACK			SC_UPD
4	COM_REQ				SC_ACK	
5		COM_ACK				
6	COM_ACK					

Figure 4.2: Scenario that maximizes the delay of link rate adjustment.

encoded using $\lceil log_2 D_{max} \rceil \cdot T_{system}$ bits.

In addition to the local schedule, BUSY_BIT, NEIGHBOR_BBIT_VEC and LOCK_VEC require 1, D_{max}, and T_{system} bits, respectively. Hence, the total storage for local variables required by the protocol is:

$$B_{storage} = 1 + D_{max} + (\lceil log_2 D_{max} \rceil + 1) \cdot T_{system} \ bits \tag{4.1}$$

Communication requirements

Each half-duplex mini-slot can be used by either a data or control packet. Hence, each packet has a header that includes the packet type (DATA, SC_INFO, SC_UPD, COMMIT_REQ, COMMIT_ACK) and the BUSY_BIT state of the sender node. This information can be encoded using 4 bits.

A SC_INFO control packet contains the local schedule and the rate estimate of the sender node. The rate estimate range is $[1, T_{system}]$ slots and can be encoded by $\lceil log_2 T_{system} \rceil$ bits. Hence, the total size of a SC_INFO packet is $T_{system} \cdot \lceil log_2 D_{max} \rceil + \lceil log_2 T_{system} \rceil$ bits.

A SC_UPD control packet sent over link l indicates the new slot positions to be assigned to this link in the recipient's local schedule. This information can be encoded using T_{system} bits. The rest of the control packets (SC_UPD_ACK, COMMIT_REQ, COMMIT_ACK, CANCEL_REQ) contain no extra information other than their type.

If we include the 4 bits of the common header, the number of bits $B_{control}$ required per control packet is determined by the size of the SC_INFO packets:

$$B_{control} = 4 + \lceil log_2 T_{system} \rceil + \lceil log_2 D_{max} \rceil \cdot T_{system} \ \ bits \tag{4.2}$$

Since a slot can carry either a data or control packet, eq. (4.2) sets the minimum half-duplex mini-slot size in the system or equivalently, the maximum system period T_{system} that can be supported given a fixed system slot duration.

4.4 Link-level Quality of Service (QoS)

At any time instant the distributed TDMA protocol guarantees that the network operates according to a conflict-free TDMA schedule. Being conflict-free this schedule realizes a link slot allocation in the network. In order to provide QoS on a link-level basis we must be able to compute a TDMA schedule that realizes a given demand allocation on the network links. We are interested in a dynamic version of this problem where link demands may change at asynchronous time instants due to a higher layer process. This process can be a higher-layer bandwidth allocation mechanism or even mobility–in this case link failure is viewed as transition to zero link demand and link establishment as a transition from zero to positive demand.

We assume that the higher-layer process alternates between two states: an active state where the link demands change and a quiescent state where no changes occur. The end of each active state corresponds to a link demand allocation to be realized by a

network TDMA schedule. The challenge: nodes must reach such a schedule starting from the current TDMA schedule and using only local information.

The alternating states model is necessary for the definition of convergence. It implies that network topology and traffic dynamics must remain stable for a sufficient time period to allow realization of the desired allocation. However, the nodes are not aware which of the two states the network is currently in. They can only detect the demand changes on their adjacent links. Once the link demands stabilize, the nodes use the distributed TDMA protocol converge to a TDMA schedule realizing these demands.

4.4.1 Local feasibility conditions

In order for convergence to occur, the distributed TDMA protocol must always be provided with feasible link demands. Recall that a link demand allocation $\boldsymbol{\tau} = (\tau_1, ..., \tau_l, ..., \tau_{|E|})$ is feasible if a conflict-free TDMA schedule exists that can allocate τ_l conflict-free slots to every link l without exceeding the system period (T_{system} slots). In Chapter 3 we saw that determining feasibility of an arbitrary set of link rates in a slotted multi-channel ad hoc network is an NP-complete problem. However, in practice, the higher layer process changing the link demands will not be global but, in fact, the result of higher-layer bandwidth allocation or hand-off mechanisms executed locally by the nodes. We are therefore interested in identifying certain instances where feasibility can be characterized by a set of local conditions.

Let us first assume that global slot synchronization is supported in the network. In this case local conditions would require the demand sum of the links adjacent to each node to not exceed T_{system} slots. Due to link scheduling interdependence, these local conditions cannot alone guarantee feasibility (see Fig. 4.3 for an example). The additional non-local conditions require that, for every odd node subset Q ($|Q| > 1$) in

the topology graph, the sum of the demands of all links adjacent to the nodes in Q must not exceed $\lfloor (|Q|-1)/2 \cdot T_{system} \rfloor$ slots [99].

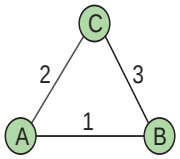


Figure 4.3: Without loss of generality, assume that all nodes are slot-synchronized and T_{system} is even. No schedule exists that can allocate $T_{system}/2$ conflict-free slots to each link, even if the local conditions $\tau_1 + \tau_2 \leq T_{system}$, $\tau_1 + \tau_3 \leq T_{system}$ and $\tau_2 + \tau_3 \leq T_{system}$ for nodes A, B, C, respectively allow this allocation. The non-local condition $\tau_1 + \tau_2 + \tau_3 \leq T_{system}$ is also required here.

There are two ways to guarantee feasibility using only local conditions: restrict the network topology or underutilize the network. If the network topology is bipartite, the entire set of feasible allocations can be captured only by local conditions. Topology control is inherent in multi-channel systems due to the need to assign channels to the discovered links before communication takes place. Alternatively, in absence of a topology control mechanism, feasibility can be ensured by restricting the maximum number of slots each node provides to its adjacent links. For slot-synchronized multi-channel ad hoc networks feasibility is guaranteed by requiring the sum of link demands on every node be less than $\lfloor 2/3 \cdot T_{system} \rfloor$ slots at any time [100]. Local conditions of this form are sufficient: they guarantee feasibility but only capture a fraction of the entire set of feasible allocations. Hence, the network is underutilized in this case.

In an asynchronous TDMA system the region of feasible rates is further restricted. Due to the additional slots needed in the slaves' local schedules, the minimum period $L_{min}(\boldsymbol{\tau})$ realizing a demand allocation $\boldsymbol{\tau}$ in an asynchronous system is greater than the

minimum period $L_{min}^{synch}(\boldsymbol{\tau})$ had the system been perfectly synchronized. Since feasibility is characterized by comparing the minimum schedule length that can realize $\boldsymbol{\tau}$ to the system period T_{system}, certain allocations feasible by a synchronized system will not be feasible when asynchronicity is present.

In Chapter 3, we derived the corrollary on feasibility of the EQUIVALENT algorithm. The corollary states, that, for any given topology and demand allocation $\boldsymbol{\tau}$, $L_{min}(\boldsymbol{\tau}) \leq 2 \cdot L_{min}^{synch}(\boldsymbol{\tau})$ [74]. Consequently, a set of sufficient local feasibility conditions would allow for nodes to offer half the slots they would offer in the corresponding synchronized system: For bipartite topologies, feasibility is guaranteed if every node offers $\lfloor 1/2 \cdot T_{system} \rfloor$ slots while for arbitrary topologies $\lfloor 1/3 \cdot T_{system} \rfloor$ slots. These conditions imply further network underutilization–$1/2$ and $2/3$ of the total capacity of bipartite and arbitrary topologies, respectively cannot be used for QoS provision.

A lower bound on the minimum period $L_{min}(\boldsymbol{\tau})$ of an asynchronous TDMA schedule realizing a demand allocation $\boldsymbol{\tau} = (\tau_1, ..., \tau_l, ..., \tau_{|E|})$ is given by:

$$LB(\boldsymbol{\tau}) = \max_{u \in N} \sum_{l \in L(u)} (\tau_l + J_l^{(u)}) \tag{4.3}$$

where $L(u)$ is the set of links adjacent to node u and,

$$J_l^{(u)} = \begin{cases} 1 & \text{if } u \, is \, slave \, on \, link \, l \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

The term τ_l in the sum of the RHS of (4.3) exists because each node can communicate to only a single link at every slot of its local schedule. The term $J_l^{(u)}$ represents the need for (at least) an additional slot for time-slot reference alignment on every link a node acts as slave. The lower bound on the minimum period is not tight but can be used to identify instances where the entire set of feasible allocations can be captured by a set of local conditions. This is summarized by the following proposition:

Proposition 4.4.1 *Consider an asynchronous TDMA ad hoc network $G(N, E)$. If for every demand slot allocation $\boldsymbol{\tau}$, $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$, then,* all *feasible allocations for $G(N, E)$ can be captured by the following set of local conditions:*

$$\sum_{l \in L(u)} \tau_l \leq T_{system} - \sum_{l \in L(u)} J_l^{(u)}, \ \forall u \in N \tag{4.5}$$

Proof We use contradiction. Let $\boldsymbol{\tau}^*$ be a demand allocation satisfying the local conditions of eq. (4.5) but is not feasible. Since $\boldsymbol{\tau}^*$ is not feasible, the minimum period for realizing it must be strictly greater than T_{system} slots: $L_{min}(\boldsymbol{\tau}^*) > T_{system}$. The demand allocation $\boldsymbol{\tau}^*$ obeys the local conditions of eq. (4.5):

$$\sum_{l \in L(u)} (\tau_l^* + J_l^{(u)}) \leq T_{system}, \ \forall u \in N \Rightarrow$$

$$\max_{u \in N} \sum_{l \in L(u)} (\tau_l^* + J_l^{(u)}) \leq T_{system} \Rightarrow$$

$$LB(\boldsymbol{\tau}^*) \leq T_{system}$$

Since $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau}), \ \forall \boldsymbol{\tau} \in G(N, E)$, we reach the conclusion that $L_{min}(\boldsymbol{\tau}^*) \leq T_{system}$, i.e. $\boldsymbol{\tau}^*$ is feasible. This contradicts our initial hypothesis. ■

Proposition 4.4.1 states that classes of topologies or specific topologies for which $L_{min}(\boldsymbol{\tau}) = LB(\boldsymbol{\tau})$ for all $\boldsymbol{\tau}$, can be fully utilized by distributed algorithms. In the next section we show the class of tree topologies satisfies this property.

4.4.2 Optimal link scheduling for tree networks

Let the ad hoc network topology $G(N, E)$ be a tree. Without loss of generality, assume that an arbitrary node in the tree is designated as root. The root provides a reference for parent-child relationships between the node endpoints of every link in the network. The parent-child relationship between the node endpoints is independent of their master-slave relationship. We define the *level of a node* to be its hop distance from the root

(the root has a level equal to zero). The *level of a link* equals the level of its child node endpoint.

Let $\boldsymbol{\tau}$ be a demand allocation. Given $\boldsymbol{\tau}$, every node is equipped with a local schedule of period $T_{system} = LB(\boldsymbol{\tau})$ slots (eq. (4.3)). The slot positions in each local schedule are indexed from 0 to $T_{system} - 1$. A set of consecutively assigned slots to link l in the local periodic schedule $\boldsymbol{S}_u$ of node u, forms a (circular) window $W_l^{(u)} = [s_l^{(u)}, e_l^{(u)}]$:

$$[s_l^{(u)}, e_l^{(u)}] = \begin{cases} s_l^{(u)}, ..., e_l^{(u)} & \text{if } s_l^{(u)} \leq e_l^{(u)} \\ s_l^{(u)}, .., 0, ..., e_l^{(u)} & \text{otherwise} \end{cases} \tag{4.6}$$

where $s_l^{(u)}$ and $e_l^{(u)}$ are the start and end slot positions assigned to link l in $\boldsymbol{S}_u$, respectively. The number of slots in $W_l^{(u)}$ is denoted as $|W_l^{(u)}|$. Modulo-T_{system} addition and subtraction are denoted by "$\oplus$" and "$\ominus$", respectively.

We now describe the operation of CENTRAL_TREE, a link scheduling algorithm that realizes $\boldsymbol{\tau}$ using a period of $LB(\boldsymbol{\tau})$ slots. Initially, all local schedules are empty. Links are scheduled in a breadth-first manner. In the first iteration, the root node r starts from slot 0 in its local schedule $\boldsymbol{S}_r$ and schedules its children links (level-1 links) until their total demand is satisfied. The links are scheduled non-preemptively in successive windows: For each child link $l = (r, c)$, r allocates in $\boldsymbol{S}_r$ a window of $\tau_l + J_l^{(r)}$ consecutive slots, immediately succeeding the window of the previously scheduled child link. After l has been scheduled in $\boldsymbol{S}_r$, the child node c assigns $\tau_l + J_l^{(c)}$ time-overlapping slots to link l in its own local schedule $\boldsymbol{S}_c$.

In the next iteration, the root children (level-1 nodes) schedule their own children links (level-2 links). For each such node u, its parent link l_p has already been satisfied by a window $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ in $\boldsymbol{S}_u$, during the previous iteration. Node u starts from slot position $e_{l_p}^{(u)} \oplus 1$ and schedules its children links non-preemptively in successive windows by filling $\boldsymbol{S}_u$ towards slot position $s_{l_p}^{(u)}$ in a circular fashion. For each link

$l = (r, c)$ scheduled in $\boldsymbol{S}_u$, the child node endpoint c assigns time-overlapping slots in $\boldsymbol{S}_c$.

The scheduling process is repeated recursively until the highest-level links have been scheduled and the leaf nodes have updated their local schedules.

Theorem 4.4.2 *If the network topology is a tree, any demand allocation $\boldsymbol{\tau}$ can be realized by algorithm CENTRAL_TREE using a period of $LB(\boldsymbol{\tau})$ slots.*

Proof Since T_{system} is set to $LB(\boldsymbol{\tau})$ slots, it suffices to show that no node runs out of slots in its local schedule during the algorithm execution. We use induction on the link levels over the tree.

Level 1: Starting at slot 0 in its local schedule $\boldsymbol{S}_r$, the root r schedules $\sum_{l \in L(r)} \tau_l + J_l^{(r)}$ slots which, by definition, is less than or equal to $LB(\boldsymbol{\tau})$. In addition, each child node c on child link l allocates $\tau_l + J_l^{(c)}$ time-overlapping slots in its local schedule, which does not exceed $LB(\boldsymbol{\tau})$.

Level k-1: Assume that no node has run out of slots after all level $k-1$ links have been scheduled. Due to the breadth-first recursion, each of the level $k-1$ nodes that are children of the same (level $k-2$) parent has been assigned a single window, the windows of such nodes being mutually exclusive.

Therefore, without loss of generality, we can consider a link l_p of level $k-1$ and its child node u in isolation. During the previous iteration u has been allocated $\tau_{l_p} + J_{l_p}^{(u)}$ consecutive slots in $\boldsymbol{S}_u$.

Let $CH(u)$ be the set of children links of node u. Node u will need $\sum_{l \in CH(u)} (\tau_l + J_l^{(u)})$ slots in $\boldsymbol{S}_u$ to schedule its children links in mutually exclusive windows. Thus node u will have assigned a total of $\sum_{l \in CH(u)} (\tau_l + J_l^{(u)}) + \tau_{l_p} + J_{l_p}^{(u)} = \sum_{l \in L(u)} (\tau_l + J_l^{(u)})$ slots at the end of iteration k, which, by definition does not exceed $LB(\boldsymbol{\tau})$. Also, for each link $l = (u, c) \in CH(u)$, the child node c will allocate $\tau_l + J_l^{(c)}$ in $\boldsymbol{S}_c$, which does

not exceed $LB(\boldsymbol{\tau})$. Therefore no node runs out of slots at the end of iteration k. The induction step is complete. ■

Algorithm CENTRAL_TREE cannot be used in practice because it requires global information and a priori knowledge of a static demand allocation for which it computes an optimal schedule. Its importance lies in establishing that the entire set of feasible allocations for tree topologies can be captured by local conditions, even for the case of asynchronous TDMA.

4.4.3 Some practical considerations

To summarize our results on local feasibility conditions, guaranteeing feasibility of the global link demand allocation requires each node u to provide at most T_u^R slots as demands on its adjacent links:

$$\sum_{l \in L(u)} \tau_l \leq T_u^R \tag{4.7}$$

where the maximum value for T_u^R is given by the table in Figure 4.4. According to our previous discussion T_u^R depends on whether or not the network is slot-synchronized and whether a distributed protocol for enforcing bipartite or tree topologies exists.

	Topology		
	Arbitrary	Bipartite	Tree
Synchronized	$\lfloor \frac{2}{3} T_{system} \rfloor$	T_{system}	T_{system}
Asynchronous	$\lfloor \frac{\lfloor \frac{2}{3} T_{system} \rfloor}{2} \rfloor$	$\lfloor \frac{T_{system}}{2} \rfloor$	$T_{system} - \sum_{l \in L(u)} J_l^{(u)}$

Figure 4.4: Maximum known values (in # of slots) for the QoS utilization parameter T_u^R ensuring feasibility under various assumptions on topology control and slot synchronization.

In a multi-channel system such as the one considered here, bipartite topologies can be easily enforced using local information: every node acts only as master or slave to all its adjacent links and the channel assigned to each link is derived from the (unique) address of the master node endpoint. In practice, bipartite topologies arise in clustered architectures [35][36]. In these architectures each cluster is defined and controlled by a clusterhead node. Inter-cluster communication is performed by non-clusterhead gateway nodes that participate in multiple clusters.

Tree topologies can be enforced using existing algorithms for dynamic tree formation and maintenance [29][31][77]. Trees manifest in various ad hoc networking applications. Existing topology construction algorithms for Bluetooth ad hoc networks generate tree topologies [25] [29][31][101]. According to the sensor network communication paradigm, sensors report data back to a single source over a tree structure [102][103]. Tree topologies are also used for energy-efficient broadcasting [21][104]. Several non-tree ad hoc networks use a certain subset of nodes as a tree backbone for facilitating administrative purposes such as routing.

When node u joins the network it queries its neighbors about existence of global synchronization or topology control protocol and sets its T_u^R according to the table in Figure 4.4. The table demonstrates that more restricted topologies allow for greater node utilization. Trees allow for maximum utilization in both synchronized and asynchronous systems; in addition, the entire set of feasible allocations is captured only by local conditions.

We would like to emphasize that the terms ”feasibility” and ”underutilization” are with respect to provision of QoS guarantees, i.e. when nodes aim to realize a link demand allocation in the network. The QoS utilization parameter indicates a sufficient number of slots each node can provide for QoS traffic. However, the remaining slots in

the nodes' local schedules need not be idle–they are always available for other purposes including control or best-effort traffic.

4.5 Summary

We introduced a distributed TDMA protocol for multi-channel ad hoc networks that operates with no global assumptions such as network-wide slot synchronization, knowledge of number of nodes in the network or universal slot enumeration. The protocol reacts locally to topology or traffic changes, adjusting link rates by means of conflict-free slot reassignments.

The TDMA nature of the protocol allows provision of QoS guarantees–a set of link rates realized by a TDMA schedule. We identified local feasibility conditions that depend on the topology control algorithm used in the network and the existence (or lack thereof) of global slot synchronization. In general these conditions capture a subset of feasible allocations but ensure that a distributed algorithm will be able to reach a schedule that realizes them. The definition as well as distributed algorithms for enforcement of QoS objectives within this subset of feasible allocations is the subject of the following chapters.

Chapter 5

Link-level max-min fairness in wireless ad hoc networks

In this chapter we focus on distributed bandwidth allocation mechanisms that operate in the subset of feasible allocations defined in chapter 4 and aim at generating and enforcing link demands for the realization of various Quality of Service (QoS) objectives.

One possible QoS objective is for users to impose specific slot demands on the network links. This would require additional mechanisms for admission control and resource provisioning that might be costly in the mobile ad hoc network setting. Another QoS model might be for users to specify a utility that expresses their satisfaction level as a function of the bandwidth they receive. Given the user utility functions the network tries to allocate bandwidth accordingly. Since it is not always easy to characterize a user satisfaction in terms of bandwidth, defining utility functions that are meaningful is generally difficult. Max-min fairness is an intuitive and desirable objective in application scenarios where no explicit knowledge exists about the bandwidth requirements of the users in the network. A max-min fair allocation tries to allocate an equal amount of bandwidth to all users. If a user cannot utilize all the bandwidth because of a constraint, then the residual bandwidth is distributed to less constrained users. Among any feasi-

ble bandwidth allocations, a max-min fair allocation ensures that the most constrained users are allotted the maximum possible bandwidth. This form of fairness has also been shown to be a good trade-off between maximizing network utilization and providing fair access to the network users.

In this chapter, our focus is at the medium access layer; as in [105][106][107] [108], we address max-min fairness for the case where the ”users” are single-hop flows (links) instead of multi-hop sessions. Two reasons motivate this approach. First, maintenance of state for end-to-end sessions may not be possible in lightweight mobile nodes nor even desirable in a highly mobile network. Still, transmissions must be coordinated such that robust and balanced access is provided to the higher layers. Second, provision of fairness on a multi-hop session basis can be viewed as an orthogonal objective. Recently, two distributed algorithms have been proposed in [109] and [61] for end-to-end utility-based fairness and max-min fairness, respectively. Operating at a higher layer, these algorithms compute the fair session rates, but they do not enforce these rates–a distributed medium access mechanism is needed.

We first introduce a fluid model that captures only the bandwidth allocation constraints without taking into account the conflict-free requirement. In this model we propose a distributed algorithm that starts from an initial rate allocation and eventually converges to the max-min fair solution after a series of asynchronous link rate adjustments. The slotted version of the algorithm uses the distributed TDMA protocol and attempts to emulate the one of the fluid model with the basic difference that whenever it adjusts the rate of a link it does so by re-assigning transmission slots directly on the network schedule without violating the conflict constraints. Since the fluid algorithm converges to the max-min fair rates under asynchronous distributed operation, the slotted one is expected to have similar properties.

Max-min fairness in slotted multi-channel wireless systems was first addressed in [108]. The authors provide an on-line scheduling policy and prove analytically that it converges to the max-min fair solution. However, the policy uses global network information to compute the conflict-free link schedule and, therefore, cannot be implemented in practice. The slotted version of the distributed algorithm proposed here is implementable but no analytical proof exists for its exact convergence as in the fluid case. Through extensive simulations in static and dynamic networks we show that the algorithm possesses very good tracking properties of the max-min fair rate allocation.

The rest of the chapter is organized as follows: Section 5.1 presents the network model and definition of max-min fairness. Section 5.2 introduces the fluid part of the asynchronous algorithm that computes the amount of rate adjustments. Section 5.3 describes the scheduling technique that enforces these rate adjustments by means of conflict-free slot reallocations. The algorithm performance is evaluated in Section 5.4. A traffic-adaptive extension of the basic algorithm is presented in section 5.4.4. We discuss related work in Section 6.4. Section 5.6 concludes.

5.1 Network Communication Model

We represent the multi-channel ad hoc network as a graph $G(N, L)$ where vertices correspond to the wireless nodes and edges correspond to established communication links. Since only primary interference exists, any set of links that do not have a common node endpoint can transmit simultaneously without conflict. In other words a link activation set is a matching in $G(N, L)$ (see Fig. 5.1).

We use two models to represent bandwidth allocation. In the *slot model* the bandwidth allocated to a link l is expressed as the number of slots τ_l in a TDMA schedule of

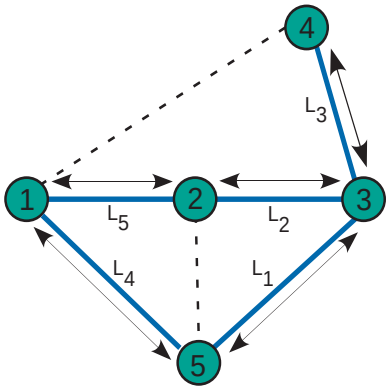


Figure 5.1: Solid lines denote established links, while dotted lines denote wireless proximity. Links have been assigned communication channels such that secondary interference is tolerated and support bidirectional transfer per slot. Since links L_3 and L_5 use different channels, they can transmit simultaneously without conflict even if nodes 1 and 4 are within range. Still, every node can communicate to only a single link at a time due to the single transceiver constraint. Thus, only sets of links not having common node endpoints can transmit simultaneously without conflict (e.g. $\{L_3, L_4\}$ or $\{L_1, L_5\}$).

period T_{system} slots. The *fluid model* does not refer to a slotted system. The bandwidth allocated to a link l is expressed as a normalized rate r_l and is the time fraction the node endpoints spend communicating conflict-free on this link.

Given link slot allocation $\boldsymbol{\tau} = (\tau_1, ..., \tau_{|L|})$ in the slot model, the corresponding normalized rate allocation in the fluid model equals $\boldsymbol{r} = \boldsymbol{\tau}/T_{system}$. Conversely, the slot allocation in a T_{system}-periodic system corresponding to rate allocation $\boldsymbol{r} = (r_1, ..., r_{|L|})$ is $\boldsymbol{\tau} = \lfloor \boldsymbol{r} \cdot T_{system} \rfloor$.

The two models serve different purposes: the fluid model is more general and intuitive and can be used to describe bandwidth sharing as well as notions such as feasibility and max-min fairness. On the other hand, a real system uses the slotted model–it always works in the discrete domain using a TDMA schedule of period T_{system} slots.

5.1.1 Rate feasibility and max-min fairness

Under the fluid model, the *effective capacity* C_u of a node u is defined as the maximum rate it provides to its adjacent links $L(u)$ for communication. If C_u is less than unity then the node is partially utilized and remains idle for the rest of the time.

A link rate allocation $\boldsymbol{r} = (r_1, ..., r_l, ..., r_{|L|})$ is *feasible* if there exists a conflict-free (not necessarily periodic) TDMA schedule that allocates to every link l a long term rate equal to r_l. Since each node u cannot communicate on different adjacent links simultaneously the sum of the rates of all links in $L(u)$ must be less than C_u. A node effective capacity of unity guarantees feasible rate allocations when the network topology is bipartite [83]. For arbitrary topologies, the feasibility region cannot be characterized only by these local conditions. The additional non-local conditions require that, for every odd node subset Q ($|Q| > 1$) in the topology graph, the sum of the rates of all links adjacent to the nodes in Q must not exceed $(|Q| - 1)/2$. Alternatively, an effective node capac-

ity of $2/3$ provides with a sufficient (albeit not necessary) characterization of feasibility [83]. Hence, we will use the following local capacity constraints for the fluid model:

$$\sum_{l \in L(u)} r_l \leq C_u \ , \ \ \forall u \in N \ , where C_u = \begin{cases} 1 & \text{if } G(N,L) \ is \ bipartite \\ 2/3 & \text{otherwise} \end{cases}$$

If a link l has a long-term arrival rate B_l we also need a *demand constraint* on its maximum allowable rate:

$$r_l \leq B_l \tag{5.1}$$

Here, we implicitly assume the the B_l are such that feasibility is maintained: for every link l, $B_l \leq 1$ and for every node u, $\sum_{l \in L(u)} B_l \leq C_u$. These conditions hold because, in practice, the B_l are estimated online with respect to the node effective capacity. We will outline such an estimation procedure in section 5.4.4.

A feasible rate allocation is *max-min fair (MMF)* if the rate allocated to a link cannot be increased without decreasing the rates of other contending links having equal or less rate. More formally, we define a rate allocation $\boldsymbol{r}$ to be MMF if:

1. It is feasible i.e. satisfies the capacity and demand constraints given by equations (5.1) and (5.1).

2. It is lexicographically greater than any other feasible rate allocation vector $\boldsymbol{r}'$. In other words, if we sort both $\boldsymbol{r}$ and $\boldsymbol{r}'$ in increasing order of their rates and start comparing the rates of the respective permuted vectors $\widetilde{\boldsymbol{r}}$ and $\widetilde{\boldsymbol{r}}'$ starting from the lowest index, then after a possible set of equal rates there will be an index l such that $\widetilde{r}_l' < \widetilde{r}_l$.

Intuitively, if all links have equal access right, the most constrained links are provided the maximum possible bandwidth.

Node u is defined as a *bottleneck node* to an adjacent link $l \in L(u)$ if it is fully utilized with respect to C_u and the rate of link l is greater than or equal to the rate of all other links in $L(u)$. The definition of bottleneck node gives rise to a distributed criterion to determine whether a given allocation is MMF or not:

Theorem 5.1.1 MMF criterion: *A bandwidth allocation is MMF if and only if every link l in the network satisfies at least one of the following conditions:*

- *The bandwidth allocated to link l equals its long-term arrival rate B_l.*
- *The link l has at least one bottleneck node.*

A similar criterion has been used for determining MMF allocations of end-to-end sessions sharing wireline networks; the proof of theorem 5.1.1 can be derived from the proof of the wireline criterion [110].

The link MMF rates can be computed using an iterative, off-line centralized algorithm. During each iteration, each node equally divides its available bandwidth to its adjacent links. The bottlenecks are the nodes for which this division is minimum; the minimum ratio is the MMF rate for this iteration and is allocated to the links adjacent to the bottleneck nodes. We then remove the bottleneck nodes and their adjacent links from the network and reduce the available bandwidth of the remaining nodes by the amount consumed by the removed links. Any node whose available bandwidth becomes zero is also removed. In the next iteration, we consider the reduced network, determine the (next-level) bottleneck nodes and repeat the procedure. The process continues until all links have been allocated their rates. Upon termination, this algorithm yields the link MMF rates because the links removed in each iteration have at least one bottleneck node. The centralized algorithm is similar in spirit to the algorithm of Bertsekas and Gallager [110] that computes MMF rates for end-to-end sessions sharing the links of a

wireline network–in our case, the shared resources are the nodes rather than wired links and the entities sharing resource bandwidth are the wireless links rather than end-to-end sessions.

Figure 5.2 is an example of the centralized algorithm performing the MMF rate computation. The topology is bipartite and all links are assumed backlogged ($B_l = 1$). The algorithm pseudocode in Figure 5.14 in Chapter Appendix 5.A includes the case where demand constraints are taken into account.

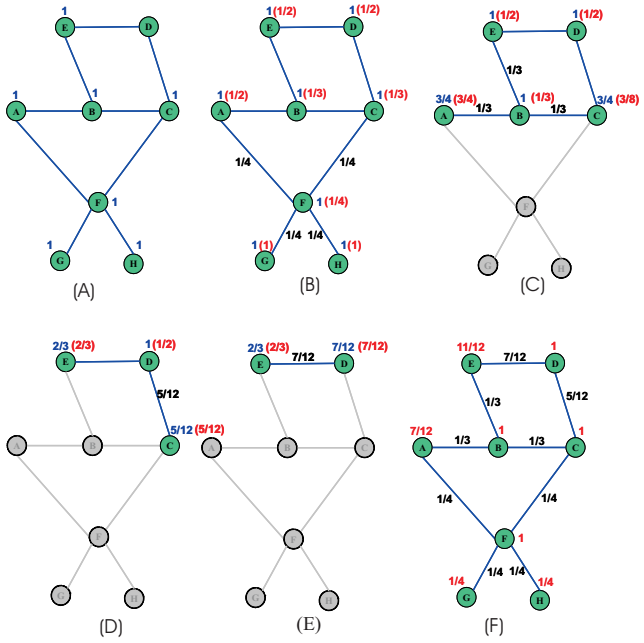


Figure 5.2: (a) Initialization: All nodes set their effective capacities to 1 (bipartite topology). (b) Iteration 1: Bottleneck node is F–over all nodes, it provides the minimum fair share of 1/4 to its adjacent links. (c) Iteration 2: Bottleneck node is B (MMF rate is 1/3). (d) Iteration 3: Bottleneck node is C (MMF rate is 5/12). (e) Iteration 4: Bottleneck node is D, MMF rate is 7/12. (f) The MMF link rate allocation and corresponding node utilizations.

The centralized algorithm demonstrates that max-min fairness for wireless links is a

global objective–the optimal allocation is dependent on the entire topology. Since nodes have access to only local information, they never know the MMF rates of their adjacent links. We seek an asynchronous distributed algorithm where nodes incrementally reach the global MMF link rate allocation through local rate adjustments. Such an algorithm would allow convergence to the MMF solution provided the topology remains stable for a sufficient amount of time. A second challenge (not addressed even by the centralized algorithm) is for the nodes to reach a TDMA schedule that enforces these rates.

We first introduce an algorithm that computes the MMF rates using only local information. This algorithm is then used in the slotted system to guide slot re-assignments for rate adjustments. We thus aim for rate computation and enforcement to occur in parallel. Our approach will be presented in detail in the following sections.

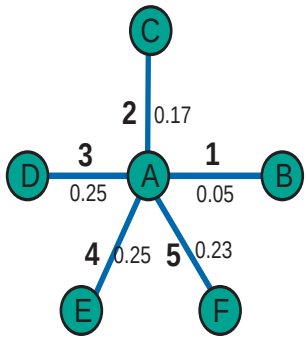
5.2 Distributed algorithm–Fluid model

In this section we introduce an asynchronous distributed algorithm for the fluid model that works in the feasible rates region and eventually converges to the MMF allocation.

5.2.1 Fairness deficit

A central component of the distributed algorithm is the *fairness deficit computation (FDC)*, performed by a node u with respect to an adjacent link $l = (u, v)$: node u starts from the current allocation $\boldsymbol{r}_u = \{r_l : l \in L(u)\}$ on its adjacent links and computes a new allocation $\boldsymbol{r}'_u$ where it is a bottleneck for l. Then, the *fairness deficit of node u for link l* is defined as $fd_l^{(u)} = r'_l - r_l$.

The FDC can be implemented by the following iterative algorithm: Initially, $\boldsymbol{r}'_i = \boldsymbol{r}_i$. The rate of link l is increased by the excess capacity $E_u = C_u - \sum_{k \in L(u)} r_k$ of node u.



r_A / step	r_1	r_2	r_3	r_4	r_5	E_1	max_rate
0	0.05	0.17	0.25	0.25	0.23	0.05	0.25
1	0.10	0.17	0.25	0.25	0.23	0.00	0.25
2	0.20	0.17	0.20	0.20	0.23	0.00	0.23
3	0.215	0.17	0.20	0.20	0.215	0.00	0.215

Figure 5.3: The FDC algorithm for link 1 at node A ($C_A = 1.0, B_1 = 1.0$). The shaded entries during each iteration t denote $M^{(t)}$. The last row is $\boldsymbol{r}'_A$; the fairness deficit is $fd_1^{(A)} = 0.215 - 0.05 = 0.165$.

Then, at each iteration t, we consider the set $M^{(t)}$ of maximum rates in $\boldsymbol{r}'_u$. If r'_l is not in $M^{(t)}$, the total bandwidth of $M^{(t)}$ plus r'_l is equally distributed to the links in $M^{(t)}$ and link l. This operation decreases the rates of links in $M^{(t)}$ and increases the rate of link l; it also determines the maximum rate set of the next iteration. The process is repeated until r'_l is in the maximum rate set.

The above description assumes that l is a greedy link (demand constraint $B_l = 1$). If $B_l < 1$ the iterations stop when either r'_l is in the maximum rate set or when r'_l becomes greater than or equal to B_l. In this case, the excess bandwidth $r'_l - B_l$ is equally distributed to the links in the maximum rate set of the last iteration and r'_l is set to B_l. The FDC steps are described by the pseudocode in Figure 5.15; figure 5.3 is a representative example of the FDC operation.

5.2.2 Fluid distributed algorithm

The distributed fluid algorithm starts from an arbitrary feasible link rate allocation. Links are continuously activated for rate adjustment at asynchronous time instants. When a link $l = (u, v)$ is activated for rate adjustment, the algorithm seeks to increase its rate such that one of the node endpoints becomes a bottleneck for this link. More specifically, the following actions take place:

1. Nodes u and v perform the FDC for link l and exchange their fairness deficits. The *link fairness deficit* is $fd_l = min\{fd_l^{(u)}, fd_l^{(v)}\}$.

2. If the link fairness deficit is zero, then no rate adjustment takes place, steps 3 and 4 are not executed and no further action is taken.

3. If both deficits are non-zero, then the rate of link l is increased by fd_l.

4. Nodes u and v adjust the rates of the rest of their adjacent links accordingly. The new link rate allocation $\boldsymbol{r}'_u$ for the minimum deficit node u has already been computed by the FDC in step 1. For the maximum deficit node v, any new link rate allocation $\boldsymbol{r}'_v$ where the sum of rates does not exceed C_v and link l has rate equal to $r'_l = r_l + fd_l$, is acceptable. For example, such an allocation can be reached if v applies again the FDC on link l with an upper bound equal to $min\{B_l, r_l + fd_l\}$.

Note that in order to perform the above adjustments we only need to reduce the rates of certain links adjacent to nodes u and v except link l, the rate of which is increased by fd_l.

Theorem 5.2.1 ((Convergence Theorem)) *Given a static topology and an arbitrary initial feasible link rate allocation, the distributed fluid algorithm converges to the net-*

work MMF allocation after a finite number of link activations for rate adjustment.

Proof We assume that every link in the network will be asynchronously activated for rate adjustment infinitely often. In other words, links do not stop attempting to perform rate adjustments and intervals in-between consecutive rate adjustments of a specific link are finite. For simplicity, we assume that all links are backlogged (i.e. $B_l = 1 \forall l \in G(N, L)$). The proof under demand constraints ($B_l < 1$) follows a similar reasoning.

Let the link rate adjustment process start at time t_0. Consider the set of most constrained nodes $N^{(0)}$, for which the ratio $C_k/|L(k)|$ is equal and minimum:

$$N^{(0)} = \{u : u = arg \min_{w \in N} \{C_w/|L(w)|\}\}.$$

When a link l adjacent to a node u in $N^{(0)}$ is activated for rate adjustment:

- Node u is always the bottleneck node for l because it offers the minimum deficit.
- According to the FDC algorithm of u, link l will belong to the maximum rate set of the new rate allocation $\boldsymbol{r}'_u$. Also, the cardinality of the new maximum rate set of node u increases by one link.

When all adjacent links of u have been activated for rate adjustment, its maximum rate set will have $|L(u)|$ links, each link allocated rate $C_u/|L(u)|$. From that point on, when a link $l \in L(u)$ is activated for rate adjustment, u will be giving it a fairness deficit of zero, and no further rate adjustment will take place for such a link. Since links are activated infinitely often for rate adjustment, there will be a point $t_1 > t_0$ where all adjacent links to all nodes u in $N^{(0)}$ have been allocated a rate of $C_u/|L(u)|$.

Let $L^{(0)}$ be the set of all links adjacent to the nodes in $N^{(0)}$ and consider the algorithm operation after time t_1.

Nodes in $N^{(0)}$ will never adjust the rates of their adjacent links. When a node u in $N - N^{(0)}$ executes the FDC algorithm for an adjacent link l not in $L^{(0)}$, it may decrease the rates of other adjacent links except those in $L^{(0)}$–these links have the global minimum rate in the network and will never belong to the maximum rate set during the FDC computation of u. This is equivalent to saying that links in $L^{(0)}$ and the bandwidth they consume have been "removed" from the network; the nodes in $N-N^{(0)}$ redistribute their remaining capacity to their adjacent, non-saturated links.

After time t_1, denote by $N^{(1)}$ the set of the next most constrained nodes in the network:

$$N^{(1)} = \{u : u = arg \min_{w \in N - N^{(0)}} \{(C_w - \sum_{k \in (L(w) \cap L^{(0)})} r_k)/|L(w)|\}\}.$$

When a link $l = (u, v)$ adjacent to a node $u \in N^{(1)}$ is activated for a rate adjustment:

- If the other endpoint node v is in $N^{(0)}$, no rate reallocation takes place because the link fairness deficit is zero.
- Otherwise, node u is the bottleneck node for this link. Now if there is another link in $L(u)$ for which the endpoint node $w \neq v$ is in $N^{(0)}$, then its rate cannot be decreased further by the FDC algorithm of u because it has already established the minimum possible fair share in the network ($C_w/|L(w)|$).
- The cardinality of the new maximum rate set of node u increases by one link.

Now let $t_2 > t_1$ be the time instant where all adjacent links to all nodes u in $N^{(1)}$ (except the links $k \in L^{(0)}$) will have been allocated their fair rates ($(C_u - \sum_{k \in (L(u) \cap L^{(0)})} r_k)/|L(u)|$). It is straightforward to show by induction that there exists a future finite time instant t_{n+1} until every set of constrained nodes

$$N^{(n)} = \{u : u = arg \min_{w \in N - N^{(0)} \cup \ldots \cup N^{(n)}} \{(C_w - \sum_{k \in L(w) \cap (L^{(0)} \cup \ldots \cup L^{(n)})} r_k)/|L(w)|\}\}$$

will saturate its remaining links. It follows that the algorithm converges to the MMF allocation in a finite number of steps. ■

The algorithm is self-terminating–no explicit message needs to be sent to the entire network to signal convergence. When a link is activated for a possible rate adjustment, adjustment occurs only if the link fairness deficit is non-zero. Upon convergence, all links will have at least one bottleneck node–the link fairness deficit will be zero for all links in the network.

5.3 Distributed algorithm–Slot model

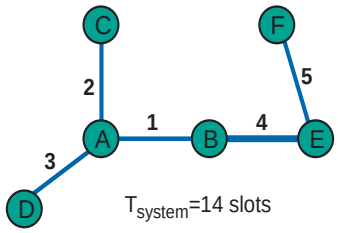
The fluid algorithm guarantees convergence to the MMF rates but does not yield a conflict-free schedule that realizes these rates. This is because the fluid model does not refer to a slotted system but is mainly concerned with how to redistribute the bandwidth.

The slotted algorithm emulates the fluid algorithm: it adjusts the rate of a link by re-assigning transmission slots directly on the network schedule without violating the conflict constraints. Since the fluid algorithm converges to the MMF rates under asynchronous distributed operation, the slotted algorithm will have similar properties, provided it yields a conflict-free schedule after each rate adjustment.

Next, we describe the three components needed to use the fluid algorithm in the slotted system: 1) a modification in the local conditions 2) the slotted FDC and 3) the slot assignment algorithm.

5.3.1 Local conditions

When the fluid algorithm is applied to the slotted system, rates will be quantized to slots. In order for the resulting slot demands to be feasible, we need to restrict the fluid model



slot#	0	1	2	3	4	5	6	7	8	9	10	11	12	13
S_A	3	2	2	3	2	3	2	3	1	2	1	3	2	3
S_B	-	4	4	4	4	4	4	4	1	4	1	-	-	-
S_C	-	2	2	-	2	-	2	-	-	2	-	-	2	-
S_D	3	-	-	3	-	3	-	3	-	-	-	3	-	3
S_E	5	4	4	4	4	4	4	4	5	4	-	5	5	5
S_F	5	-	-	-	-	-	-	-	5	-	-	5	5	5

Figure 5.4: A wireless ad hoc network using a conflict-free TDMA link schedule (system period $T_{system}$is 14 slots). Each slot in a local schedule $\boldsymbol{S}_u$ indicates the link assigned to this slot by node u.

local conditions to fit the corresponding slotted system:

$$\sum_{l \in L(u)} r_l \leq C_u, \;\; C_u = \frac{T_u^R}{T_{system}}, \;\; \forall u \in N \tag{5.2}$$

where T_u^R depends on the topology control algorithm and the existence of network-wide slot synchronization and its maximum allowed value is given by the Table in Figure 4.4 of Chapter 4.

The capacity conditions (eq. (5.2)) allow the fluid algorithm to operate over both asynchronous and synchronized TDMA systems. For ease of illustration through the rest of the chapter we will describe the mapping of the fluid algorithm to a slot-synchronized ad hoc network.

5.3.2 Slotted FDC

The number of conflict-free slots each node u transmits on its adjacent links in its local schedule $\boldsymbol{S}_u$ determines its slot allocation $\boldsymbol{\tau}_u$. In the slotted FDC node u uses the fluid

FDC to reach from the initial slot allocation $\boldsymbol{\tau}_u$ to a new slot allocation $\boldsymbol{\tau}'_u$; the slotted FDC outputs the difference vector $\boldsymbol{x}_u = \boldsymbol{\tau}'_u - \boldsymbol{\tau}_u$. An example of the slotted FDC operation is shown in Figure 5.5.

step		Link 1	Link 2	Link 3	rem	Actions
0	$\boldsymbol{\tau}_A$	2	6	6	0	$T_{system} = 14$
1	$\boldsymbol{r}_A$	2/14	6/14	6/14	0/14	$\boldsymbol{r}_A = \boldsymbol{\tau}_A / T_{system}$
2	$\boldsymbol{r}'_A$	0.333	0.333	0.333	0.000	Fluid FDC
3	$\boldsymbol{\tau}'_A$	4	4	4	2	$\boldsymbol{\tau}_A = \lfloor \boldsymbol{r}_A \cdot T_{system} \rfloor$
4	$\boldsymbol{\tau}'_A$	6	4	4	0	Give remainder slots to link 1
5	$\boldsymbol{x}_A$	+4	-2	-2	0	$\boldsymbol{x}_A = \boldsymbol{\tau}'_A - \boldsymbol{\tau}_A$

Figure 5.5: The slotted FDC for node A on link 1 in the network of Fig. 5.4: 1) slots are converted to rates. 2) fluid FDC is applied to rates. 3) Resulting rates are quantized to slots. 4) Excess slots due to the quantization of step 3 are given to link 1. (5) Difference vector $\boldsymbol{x}_A$–the discrete fairness deficit for link 1 is 4 slots.

5.3.3 Slot assignment algorithm

Given $\boldsymbol{x}_u$, a positive (or negative) element x_k indicates the rate of link k must be increased (or decreased) by x_k slots. A zero element indicates no change in the rate of the corresponding link. The set of surplus links (i.e. the links affected by the rate adjustment on link l) is $X_u^- = \{k : x_k < 0\}$. Also x_l is positive and equal to the fairness deficit amount of slots that must be assigned to link l.

The slot assignment algorithm decides for each surplus link k *which* x_k out of the τ_k current slot positions will be re-assigned to link l. The slot assignment algorithm consists of two phases. In **Phase I**, node u takes into account the local link schedule

$\boldsymbol{S}_v$ of the other node endpoint v and assigns slot positions to link l in the following prioritized manner:

1. First, link l is assigned slot positions that are currently assigned idle in both local schedules $\boldsymbol{S}_u$ and $\boldsymbol{S}_v$, if such positions exist.

2. If step 1 did not find enough matching slot positions, link l is assigned slot positions that are currently assigned to surplus link k in $\boldsymbol{S}_u$ and idle in $\boldsymbol{S}_v$, if such positions exist.

The number of slot positions that matched during Phase I may still be less than the required deficit for link l. For each surplus link k that Phase I selected only m_k out of x_k slots, **Phase II** randomly selects extra $x_k - m_k$ slot positions that are still assigned to k in $\boldsymbol{S}_u$ and reassigns them to link l. The algorithm outputs a list indicating the extra slot positions that should be assigned to link l.

Figure 5.16 in Chapter Appendix 5.A contains the pseudocode of the slot assignment algorithm. As an example, after the FDC of Figure 5.5, node A is called to decide on the extra slot positions that will be assigned to link 1 based on its own and node B's local schedules (see Figure 5.6). The rate difference vector (row 5 in Figure 5.5) indicates that links 2 and 3 must give away two slots each and link 1 should be assigned four extra slots. By matching the idle slots of $\boldsymbol{S}_B$, node A reassigns slot positions $\{7, 12\}$ from 2 and $\{11, 13\}$ (selected randomly from $\{0, 11, 13\}$) from 3 to link 1.

5.3.4 Slotted distributed algorithm

The ad hoc network operates according to a TDMA schedule of period T_{system} slots. At asynchronous time instants, nodes use the TDMA protocol of Chapter 4 to adjust the rates of their adjacent links through local slot reassignments. More specifically, when a

Slot	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
$\boldsymbol{S}_A$	3	2	2	3	2	3	2	2	1	2	1	2	2	3
$\boldsymbol{S}_B$	-	4	4	4	4	4	4	-	1	4	1	-	-	-

Figure 5.6: The matching slot positions in local schedules $\boldsymbol{S}_A$ and $\boldsymbol{S}_B$ are $\{0, 7, 11, 12, 13\}$: In $\boldsymbol{S}_A$ they are assigned to surplus links 2 and 3, while in $\boldsymbol{S}_B$ they are assigned idle. Taking this information into account, node A eventually selects slot positions $\{7, 11, 12, 13\}$ for link 1.

link $l = (u, v)$ is activated for rate adjustment, the following actions take place:

1. Endpoint nodes u and v perform the slotted FDC for link l and exchange their discrete fairness deficits $fd_l^{(u)}$ and $fd_l^{(v)}$. The link fairness deficit is $fd_l = min\{fd_l^{(u)}, fd_l^{(v)}\}$.

2. If the link fairness deficit is zero, then no rate adjustment takes place, steps 3 and 4 are not executed and no further action is taken.

3. If both deficits are non-zero, link l must be assigned fd_l additional slots in concurrent positions of both endpoint local schedules.

4. The minimum deficit node endpoint operates as ASSIGNER and executes the slot assignment algorithm.

5. The ASSIGNER transmits the new slot positions to the other end; the TDMA protocol in Chapter 4 ensures that the modified local schedules of the endpoints and their one-hop neighbors will be free of transmission conflicts.

The slotted distributed algorithm does not enjoy analytical convergence properties like the fluid couterpart that guides the slot reassignments. We will evaluate its performance using simulations in large networks and various static and dynamic scenarios.

5.4 Performance evaluation

5.4.1 Experimental model and setting

We have implemented a packet-level simulator environment in C++ to evaluate the algorithm performance. The simulator includes the generation of various static and dynamic topology scenarios as well as an implementation of the proposed protocol.

Topology dynamics are modeled by having links going up and down in a static baseline topology [111]. This model captures the way mobility is manifested in multi-channel systems without delving into the details of the complex hand-off and link establishment protocols that should be used by a multi-channel system when nodes actually move. While important, such protocols are beyond our scope. Also this model allows for explicit control of parameters that affect the protocol performance such as topology density and frequency of topology changes.

Each link in the baseline topology cycles independently between an ACTIVE ("link up") and INACTIVE ("link down") state. A link remains ACTIVE for a geometrically distributed number of slots with mean T_{active}. Since all links alternate between the two states independently, the long-term fraction of time p a link is ACTIVE equals the average percentage of active links in the baseline topology at any time. In addition, certain multi-channel technologies impose a limit on the number of physical links a wireless node can maintain simultaneously. This restricts the maximum node degree to D_{max} (e.g. in Bluetooth D_{max} is 7). The parameter T_{active} is used to tune the rate of topology changes while p and D_{max} affect the average network density. The frequency of rate adjustments is controlled by the protocol parameter T_{adjust}. After a link rate adjustment, the endpoint nodes agree on a random rate adjustment timer chosen uniformly between 0 and T_{adjust} slots. The timer decreases on each future time slot the link is used for

transmissions. When the timer expires, the link is activated for rate adjustment.

We use two metrics to evaluate performance:

- **Relative computation error**: If the MMF rate of a link l at time t is $r_l^{MMF}(t)$ and the computed rate is $r_l(t)$, the relative computation error for link l at time t is $|1 - r_l(t)/r_l^{MMF}(t)|$. For each slot t, we consider the maximum and average relative computation error over all currently ACTIVE links. After each topology change, the reference link MMF rates are computed off-line using the centralized algorithm.

- **Control Overhead**: During network operation, a slot can be idle, used for transmission of a DATA packet or for exchange of control information conveyed by the control packets of the TDMA protocol. The control overhead is the ratio of control packets over the total number of packets transmitted during a simulation run.

In the experiments we consider a slot-synchronized network; we set $T_u^R = T_{system}$ ($C_u = 1$) for every node node u and consider bipartite topologies–in this case the entire feasibility region can be captured by the local conditions. For arbitrary topologies, nodes can set their QoS utilization parameters to $T_u^R = \lfloor 2/3 \cdot T_{system} \rfloor$ and the algorithm will target for MMF allocations with respect to this fractional capacity. We use an $N = 100$ node bipartite baseline topology with 50 nodes per bipartite set. This yields a rich set of $N^2/4 = 2500$ possible links in the baseline topology that can be ACTIVE or INACTIVE at any time. In terms of traffic demands, all links are assumed backlogged (no demand constraints) when ACTIVE.

5.4.2 Experiments on static networks

Given the baseline topology, the parameters p and D_{max} are used to derive static topologies of various density and maximum degree characteristics (e.g. Figure 5.7). All simulations in static topologies were run for 500000 slots.

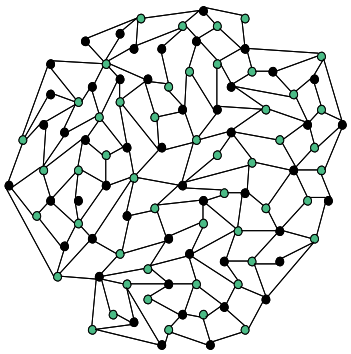
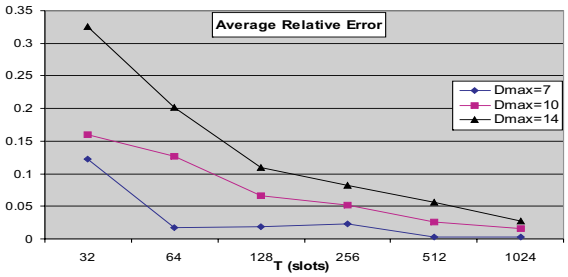


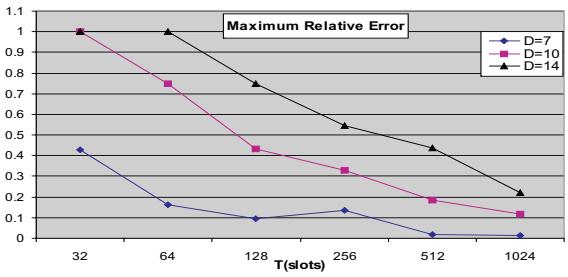
Figure 5.7: A sample $N = 100(50/50)$ bipartite topology of $p = 0.1$ and $D_{max} = 7$ derived from the baseline topology graph. Only ACTIVE links are shown.

In Figures 5.8 and 5.9 we set $p = 1.0$ so that every node has a degree of D_{max}. The target MMF rate every link in the network must reach is $1/D_{max}$ (approximated by T_{system}/D_{max} slots). Figure 5.8 shows the effect of the schedule period T_{system} and maximum degree constraint D_{max} on the average and maximum relative errors. For a fixed D_{max}, both errors decrease as T_{system} increases. One reason to explain this is that a larger period provides a better approximation to the reference (continuous) MMF rates.

For example a period of $T_{system} = 64$ cannot provide enough granularity for $D_{max} = 14$; the resulting errors are very high. The other reason is that a larger T_{system} offers more transmission slots to a link per period. This incurs more frequent expirations of the rate adjustment timer and, hence, more overall activations for link rate adjustment. This is also the explanation for the increase in the control overhead in Figure 5.9 as the period T_{system} increases. The maximum node degree D_{max} has a more pronounced effect both



(a) Average relative error



(b) Maximum relative error

Figure 5.8: (a) Average and (b) Maximum Relative Errors for a static network of $N = 100$ $p = 1.0$ and $T_{adjust} = 512$ slots for various choices of T_{system} and D_{max}. The average and maximum relative errors are computed over all active links at the last slot of each simulation run.

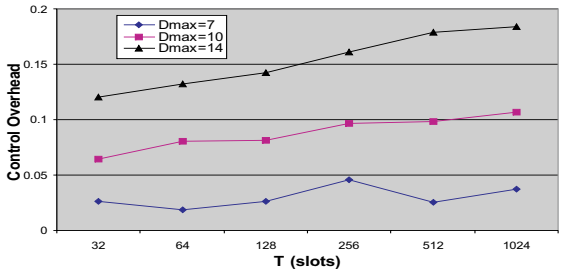


Figure 5.9: Control Overhead for a static network of $N = 100$, $p = 1.0$ and $T_{adjust} = 512$ slots for various choices of T_{system} and D_{max}.

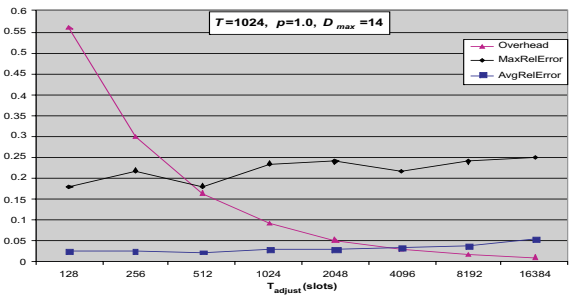
in the amount of error and control overhead. This is illustrated by the distance between the curves in both Figures 5.8 and 5.9. In the error curves, the effect of D_{max} decreases as the period T_{system} increases. After $T_{system} = 1024$ slots, the average relative error becomes less than 3% and the maximum error less than 20% for all cases. However, in terms of control overhead, the difference between the curves does not decrease with T_{system}. Thus for $T_{system} = 1024$, a $D_{max} = 7$ spends only 3% of transmissions in exchange of control packets while a $D_{max} = 14$ spends 17%. To keep the control overhead low, we need to reduce the frequency of rate adjustments that is controlled by the T_{adjust} parameter.

Figure 5.10(a) illustrates the effect of T_{adjust} on a ($T_{system} = 1024$, $D_{max} = 14$) system. By increasing T_{adjust} (hence decreasing the frequency of link rate adjustments) the control overhead decreases without any noticeable effect in the resulting maximum and average discrepancy from the MMF solution. At $T_{adjust} = 16384$ slots, the control overhead becomes negligible. Still, decreasing the frequency of link activations leads to a slower convergence. This will become obvious in the experiments of the dynamic topologies.

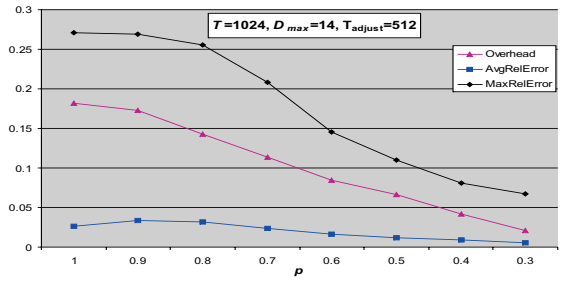
Figure 5.10(b) shows the effect of the topology density parameter p on the three metrics of interest. As the density decreases, less nodes need to establish the maximum number of links D_{max} and this leads to a reduction of both errors and control overhead in the network.

5.4.3 Experiments on dynamic networks

The parameter controlling the network dynamics is T_{active} for the rate of topology changes. To see how the time scale of topology dynamics affects the algorithm performance, we use the system technology parameters of Bluetooth. Bluetooth supports a



(a)



(b)

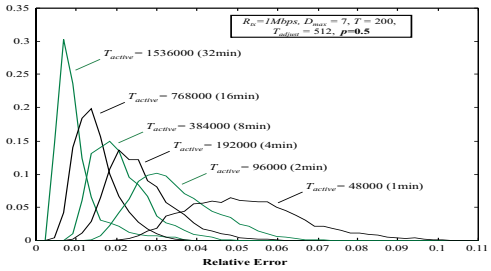
Figure 5.10: Effect of the frequency of link rate adjustments T_{adjust} (for $p = 1.0$) and (b) topology density p (for $T_{adjust} = 512$ slots) on the average and maximum link MMF errors and the control overhead. ($N = 100$ nodes, $T_{system} = 200$ slots, $D_{max} = 14$ links.)

raw transmission rate of $R_{tx} = 1Mbps$ and a maximum number of simultaneously active links $D_{max} = 7$. The system slot duration is $1.25ms$. We use a period of $T_{system} = 200$ slots, which is the maximum that can be supported by the current Bluetooth specification[1]. All simulations were run for 500000 slots. We consider the pdf distribution of the average relative error during the last 100000 slots.

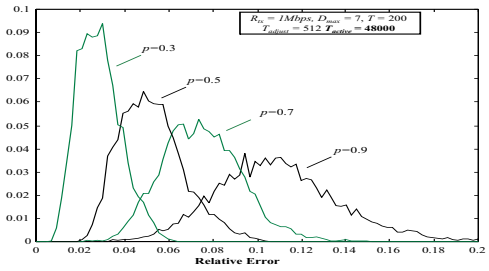
Figure 5.11 illustrates the effect of mobility and network density on the error distribution. The bell-shaped curves indicate that the MMF rate discrepancy experienced by an average link generally oscillates around a mean value. In Figure 5.11a, we let a link spend an equal average amount of time in the ACTIVE or INACTIVE state, by setting $p = 0.5$. The average time T_{active} a link alternates between the two states varies from $32min$ (1536000 slots) to $1min$ (48000 slots). As the rate of topology changes increases, both error mean and variance increase. This is illustrated by a right-shift and "spreading" of the error distribution curves as the parameter T_{active} decreases. For a quasi-static network ($T_{active} = 32min$), the MMF discrepancy of an average link is centered at 0.7% and varies between 0.2% and 4%. For $T_{active} = 1min$ the peak consists of a range of error values ($4\% - 6\%$) and the overall error dynamic range is $2\% - 10\%$.

For the same rate of topology changes, the mean and variance of the average relative error increase with topology density (Figure 5.11b). The reason is that a denser topology allows for less simultaneous conflict-free transmissions per period and hence less frequent expirations of the rate adjustment timer per link. Therefore rate adjustments are happening at a slower rate and this affects the ability of the algorithm to track topology

[1]Half duplex mini-slots in our model correspond to single-slot Bluetooth baseband ACL packets. The payload size of these packets is limited to 240 bits. If we exclude the higher layer headers and the CRC, only 216 bits are left for the protocol information (DH1 packets). When FEC is added (DM1 packets), the available space goes down to 136 bits. Using equation (4.2, Chapter 4), we can see that the maximum period T_{system} for DH1 packets is 200 slots and for DM1 packets 122 slots.



(a)



(b)

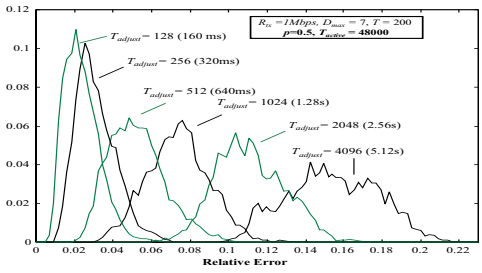
Figure 5.11: Effect of (a) rate of topology changes T_{active} (for $p = 0.5$) and (b) topology density p (for $T_{active} = 48000$ slots) on the distribution of the average link MMF error ($N = 100$ nodes, $T_{system} = 200$ slots, $D_{max} = 7$ links, $T_{adjust} = 512$ slots.).

changes. Still, even in the most dense topology ($p = 0.9$) and high rate of topology changes of $T_{active} = 1min$ (48000 slots), an average link will achieve above 80% of its target MMF rate.

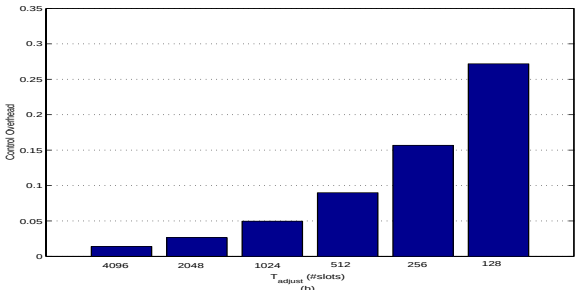
Figure 5.12 shows the effect of the rate adjustment parameter T_{adjust} in the most dynamic case where links form and fail every 1 minute (48000 slots) on the average. As T_{adjust} varies from $5.12s$ (4096 slots) to $160ms$ (128 slots), the error mean and variance decrease slightly (Figure 5.12a) but the control overhead increases (Figure 5.12b). For $T_{adjust} = 160ms$ (128 slots), the error is centered at 2% of the MMF rate but the control overhead needed to sustain it amounts to 27% of the overall number of transmissions. A T_{adjust} greater than 640ms (512 slots) keeps the overhead below 9% but the error mean and variance will gracefully increase according to Figure 5.12a.

Figure 5.13 illustrates how topology dynamics and density affect the algorithm performance had the reference technology specification allowed for a larger D_{max}. The curve trends are the same as in Figure 5.11 but the error means and variances increase with D_{max}. This shows the algorithm performance degradation for technologies using a certain radio transmission rate and wish to support a larger maximum number of MMF links per node in a dynamic network.

Technologies supporting higher transmission rates result in a better performance because they can use a shorter slot duration. For example if $R_{tx} = 2Mbps$ in the reference system, the system slot duration is $0.625ms$ instead of $1.25ms$ and therefore "$T_{active} = 2min$" in Figure 5.11a will now correspond to the error distribution of $T_{active} = 192000$ instead of the one of 96000 slots. As we double the transmission rate, we can see the corresponding performance improvement by moving one error distribution curve to the left in Figures 5.11a, 5.12a, 5.13a and one bar to the left in Figure 5.12b for the control overhead.

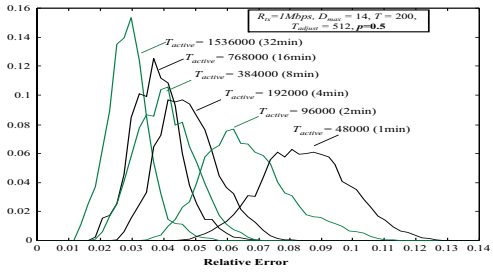


(a)

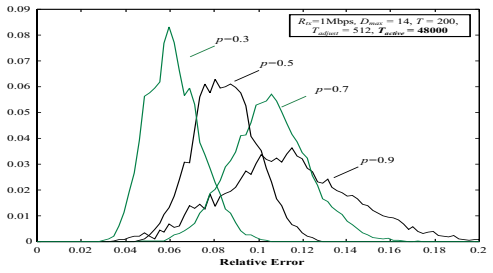


(b)

Figure 5.12: Effect of frequency of link activations T_{adjust} on (a) the distribution of the average link MMF error and (b) control overhead. ($N = 100$ nodes, $T_{system} = 200$ slots, $D_{max} = 7$ links, $T_{active} = 48000$ slots.)



(a)



(b)

Figure 5.13: Effect of (a) rate of topology changes T_{active} ($p = 0.5$) and (b) topology density p ($T_{active} = 48000$ slots) on the distribution of the average link MMF error ($N = 100$ nodes, $T_{system} = 200$ slots, $D_{max} = 14$ links, $T_{adjust} = 512$ slots.).

5.4.4 Traffic adaptation

In the previous section we considered backlogged links to measure the algorithm ability to track the MMF allocation subject to topology changes. Responsiveness to traffic dynamics can be incorporated using the upper bound B_l.

For each adjacent link l, each node measures the fraction of allocated slots that were actually utilized for transmission over a measurement interval $T_{measure}$ ($T_{measure} > T_{system}$). Let $\widehat{\rho}_l(n)$ be the estimated rate of link l at the beginning of the n_{th} measurement interval, and $\widetilde{\rho}_l(n)$ be the measured utilization after the n_{th} measurement interval. The estimated rate $\widehat{\rho}_l(n+1)$ for link l during the $n+1_{th}$ measurement interval can be computed using exponential averaging:

$$\widehat{\rho}_l(n+1) = \alpha\widehat{\rho}_l(n) + (1-\alpha)\widetilde{\rho}_l(n) \tag{5.3}$$

The parameter α ($0 \leq \alpha \leq 1$) is the weight given to the history of previous samples. This parameter can be set a priori depending on the desired degree of network adaptivity to traffic dynamics. Recall that provision of a robust view to the higher layers (more static TDMA schedule) and traffic adaptivity (more dynamic TDMA schedule) are two conflicting objectives. A high value of α should be preferred in the first case; a low in the second. Alternatively, α can be computed online based on the observed traffic. An excellent treatment of this topic can be found in [112].

The estimated value of $\widehat{\rho}_l(n+1)$ is used to set the upper bound parameter B_l of the link. More specifically, if $\widehat{\rho}_l(n+1)$ is greater than a threshold β (β should typically be greater than 0.9), the link is considered backlogged and B_l is set to 1. Otherwise, B_l is set to $\widehat{\rho}_l(n+1)$. The updated value of B_l is then passed to the MMF link scheduling algorithm. We are currently experimenting with this technique for various topology and traffic dynamics.

5.5 Related work

The max-min fairness objective has been addressed for both single channel and multi-channel ad hoc networks. Fairness is defined and addressed for single-hop flows in all cases. Single channel systems are considered in [105][106][107]. The work in [105] uses a weighted fairness scheme to first allocate a minimum fair bandwidth to the network flows and then maximize the system utilization subject to this allocation. This approach can reach the MMF allocation using appropriate flow weights. However, the weight computation would require knowledge of the MMF rates. This in turn would require a global network MMF rate pre-computation phase a difficult task in a large dynamic network. Nandagopal et al. [107] define fairness in terms of maximizing total logarithmic user utility functions and implement proportional fairness within this framework. Max-min fairness is mentioned as a asymptotic case of this utility fairness model. A centralized and a distributed algorithm specifically targeted for max-min fairness are proposed in [106]. The centralized algorithm reaches an approximate solution for large networks because it relies on the computation of the clique corpus of a graph, which is a NP-complete problem. In the distributed algorithm a node maintains a subset of the contention graph and heuristically computes a coarser allocation.

It should be noted that in [105][106][107], the distributed algorithms that approximate the fairness models are implemented using a random access MAC protocol and attempt to achieve the desired rates by setting a per-flow back-off timer according to the fair weight of the flow. Since random access cannot support strict bandwidth allocation guarantees, fairness can be achieved only in a probabilistic sense in this case (very large time scales).

The work in [108] defines the max-min fairness objective in a slotted multi-channel system using scheduled access and provides a scheduling policy that achieves max-min

fair allocation of flows. At each slot, a node first assigns appropriate weights to each of its adjacent flows by using a round robin token generation scheme. Then the flows that constitute a maximum weighted matching on the network are scheduled to transmit conflict-free. This step makes this approach unsuitable for distributed implementation because it requires global topology information for the maximum weighted matching computation.

DSSA [81], a distributed TDMA scheduling algorithm for Bluetooth scatternets, cannot be applied to the max-min fairness objective. In DSSA nodes start with an knowledge of demands on their adjacent links and try to reach a conflict-free schedule of short length that satisfies these demands. However max-min fairness is a global objective. Hence, to use this algorithm one must first pre-compute the MMF rates and then provide them as local traffic demands to every node in the network–this is not practical.

Distributed algorithms for MMF rate computation for multi-hop sessions have also been studied extensively in the wireline networks context [113][114]. Our algorithm is similar because it is asynchronous, distributed and targets max-min fairness. The difference is that these algorithms perform only the fluid model portion: they only compute the MMF rates but do not specify how to enforce them. Rate enforcement is treated separately by using end-to-end or hop-by-hop link schedulers and traffic shapers [115][116]. This separation is perfectly justified due to the link scheduling independence in wireline networks. In the wireless case, rate adjustment on a link has an effect on the rates of links adjacent to both endpoint nodes; the problems of rate computation (fairness deficit computation) and rate enforcement (conflict-free slot assignment) must be addressed jointly.

5.6 Conclusions

Future deployment of wireless ad hoc networks calls for decentralized techniques that efficiently allocate the scarce wireless medium to mobile users. We presented a distributed asynchronous algorithm of low complexity aiming for max-min fairness. Bandwidth allocations are realized by conflict-free periodic link schedules. This implies both short-term (with respect to T_{system}) and long-term fairness properties.

A unique feature of the distributed scheduling technique is that it does not assume any initial knowledge about the (global) MMF objective. Instead, the rate computation and enforcement occur simultaneously by means of local and incremental conflict-free schedule updates. This incremental property allows for natural adaptation to network dynamics without the need to suspend communications and restart the schedule computation from scratch. The scheduling mechanism is driven by the rate computation algorithm, which converges to the MMF solution under the fluid model. Still, when emulating the fluid algorithm in the slotted world the convergence is not exact and there are restrictions and trade-offs a designer has to take into account. To this end, we provide an analysis of the algorithm communication requirements and its effect on the design choices of a technology supporting it.

The algorithm was extensively tested under various technology choices and topology dynamics. For static networks it demonstrated excellent convergence properties especially as the schedule period T_{system} increases. For dynamic scenarios, an average link typically experiences a certain mean MMF discrepancy with a finite variance. Performance gracefully degrades with the increase in the rate of topology changes, network density and desired maximum number of physical links supported by a wireless node. In highly dynamic scenarios and stringent technology constraints (modest R_{tx} and high D_{max}), the incremental nature of the algorithm allows the network to be rea-

sonably close to the MMF solution most of the time. In addition, the frequency of link rate adjustments can be fine tuned to achieve acceptable performance for low control overhead.

Chapter Appendix 5.A—Pseudocodes of Centralized MMF, FDC and slot assignment algorithms

Procedure `CentralizedComputeMMF`

Computing the MMF rates using global information

input : $G(N,L), \{0 \leq B_l \leq 1,\}$

output : MMF vector $r = (r_1, ..., r_f, ..., r_{|L|})$

Intialization: $i = 1, U_n^0 = 0\ \forall n \in N, r_l^0 = 0\ \forall f \in L, L^1 = L, N^1 = N$

$$C_n = \begin{cases} 1 & \text{if } G(N,L) \textit{ bipartite} \\ 2/3 & \text{otherwise} \end{cases} \quad \forall n \in N$$

repeat

1 $f_n^i =$ # of links in F^i adjacent to node n;

2 $K_1 = \min_{n \in N^i} \left(\dfrac{C_n - U_n^{i-1}}{f_n^i} \right), K_2 = \min_{f \in F^i} (B_f - r_f^{i-1})$;

3 $dr^i = min(K_1, K_2)$;

4 $B^i = \left\{ m : m = arg \min_{n \in N^i} \left(\dfrac{C_n - U_n^{i-1}}{f_n^i} \right) \right\}$;

5 $$\widehat{F}^i = \begin{cases} \left\{ f : f = arg \min_{j \in F^i} \left(B_j - r_f^{i-1} \right) \right\} & \text{if } K_1 > K_2 \\ \left\{ f : f \textit{ is adjacent to every } n \in B^i \right\} & \text{otherwise} \end{cases}$$

6 $r_f^i = r_f^{i-1} + dr^i, \forall f \in F^i$;

7 $U_n^i = \sum_{f \textit{ adjacent to } n} r_f^i$;

8 $N^{i+1} = \{n : C_n - U_n^i > 0\}$;

9 $F^{i+1} = F^i - \widehat{F}^i$;

10 $i = i + 1$;

until *(F^i is empty)*;

Figure 5.14: Centralized algorithm for computing the link MMF rates

Procedure FDC

The fairness deficit computation algorithm

input : $\boldsymbol{r}_u,\ l,\ B_l$

output : $\boldsymbol{r}'_u,\ fd_l^{(u)}$

Intialization: $t = 0,\ \boldsymbol{r}'_u = \boldsymbol{r}_u\ E_u = C_u - \sum_{k \in L(u)} r_k$;

1 $r'_l = r_l + E_u$ /*Increase by the available node bandwidth*/;

2 $r_{max} = \max_{k \in L(u)} r'_k$;

3 **while** *($r'_l < r_{max}$) and ($r'_l < B_l$)* **do**

$t = t + 1$;

$r_{max} = \max_{k \in F(u) - \{l\}} r'_k$;

$M^{(t)} = \left\{k_1, ..., k_m : r'_{k_1} = ... = r'_{k_m} = r_{max}\right\}$;

$m = |M^{(t)}|$;

$r'_{k_1} = ... = r'_{k_m} = r'_l = \frac{r'_{k_1} + ... + r'_{k_m} + r'_l}{m+1}$;

end

4 **if** *($r'_l \geq B_l$) /*Demand constraint less than the fair share*/* **then**

$r'_l = B_l$;

$r'_k = r'_k + \frac{r'_l - B_l}{m}, \forall k \in M^{(t)}$;

end

5 $fd_l^{(u)} = r'_l - r_l$;

Figure 5.15: FDC pseudocode

Procedure `AssignSlots`

The slot assignment algorithm

input : $\boldsymbol{x}_u,\ l,\ \boldsymbol{S}_u,\ \boldsymbol{S}_v,\ T_{system}$

output : $\boldsymbol{S}'_u,\ \boldsymbol{d}_u$

Intialization: $\boldsymbol{S}'_u = \boldsymbol{S}_u,\ \boldsymbol{d}_u(s) = 0,\ \forall s = 0, ..., T_{system} - 1$;

begin

/*Phase I: Assign to l the slots in $\boldsymbol{S}_u$ that are concurrent to idle slots in $\boldsymbol{S}_v$*/;

1 $\boldsymbol{I}_0 = \{s : \boldsymbol{S}'_u(s) = idle$ **and** $\boldsymbol{S}'_v(s) = idle\}$;

repeat

Select a random slot position s from set I_0 ;

$\boldsymbol{S}'_u(s) = l$, $\boldsymbol{d}_u(s) = 1$ /*Assign slot position s to link l in $\boldsymbol{S}'_u$*/;

$x_l = x_l - 1$, $\boldsymbol{I}_0 = \boldsymbol{I}_0 - \{s\}$;

until *($x_l == 0$ OR $\boldsymbol{I}_0$ is empty)*;

if *($x_l == 0$)* **then**

Stop and exit procedure ;

end

2 Form the set of surplus links $X_u^- = \{k : x_k < 0\}$;

for *Every link* $k \in X_u^-$ **do**

$\boldsymbol{I}_k = \{s : \boldsymbol{S}'_u(s) = idle$ **and** $\boldsymbol{S}'_v(s) = idle\}$;

repeat

Select a random slot position s from set I_k ;

$\boldsymbol{S}'_u(s) = l$, $\boldsymbol{d}_u(s) = 1$ /*Assign slot position s to link l in $\boldsymbol{S}'_u$*/;

$x_l = x_l - 1$, $\boldsymbol{I}_k = \boldsymbol{I}_k - \{s\}$;

until *($x_k == 0$ OR $\boldsymbol{I}_k$ is empty)*;

if *($x_l == 0$)* **then**

Stop and exit procedure ;

end

end

/*Phase II starts here*/

3 **for** *Every link* $k \in X_u^-$ **do**

if $x_k < 0$ **then**

/*If link k has still slots to provide after Phase I*/

Select random x_k positions and form $\boldsymbol{I}_k = \{s : \boldsymbol{S}'_u(s)\}$;

for *every slot position* $s \in \boldsymbol{I}_k$ **do**

$\boldsymbol{S}'_u = l,\ \boldsymbol{d}_u(s) = 1$ /*Assign slot s to link l in $\boldsymbol{S}'_u$*/;

end

end

end

end

Figure 5.16: The slot assignment algorithm

Chapter 6

End-to-end rate guarantees in wireless ad hoc networks

We present a framework for provision of end-to-end bandwidth guarantees in wireless ad hoc networks. Guided by local feasibility conditions multi-hop sessions are dynamically offered bandwidth, further translated to link slot demands. Using the distributed TDMA protocol, nodes adapt to the demand changes on their adjacent links by local, conflict-free slot reassignments. As soon as the demand changes stabilize the nodes must incrementally converge to a TDMA schedule that enforces the global link (and session) demand allocation. Therefore, the framework consists of two processes that operate in parallel: an end-to-end algorithm for computing session rates according to a QoS objective and a dynamic link scheduling algorithm for enforcing these rates.

The dynamic link scheduling problem was partially addressed in Chapter 4 for enforcement of link MMF rates. The resulting slot assignment algorithm operated for arbitrary topologies but is specific to slot synchronized systems. In addition, though it demonstrated excellent properties through simulations, no analytical guarantees were provided with respect to convergence. In this chapter we solve the dynamic link scheduling problem for tree topologies. The link scheduling algorithm does not require slot

synchronization, realizes all feasible rates for trees and guarantees convergence within a finite time period. An upper bound on the convergence delay is also computed. Tree topologies arise in several ad hoc networking applications such as Bluetooth scatternets, sensor networks, power-aware multicasting or backbone structures used for administrative purposes such as routing.

Dynamic link scheduling focuses on converging to a TDMA schedule realizing the link demands and is agnostic of the specifics of the higher layer algorithm that allocates bandwidth to the end-to-end sessions. This allows definition and realization of various models for end-to-end Quality of Service provision. We consider both Constant Bit Rate (CBR) and Available Bit Rate (ABR) services.

End-to-end CBR service is implemented by QoS routing algorithms. Current approaches for QoS routing in ad hoc networks either focus on mobility and do not take medium access into account [59][36] or use complex admission control tightly coupled with the underlying TDMA protocol [58][57]. Both approaches result in network underutilization. We show that admission control within our framework is far simpler than [58][57] bearing a similar formulation to wireline networks. For tree topologies it yields maximum network utilization.

According to ABR, sessions do not have specific bandwidth requirements–they request from the network the maximum available bandwidth. In this setting, network resources must be shared to the sessions in a fair manner. For ABR, the preferred notion of fairness is MMF. In [61], Sarkar and Tassiulas introduce a backpressure/window-based flow control algorithm for computing the session MMF rates. However, the slotted TDMA scheme that enforces these rates requires global topology information and network-wide slot synchronization. We introduce an asynchronous distributed rate-based algorithm for MMF rate computation, similar to approaches used in wireline ATM

networks. Being rate-based, this algorithm can be combined with a distributed TDMA link scheduling protocol to enforce the computed MMF rates.

Finally, we present the implementation of our framework over Bluetooth. The interaction of the end-to-end MMF rate computation and tree link scheduling algorithm are investigated through extensive simulations–both algorithms demonstrate excellent performance in practice.

The chapter is structured as follows: In Section 6.1 we present the distributed dynamic scheduling algorithm for trees. Section 6.2 elaborates on the integration of link scheduling with end-to-end bandwidth allocation. Section 6.3 presents the detailed implementation of the bandwidth allocation framework over Bluetooth. Section 6.4 surveys related work. Section 6.5 concludes.

6.1 Distributed dynamic link scheduling for tree-based ad hoc networks

6.1.1 Network architecture, assumptions and definitions

The ad hoc network uses the multi-channel TDMA architecture and distributed TDMA protocol of chapter 3. Each node uses a local periodic schedule $\boldsymbol{S}_u$ of T_{system} slots to coordinate transmissions on its adjacent links. We will consider the (more general) asynchronous mode, where the local schedules are not slot aligned and time slot reference for communication on each link is provided by the master node endpoint. The network topology is a tree. If the network is mobile, we assume a distributed topology control protocol maintains the tree structure [29][31][77]. The nodes neither maintain a global view of the network nor know their level within the tree–they are only aware of

parent-child relationships with their one-hop neighbors.

Chapter 3 established that feasibility is guaranteed if each node u uses the following local conditions for the demands on its adjacent links:

$$\sum_{l \in L(u)} \tau_l \leq T_u^R, \quad T_u^R \leq T_{system} - \sum_{l \in L(u)} J_l^{(u)} \tag{6.1}$$

where,

$$J_l^{(u)} = \begin{cases} 1 & \text{if } asynchronous\, mode \textbf{ and } u\, is\, slave\, on\, link\, l \\ 0 & \text{otherwise} \end{cases} \tag{6.2}$$

If the QoS utilization parameter T_u^R equals the upper bound, the local conditions capture the entire set of feasible allocations. Without loss of generality we will assume this is the case here, i.e. all network capacity is used for QoS traffic.

Nodes reassign slots in response to demand changes on their adjacent links. The distributed TDMA protocol ensures that the network is always free of transmission conflicts. The distributed link scheduling algorithm runs on top of this protocol and determines which slot positions should be modified during each link rate adjustment so that eventually nodes converge to the global TDMA schedule realizing the current link demand allocation. Before presenting the algorithm we introduce the notions of *satisfied* and *stable* links. Let τ_l be the current demand for link $l = (u, v)$, and $t_l^{(u)}$ be the number of conflict-free slots currently assigned to l in the local schedule $\boldsymbol{S}_u$ of node u.

Definition 1: Node u calls its child link l_c *satisfied* if the following conditions hold:

STF1: The link is scheduled in a single window $W_{l_c}^{(u)} = [s_{l_c}^{(u)}, e_{l_c}^{(u)}]$ in $\boldsymbol{S}_u$.

STF2: The current demand is exactly satisfied by the current assignment: $t_{l_c}^{(u)} = \tau_l + J_{l_c}^{(u)}$.

where $J_l^{(u)}$ is given by eq. (6.2).

Let the parent link $l_p = (u, p)$ of node u be satisfied by a window $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ in $\boldsymbol{S}_u$. Also, let the children links $l_c = (u, c)$ of u be assigned distinct priorities p_{l_c}.

A child link l_c of u is *stable* if 1) it is satisfied and 2) the position of window $W_{l_c}^{(u)} = [s_{l_c}^{(u)}, e_{l_c}^{(u)}]$ in $\boldsymbol{S}_u$ provides enough room for scheduling all links of lower priority according to their current demands. More formally, this can be expressed as follows:

Definition 2: Node u calls its child link l_c stable if the following conditions hold:

STBL1: Link l_c is satisfied.

STBL2: $|[e_{l_c}^{(u)} \oplus 1, s_{l_p}^{(u)} \ominus 1]| \geq \sum_{k \in CH(u): p_k < p_{l_c}} (\tau_k + J_k^{(u)})$

where $CH(u)$ is the set of children links of u and "$\oplus$" and "$\ominus$" are Modulo-T_{system} addition and subtraction, respectively.

6.1.2 The distributed algorithm

Central to the algorithm operation is procedure SampleReschedule(). This procedure is asynchronously triggered for execution at a node either 1) when the higher layer process changes the demand of an adjacent link or 2) right after an adjacent link is rescheduled. When either of these events occurs, a non-root node u proceeds in execution of SampleReschedule() only if its parent link l_p is satisfied; the root proceeds in execution unconditionally.

During SampleReschedule() the following actions are performed at node u:

1) If u is not the root, let $W_{l_p}^{(u)} = [s_{l_p}^{(u)}, e_{l_p}^{(u)}]$ be the window in $\boldsymbol{S}_u$ satisfying its parent link l_p. First, u assigns decreasing priorities to its children links in the (circular) order that they currently appear in $\boldsymbol{S}_u$, starting at slot $e_{l_p}^{(u)}$ and ending at $s_{l_p}^{(u)}$. (The root node assigns priorities using 0 and $T_{system} - 1$ as start and end slots, respectively).

2) By inspecting $\boldsymbol{S}_u$, node u samples its children in decreasing priority for violation of the stability conditions. If all links are found stable, SampleReschedule() terminates and no action takes place. Otherwise, the highest priority unstable child link l_c is found. Let l_m be the stable link of immediately higher priority than l_c. If l_c is the highest priority child link, l_m is defined to be the parent link l_p. In either case, link l_m is satisfied. Link l_c needs to be rescheduled and stabilized.

3) Node u initiates rate adjustment on l_c by exchanging SC_INFO packets with the child endpoint c. After the exchange, u becomes the ASSIGNER and needs to determine a new set of slot positions for link l_c. First, u erases from $\boldsymbol{S}_u$ all slots currently allocated to l_c and considers a fresh allocation for a window W_{l_c} of $\tau_{l_c} + J_{l_c}^{(u)}$ contiguous slots. The position of W_{l_c} in $\boldsymbol{S}_u$ is determined as follows:

- First, node u computes the closest slot position to $s_{l_p}^{(u)}$ for which the stability conditions for l_c will hold:

$$s_{max} = s_{l_p}^{(u)} \ominus \sum_{k \in CH(u): p_k < p_{l_c}} (\tau_k + J_k^{(u)}) \tag{6.3}$$

  Let $W_{l_m}^{(u)} = [s_{l_m}^{(u)}, e_{l_m}^{(u)}]$ be the window satisfying the demand of link l_m in $\boldsymbol{S}_u$. Link l_c will be stable if window W_{l_c} is scheduled within the window $W_{max}^{(u)} = [e_{l_m}^{(u)} \oplus 1, s_{max} \ominus 1]$.

- Node u decides on the position of $W_{l_c}^{(u)}$ within W_{max}: The new position of $W_{l_c}^{(u)}$ may cancel slots of lower-priority children links in $\boldsymbol{S}_u$. Also, the position of $W_{l_c}^{(u)}$ will be enforced to the local schedule of the child node c and may cancel slots

on some of the children links of c. Using the local schedule of c (provided in the SC_INFO packet) the position of $W_{l_c}^{(u)}$ is selected within W_{max} such that the total number of affected links at both node endpoints is minimized.

4) Once u determines the position of $W_{l_c}^{(u)}$, it initiates the distributed coordination mechanism of the TDMA protocol by issuing SC_UPD packets to its affected neighbors. The protocol ensures that the local schedules of endpoint nodes u and c (as well as the local schedules of their affected neighbors) will be free of transmission conflicts after the update.

After l_c is scheduled, node u must restart sampling from the highest priority child link for any violation of the stability conditions. This must be done because the demands of links of higher priority than l_c may have changed while the rate adjustment was taking place. If the demands stop changing, repetitive invocation of procedure SampleReschedule() will reschedule and stabilize the unstable links in decreasing priority. The sampling-rescheduling loop terminates when all child links are found to be stable.

An example of SampleReschedule() is shown in Fig. 6.1. According to the initial local schedule $\boldsymbol{S}_u$ (Fig. 6.1(c)), the allocations on adjacent links of node u are $(t_{l_p}^{(u)}, t_1^{(u)}, .., t_4^{(u)}) = (2, 2, 3, 4, 3)$ and corresponding demands are $(\tau_{l_p}, \tau_1, ..., \tau_4) = (2, 2, 2, 3, 3)$. In Fig. 6.1(b) the demand of link 3 changes from 3 to 6 slots. Since the parent link l_p is satisfied ($t_{l_p}^{(u)} = \tau_{l_p} + J_{l_p}^{(u)} = 4$), node u initiates SampleReschedule(). Using the window $[0, 1]$ assigned to its parent link l_p, u assigns decreasing priorities to its children links in the cyclic order they appear in $\boldsymbol{S}_u$, starting from slot 1 towards slot 0. The links in decreasing priority are $2, 1, 4, 3$. Figures 6.1(c)-(f) illustrate a sequence of steps and modifications of S_u that stabilize the links.

The above description corresponds to the desired operation of SampleReschedule() at a node u. However, the fact that nodes may be busy at any time makes things

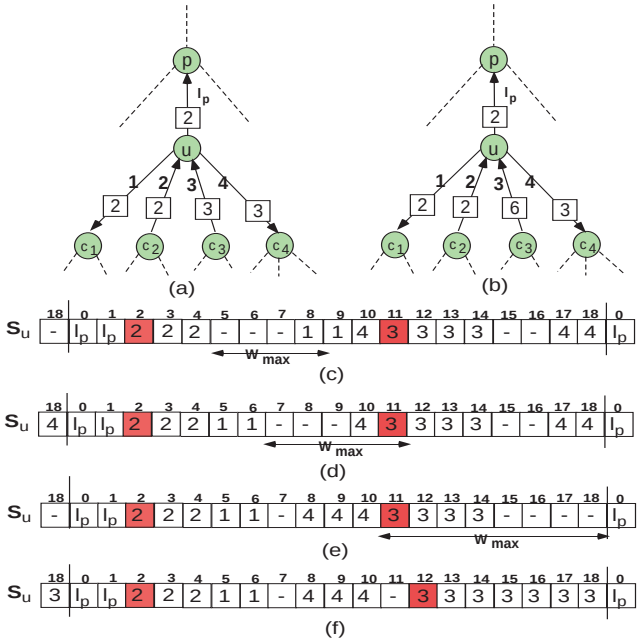


Figure 6.1: (a) Arrows denote master-slave relationships and red slots denote switching slots of links where u is slave. (b) Demand of link 3 changes from 3 to 6. (c) The highest priority child link (2) is satisfied and the distance of slot 5 to slot 18 ($|[5, 18]| = 14$) is greater than the current demand sum of the lower priority child links ((2+0)+(6+1)+(3+0)=12)–link 2 is stable. The next priority link 1 is satisfied but not stable ($|[10, 18]| = 9 < (6 + 1) + (3 + 0) = 10$). To satisfy condition **STBL2**, window W_1 ($\tau_1 + J_1^{(u)} = 2 + 0 = 2$ slots) must be within $W_{max} = [5, 8]$. (d) $\boldsymbol{S_u}$ after link 1 has been rescheduled. The position was decided after executing the TDMA protocol with node c_1 for link (u, c_1) and consulting with S_{c_1}. Link 4 is not satisfied (**STF1** does not hold); it needs to be rescheduled within $W_{max} = [7, 10]$ to become stable. (e) $\boldsymbol{S_u}$ after link 1 has been rescheduled. Link 3 is not satisfied; it can be rescheduled within $W_{max} = [11, 18]$. (f) All links are now stable–the sampling-rescheduling loop is complete.

more complicated. For example, when the highest priority unstable child link is sampled, it may be currently busy scheduling a child of its own and, therefore, unavailable for re-scheduling. Hence, a need exists for coordinating parent and children to allow proper operation of the sampling re-scheduling loop. This is accomplished by the STABLE_REQ/STABLE_ACK packet exchange. Before executing SampleReschedule() node u requests permission from its parent by sending a STABLE_REQ packet. The parent will respond in one of two possible ways: either 1) it replies with a STABLE_ACK packet as permission for node u to continue sampling and rescheduling its children or 2) it initiates a rate adjustment on this link via a SC_INFO packet.

In the example of Fig. 6.1, node u must perform a STABLE_REQ/STABLE_ACK handshake with its parent p before rescheduling any unstable child link of its own. If link l_p becomes unstable during this process, the parent will respond with an SC_INFO packet and link l_p will be rescheduled. Based on the new stable window l_p, node u will reassign priorities and resume the sampling-rescheduling loop. The detailed operation of the asynchronous protocol, called STABLE_TREE, is described in Figure **??**, in Chapter Appendix 6.A.

Theorem 6.1.1 (Convergence Theorem) *Consider an initial tree topology and network TDMA schedule. Assume that a set of arbitrary demand and topology changes occur that eventually stabilize to a new tree topology and demand allocation $\boldsymbol{\tau}$ obeying the capacity condition of eq. (6.1). The asynchronous distributed algorithm will converge to a new TDMA schedule realizing $\boldsymbol{\tau}$ in a finite number of link rate adjustments.*

Proof In general, nodes may re-assign slots using SampleReschedule() when their adjacent links are detected unsatisfied. We will show that, as soon as the changes in link demands stabilize, convergence is guaranteed to occur progressively from the root downward.

We assume that changes on a link demand are detected by both node endpoints (not necessarily at the same time instant) and that control messages are not lost due to channel errors. Mobility can be treated as a special case of link demand changes with a link failure being a transition to zero demand and a link establishment being a transition from zero to a positive demand satisfying the local feasibility conditions.

We define the level of a node to be its hop distance from the root (root has zero level). In addition, we define the level of a link to be equal the level of its child node endpoint. Given an arbitrary set of demand or topology changes that have stabilized, let K_{min} be the link level such that all links of level K_{min} or less have not been affected by the changes. We will prove convergence by induction on the link levels $k > K_{min}$ that have been affected by the changes. We distinguish two cases for K_{min}:

Case A $K_{min} = 0$**:** At least one of the child links of the root has been affected by the changes.

Level 1: Link level 1 includes the root and its children. Upon detection of any unsatisfied link, the root will run SampleReschedule() only if it is not busy or after it has finished scheduling its current link. Let l_c the highest-priority unstable child link. We distinguish two cases for the child node endpoint c of l_c:

Case 1: Node c not busy: the root initiates scheduling of l_c by sending an SC_INFO packet to c (line SR-4, Fig. 6.8, Chapter Appendix 6.A).

Case 2: Node c currently busy: the root exits SampleReschedule(). When node c completes scheduling, it will send a STABLE_REQ packet to the root (line E1-3, Fig. 6.9, Chapter Appendix 6.A). It also becomes unavailable for rescheduling its own children until it receives a response from the root (line E1-2, Fig. 6.9, Chapter Appendix 6.A). Upon reception of the STABLE_REQ packet, the root executes SampleReschedule(). Since the link demand changes have stopped, the highest priority child will be

again node c–it is guaranteed not to be busy this time (due to line E1-3 in Fig. 6.8, Chapter Appendix 6.A, node c will not enter SampleReschedule() upon reception of STABLE_REQ packets from its children.). Next the root initiates scheduling on l_c (line SR-4, Fig. 6.8, Chapter Appendix 6.A). In a similar fashion, the root will eventually schedule all level-1 unstable links in decreasing order of their priority.

Level k: Assume that all links up to and including level k have been scheduled and stabilized. We will show that all level $k+1$ unstable links will be scheduled and stabilized in a finite number of iterations .

Since every level-k node u has been independently assigned a stable parent link window $W_{l_p} = [start_{l_p}, end_{l_p}]$, it suffices to consider one such node in isolation. Each time node u needs to execute SampleReschedule(), it asks permission from its parent node p by sending a STABLE_REQ packet. Since l_p is stable, the parent p will always reply with a STABLE_ACK packet (line E2-1, Fig. 6.9, Chapter Appendix 6.A).

As soon as u receives permission to run SampleReschedule(), we have the same case of the root node and the level-1 links. Therefore, all unstable children links of node u will eventually be re-scheduled and stabilized. Since this will happen for all level-k nodes and their level $k+1$ children links, the induction step is complete.

Case B $K_{min} > 0$: This case can be proven using as initial inductive step $k = K_{min}$. The initial step holds since it is similar to the Level-k inductive step of the case $K_{min} = 0$. For level $k > K_{min}$ to $k+1$, a similar argument to the one used in case A is applicable. ■

The convergence delay of STABLE_TREE depends on the tree depth and the system period T_{system}. For a worst-case analysis, assume that all links have become unsatisfied due to the link demand changes. Since convergence is guaranteed from the root downward, in the worst-case scenario, all links will need to be rescheduled in this order. Also,

the worst tree topology is a line starting at the root node–in this case all ($N-1$) links will be scheduled sequentially in time.

According to the distributed protocol analysis in Chapter 3, the maximum duration of a link rate adjustment is $5T_{system}$ slots (Property 3). Hence, when a node samples the highest priority unstable link, it will wait at most $5T_{system}$ slots in case the child node is busy. Thus, each link on the line will be scheduled in at most $10T_{system}$ slots. We conclude that once link demands have stabilized, STABLE_TREE converges within $10NT_{system}$ slots.

The worst-case analysis assumes all links become unsatisfied and rescheduling will happen in the order that guarantees convergence–starting from the root downward. Since nodes continuously detect changes and reassign slots locally, convergence may occur faster in practice. In addition, demands may be changing locally at lower tree levels; only part of the tree will need to be rescheduled in this case. Existing tree topology control algorithms strive to maintain balanced structures. In this case, even if links will need to be scheduled from the root downwards, multiple links will be scheduled in parallel. Also, during a link rate adjustment, not all neighbors are always affected and acknowledgements may arrive in less than T_{system} slots. The convergence behavior of STABLE_TREE in practice will be investigated in Section 6.3 together with end-to-end bandwidth allocation mechanisms (addressed next).

6.2 End-to-end rate guarantees

We now introduce a framework for integrating link scheduling with end-to-end bandwidth allocation. The asynchronous TDMA ad hoc network is shared by a set of unicast multi-hop sessions. Without loss of generality, we assume that half-duplex parts of a

slot have equal duration (D_{slot}) and are used by the same session. Although bidirectional transfer is supported over a path, we assume that data flows in a single direction.

Each node can transmit at a maximum rate of R bps on a link. To support a rate of $\rho_i (\leq R)$ bps for session i over a path, the network must be able to allocate $\tau_i = \lceil (\rho_i / R) \cdot T_{system} \rceil$ conflict-free slots for i to all links in the path.

Since each slot assigned to a link can be used only by a single session, the total bandwidth consumed by the sessions $F(u)$ sharing node u must obey the local feasibility conditions:

$$\sum_{i \in F(u)} \delta_i^{(u)} \cdot \tau_i \leq T_u^R, \ \forall u \in N \tag{6.4}$$

where

$$\delta_i^{(u)} = \begin{cases} 1 & \text{if } u \, is \, source \, or \, destination \, of \, session \, i \\ 2 & \text{otherwise} \end{cases}$$

The term $\delta_i^{(u)}$ indicates that, in order to support allocation τ_i for session i, an intermediate node u must be able to communicate for τ_i slots on both upstream and downstream links of the session. The maximum value for T_u^R –given in Figure 4.4, Chapter 4–depends on existence or not of global slot synchronization and the topology control used in the network (if any).

The integrated framework provides end-to-end bandwidth guarantees using three independent components:

- **End-to-end rate allocation:** Sessions are allocated (feasible) rates according to eq. (6.4).

- **Link scheduling:** The session rates are translated to (feasible) link demands:

$$\sum_{i \in S(l)} \tau_i = \tau_l, \ \forall l \in E \tag{6.5}$$

where $S(l)$ is the set of sessions crossing link l. The link demands are realized by a distributed dynamic link scheduling algorithm. STABLE_TREE is such an algorithm for tree topologies.

- **Session packet scheduling** Once link scheduling converges, every link has been allocated enough bandwidth (conflict-free slots) to support the session demands. The slots allocated to each link can be shared to its sessions according to their demands, using Weighted Round Robin (WRR), Weighted Fair Queuing (WFQ) [115] or other single-server queuing disciplines. Since our TDMA architecture uses slots of fixed-size, WRR would be a reasonable choice. Another possibility is to combine First-Come-First-Serve (FCFS) queuing at intermediate links with explicit control of the transmission rates at the source nodes. The choice will depend on the target environment and application requirements.

Decoupling session rate allocation from link scheduling allows definition and realization of various end-to-end QoS objectives. In the following sections we introduce end-to-end rate allocation mechanisms for Constant Bit Rate (CBR) and Available Bit Rate (ABR) services.

6.2.1 Constant Bit Rate (CBR) Service

According to the CBR service model, sessions have fixed rate requirements that need to be satisfied by the network. A typical application is packetized voice. For each session arriving at a source node, a path supporting the requested rate to the destination must be determined.

Session i with rate demand ρ_i bps can be admitted on a path $u_1, u_2, ..., u_p$ if the corresponding demand allocation $\tau_i = \lceil (\rho_i / R) \cdot T_{system} \rceil$ does not exceed the minimum

available node capacity over the path. Therefore, session i is admitted if:

$$\tau_i \leq \min_{k \in 1,..,p} \left\lfloor \frac{T^R_{u_k} - \sum_{j \in F(u_k)} \delta_j^{(u_k)} \tau_j}{\delta_i^{(u_k)}} \right\rfloor \tag{6.6}$$

A similar admission control rule is used in wireline networks. The difference here is that the shared resources over the path are nodes instead of links. The rule in eq. (6.6) admits sessions without taking into account the arrangement of slots in the current TDMA schedule. This is possible due to the underlying dynamic link scheduling algorithm. If session i is admitted, the demands of all links on the selected path are increased by τ_i slots. As soon as the nodes in the path detect the demand changes on their adjacent links, they use the link scheduling algorithm to re-assign transmission slots and converge to a new TDMA schedule realizing the new link (and end-to-end) demand allocation. In case the session is admissible by multiple paths, a path selection criterion similar to ones used for wireline networks can be used (see [117] and references therein). The admission control rule over a single path and the path selection criterion together constitute a QoS routing algorithm.

TDMA-based QoS routing in ad hoc networks has also been considered in [58] for multi-channel systems and [57] for single-channel systems. The main difference of these algorithms with our approach is that they do not allow slot reassignments to accommodate incoming sessions. Instead, they keep the slots assigned to existing sessions fixed and seek to allocate available slots to incoming sessions subject to the current state of the TDMA schedule. Finding the maximum number of available slots on a path subject to the slot positions of the existing sessions is a NP-complete problem, even if global topology information is available. The authors propose distributed heuristics for available path bandwidth calculation and slot assignment.

In exchange for the more complex admission control, [58][57] operate in arbitrary topologies, while our approach currently supports rate enforcement for tree topologies.

However, [58][57] assume global slot synchronization. Due to the heuristic nature of the available path bandwidth calculation in [58][57], sessions that could be accepted are blocked, i.e. the network is underutilized. Underutilization is also unpredictable: given a set of session arrivals, the number of admitted sessions depends on the order of arrivals. For tree topologies, our approach will admit the maximum possible number of sessions irrespective of the order of session arrivals. However, the ability to admit more sessions comes with the penalty that some existing sessions may not receive their requested service while the TDMA schedule is reorganized to accommodate incoming sessions. Hence, provision of *continuous* CBR service requires a detailed experimental study of convergence delay under various traffic loads.

Although it would be interesting to experimentally compare the two approaches in terms of their strengths and weaknesses we will instead focus on end-to-end ABR–a service not currently supported for multi-hop wireless networks. According to ABR, arriving sessions do not have specific bandwidth requirements but agree to comply with what is available by the network. Such a setting necessitates provision of fair access. The approach of [58][57] cannot be applied in this case because it is specific to the path bandwidth calculation mechanism which is dependent on the current TDMA schedule.

6.2.2 Available Bit Rate (ABR) service

In the ABR framework, optimality is understood as allocating bandwidth to sessions in a max-min fair manner. For convenience and ease of illustration, we will use the fluid model to represent the sharing of bandwidth to the end-to-end sessions. A session normalized rate allocation $\boldsymbol{r} = (r_1, .., r_{|F|})$ is *feasible*, if for every session i, each link in the path $L(i)$ can support $\tau_i = \lfloor r_i \cdot T_{system} \rfloor$ slots, that is, the induced demand slot allocation on the network links is feasible. A feasible rate allocation is *max-min fair*

(MMF), if the rate of a session cannot be increased without decreasing the rate of another session of equal or lower rate. More formally, a feasible rate allocation $(r_1, .., r_{|F|})$ is MMF if it satisfies the following property with respect to another feasible rate allocation $(r_1^{'}, ..., r_{|F|}^{'})$: if there exists a session i such that $r_i < r_i'$, then there exists a j such that $r_j \leq r_i$ and $r_j^{'} < r_j$.

Determining feasibility of a session demand allocation requires determining feasibility of the corresponding link slot allocation. According to [53] this problem is NP complete for arbitrary topologies. Since MMF allocations are by definition feasible, finding or detecting them for arbitrary topologies becomes problematic. We will thus assume that only part of the overall network capacity is utilized for ABR, and seek the MMF rates with respect to this fraction. The fraction depends on the degree of topology control and is determined by the local feasibility conditions (written in terms of normalized rates by dividing both sides of (6.4) with T_{system}):

$$\sum_{i \in F(u)} \delta_i^{(u)} \cdot r_i \leq C_u^R, \ \forall u \in N \tag{6.7}$$

where $C_u^R = T_u^R / T_{system}$. Note that, for tree topologies, it is possible to compute the absolute MMF rates–eq. (6.7) captures the entire set of feasible allocations in this case.

We define node u to be a *bottleneck* for session i, if 1) u is fully utilized (with respect to C_u^R) and 2) session i has been allocated maximum rate over all sessions $F(u)$ sharing u. The definition of a bottleneck node yields a criterion for determining whether a given session allocation is MMF:

MMF criterion: A session rate allocation $\boldsymbol{r} = (r_1, ..., r_i, ..., r_{|F|})$ is MMF if and only if every session has at least one bottleneck node.

The session MMF rates can be computed using an iterative, off-line centralized al-

gorithm similar to the algorithm of Bertsekas and Gallager for wireline networks [110]. The modification must take into account that, in our case, the resources are nodes instead of links and that sessions in intermediate nodes need to consume twice the bandwidth than their allocated rate due to the slots needed at both incoming and outgoing links.

During each iteration of the centralized algorithm, each node divides its available bandwidth equally over the total number of sessions crossing its adjacent links. The bottlenecks of the current iteration are the nodes for which this division is minimum; the minimum ratio is the MMF rate for this iteration and is allocated to the sessions crossing the bottleneck nodes. We then remove the bottleneck nodes and their sessions from the network and reduce the available bandwidth of the remaining nodes by the amount consumed by the removed sessions (for each intermediate node in the path of each removed session, we must subtract twice the MMF rate from the node available bandwidth). Any node whose available bandwidth becomes zero is also removed. We then consider the next level bottleneck nodes of the reduced network and repeat the procedure. We continue until all sessions have been allocated their rates. The algorithm operation is described in Fig. 6.2.

We have implemented an asynchronous distributed version of the centralized algorithm. The distributed algorithm is similar in spirit to algorithms proposed for wireline ATM networks [113][114][118]. This is a rate-based approach for flow control where each source adjusts its transmission rate based on values seen in returning control packets, previously injected and circulated over the session path. The returning values are the most recent estimates of the session MMF rate as computed by all nodes in the session path.

Every node u maintains a subset $FC(u)$ of its sessions $F(u)$, currently seen as "constrained" by other nodes. It also maintains an estimate ϕ_u for the MMF rate it currently

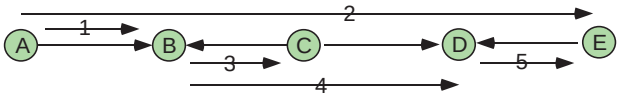


Figure 6.2: For ease of illustration, we compute the MMF session rates with respect to fractional capacities $C_u^R = C = 1 - \frac{2}{T_{system}}$. The MMF rate in the first iteration is $C/5$ (bottlenecks are B and C). Sessions 1,2,3,4 are allocated C/5 and they are removed from the network, along with bottleneck nodes B,C. Node A is also removed since all sessions crossing it have been removed. The bottleneck in the second iteration is node D providing all its remaining bandwidth ($2/5 \cdot C$) to session 5. The session MMF normalized rates are $(r_1, r_2, r_3, r_4, r_5) = (1/5, 1/5, 1/5, 1/5, 2/5) \cdot C$

provides to its unconstrained sessions. The MMF rate estimate ϕ_u is updated locally by procedure MMF_UpdateState(). The source or an intermediate node of a session invokes MMF_UpdateState() when a control packet is about to be sent to the downstream link (forward direction); the destination node invokes MMF_UpdateState() when a session control packet is about to be sent to the upstream link (reverse direction). In each case, when a control packet p of session i is about to be sent on link l, procedure MMF_UpdateState() at node u involves the following actions (Fig. 6.3):

Step 1: Node u updates the rate r_i of session i as the minimum of ϕ_u and the value in the rate field of packet p.

Step 2: The demand of link l is updated to reflect the change in r_i. If u is an intermediate node of session i, the demand of the other adjacent link k shared by i is updated in a simlilar fashion. The new link demand(s) are passed to the link scheduling algorithm.

Step 3: If ϕ_u is less than or equal to the value carried by the packet, it is copied to the packet rate field. In addition, a bit in the packet is set to indicate that the session

is constrained by a node in the path. Otherwise, session i is added to $FC(u)$ and the packet contents are not modified.

Step 4: Node u updates the MMF estimate ϕ_u by subtracting the bandwidth taken by the currently constrained sessions and equally dividing the rest of the bandwidth to the unconstrained sessions.

Step 5: The rates of some sessions in $FC(u)$ may be greater than the new ϕ_u. If this is the case, these sessions are removed from $FC(u)$ and step 4 is repeated. After the second iteration, it is guaranteed that no sessions in $FC(u)$ will have rate greater than ϕ_u.

Upon return of a control packet, the source adjusts the transmission rate according to the packet rate field. If the field indicates a value of r_i, the source adjusts its sending rate to $r_i \cdot R$ bps, where R is the maximum transmission rate of the radio in bps. The new control packets for session i are sent out with the packet rate field set to r_i and the constrained bit field set to zero.

Using arguments similar to those in [113], it can be proven that the asynchronous distributed algorithm converges in a finite number of iterations to the end-to-end MMF rate values. This holds for any topology form, given the appropriate fractional capacities C_u^R that ensure feasibility in each case. The main difference of the distributed algorithm with the wireline versions lies in the update of the MMF estimated rate that divides available rate of each node to its session parts (instead of sessions) and in that *every* node in the path–including the source and destination nodes–must update the MMF rate estimate. According to Step 2 of MMF_UpdateState() (Fig. 6.3), the demands of adjacent links are updated and passed to the link scheduling algorithm. Viewed globally, the end-to-end computation and link scheduling processes occur in parallel. The link scheduling is not aware of whether the end-to-end process is complete; it simply reacts

to the link demand updates. As soon as the end-to-end bandwidth allocation converges to the MMF rates, the link demands stabilize, allowing the link scheduling algorithm to converge.

6.3 Bluetooth Implementation

6.3.1 Design

Bluetooth [15] is a multi-channel asynchronous TDMA system with a special constraint that a node can be master to at most seven adjacent links. Channels are implemented as frequency hopping sequences, termed *piconets*. A Bluetooth ad hoc network is termed as a *scatternet*.

Figure 6.4 depicts the implementation of the end-to-end bandwidth allocation algorithm, the link scheduling algorithm and the coordination mechanism over the Bluetooth protocol stack. The Bluetooth Baseband layer operates according to the asynchronous TDMA scheme presented in Chapter 3. The Bluetooth Link Manager Protocol (LMP) is used for exchange of baseband control packets. Ideally, the link scheduling protocol and coordination mechanism would be implemented in the Baseband with the control packets being LMP messages. The current Bluetooth specification does not offer periodic scheduling at the Baseband layer. We have therefore implemented the link scheduling and coordination mechanisms in software, at the application layer.

We use the Bluetooth ”sniff mode” to instruct the Baseband to transmit according to the schedule maintained at the application layer. ”Sniff mode” is a low power mode where a slave can listen to a master for only a window of $N_{sniff\_attempt}$ slots within a period of T_{sniff} slots. Before entering sniff mode the nodes must agree on a slot offset within the period where they will commmunicate. The Bluetooth Host Controller

Procedure `MMF_UpdateState`

Update algorithm at node u for a control packet p of session i to be forwarded on link l

1 $r_i = \min(\phi_u, p.rate)$ /*update the session rate*/;

$\tau_i = \lfloor r_i \cdot T_{system} \rfloor$;

2 $\tau_l = \sum_{j \in S(l)} \tau_j$ /*update demand of link l*/;

if *($\delta_i^{(u)} == 2$) /*u is intermediate node of i*/* **then**

  $\tau_k = \sum_{j \in F(k)} \tau_j$ /*update demand of the other link k adjacent to u where session i belongs*/;

end

3 **if** *($\phi_u \leq p.rate$)* **then**

  $p.rate = \phi_u$; $p.constrained = 1$;

end

if *($\phi_u \geq p.rate$)* **then**

  $FC(u) = FC(u) \bigcup \{i\}$;

end

4 **if** *($|FC(u)| == |F(u)|$)* **then**

  $\phi_u = C_u^R - \sum_{j \in F(u)} r_j + \max_{j \in F(u)} r_j$;

else

  $\phi_u = \frac{C_u^R - \sum_{j \in FC(u)} \delta_j^{(u)} \cdot r_j}{\sum_{j \in F(u)} \delta_j^{(u)} - \sum_{j \in FC(u)} \delta_j^{(u)}}$;

end

5 **if** *exists j in $FC(u)$ such that $r_j \geq \phi_u$* **then**

  for *all j in $FC(u)$ such that $r_j \geq \phi_u$* **do**

    FC(u)=FC(u)-{j};

  end

  repeat step 4;

end

Figure 6.3: Update algorithm for session rate, link demands and MMF rate estimate ϕ_u.

Interface (HCI) exports a function where a node (either master or slave) can initiate sniff mode on a link. We can thus map T_{system} directly to T_{sniff}. Each node will impose different non-overlapping sniff windows to its neighbors. When, during the execution of the coordination mechanism, the local schedule of a node is modified at the application layer, we instruct the hardware to start sniff mode on link l on that offset by setting $N_{sniff\_attempt} = \tau_l + J_l^{(u)}$. Sniff mode has also been used in other approaches specifically targeted for scatternet scheduling [119][80].

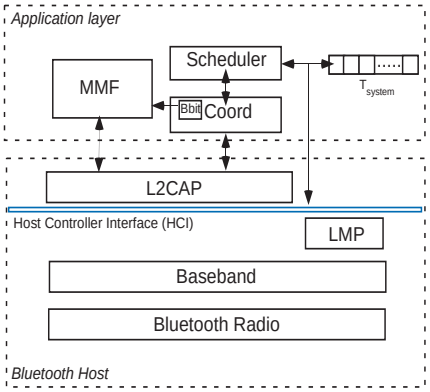


Figure 6.4: Implementation of the end-to-end bandwidth allocation framework over the Bluetooth stack

The Bluetooth L2CAP layer provides connection-oriented and connectionless services to upper layer protocols. It can support both unidirectional and bidirectional logical channels between two nodes. For the exchange of the link coordination mechanism control packets we use a bidirectional L2CAP channel. Each session consists of multiple L2CAP bidirectional channels, one for each link in the path. Thus, a session at an intermediate node is mapped on two L2CAP bidirectional channels, one to the upstream link and the other to the downstream link. Session data packets or control packets flowing in the forward direction are sent on the downstream L2CAP connection while session control packets returning to the source to the upstream L2CAP connection.

When a source receives feedback control packet with normalized rate r_i, it adjusts its transmission rate to $r_i \cdot B/D_{slot}$ bits/sec, where B/D_{slot} is the ratio of maximum payload bits per direction over the duration of a full-duplex slot. The Bluetooth baseband layer supports half-duplex slots of duration $0.625ms$. Each half-duplex slot can support up to $B = 216$ bits for payload data (Bluetooth DH1 packets). Slots can be combined in full duplex configurations of $(1,1)$, $(1,3)$, $(1,5)$ half duplex slots. In the experiments we use $(1,1)$ configuration. Thus, a full-duplex slot has a duration equal to $2 \cdot D_{slot} = 1.25ms$ and a maximum rate of $R = B/2D_{slot} = 172.8$ Kbps per direction can be supported.

In addition to rate adjustment at the sources, performance is enhanced by the use of a packet scheduler on every link l to select the type of packet that will be transmitted on a conflict-free slot. To expedite convergence of the link scheduling algorithm, link control packets are given highest priority. When the link control packet queue is empty, Weighted Round Robin (WRR) is used to share the bandwidth among the outgoing sessions on this link. When the demand of link l changes during the end-to-end algorithm execution (step 2 of MMF_UpdateState() in Fig. 6.3), the WRR weight $W_i(t)$ for each session i in the set of outgoing sessions $OUTS(l)$ of this link is updated as $W_i(t) = \frac{r_i(t)}{\min\limits_{j \in OUTS(l)} r_j(t)}$. A new WRR cycle is then constructed that will schedule sessions in proportion to their new relative weights. All WRR weights stabilize when all link demands stabilize; however, the target rates will actually be enforced when the link scheduling algorithm converges to the desired network TDMA link schedule.

6.3.2 Experiments

To test the system in complex configurations we use BlueHoc[120], the IBM Bluetooth extensions to the NS simulator [121]. We have further extended BlueHoc to support scatternets and the sniff mode. The link scheduling algorithm, the end-to-end MMF

algorithm and the coordination mechanism have been implemented as separate modules. We have performed experiments on various topology and session configurations. Here, we present and analyze a representative case.

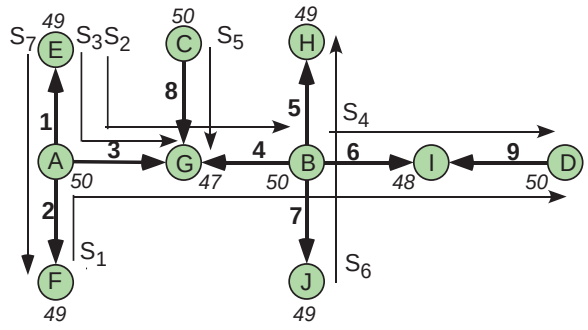


Figure 6.5: Arrows on links denote master-slave relationships. Italicized numbers on each node u denote $T_u^R = T_{system} - \sum_{l \in L(u)} J_l^{(u)}$, where T_{system} = 50 slots. The normalized capacities are $C_u^R = T_u^R / T_{system}$; the (normalized) MMF rates are $(r_{S_1}, .., r_{S_7}) = (0.125, 0.125, 0.125, 0.208, 0.315, 0.208, 0.125)$. These rates correspond to $(\tau_{S_1}, .., \tau_{S_7}) = (6, 6, 6, 10, 15, 10, 6)$ slots within $T_{system} = 50$ slots.

We consider the configuration shown in Fig. 6.5. A period of $T_{system} = 50$ slots is used. Nodes start with an arbitrary conflict-free TDMA schedule. This initial schedule is constructed when endpoints execute the link coordination mechanism to assign arbitrary slots on a newly established link; all sources start transmitting at maximum rate (172.8 Kbps) and subsequently adjust it based on the values of the received end-to-end control packets. Time is measured with respect to the time slot reference of the root node. Each simulation runs for 20000 slots (or $20000 \times 1.25ms = 25sec$).

Convergence delay is determined by D_S, the time until the link demands stabilize due to the end-to-end algorithm convergence, and D_L, the additional delay needed by the distributed link scheduling algorithm to converge to a TDMA schedule realizing these demands. The table in Fig. 6.6 includes D_S and D_L that resulted from different

	Convergence Delay (slots) and Overhead (%)				
Root	D_S	D_L	$D_S + D_L$	$O_S(\%)$	$O_L(\%)$
A	1523	469	1992	11.5	9.7
B	3145	178	3323	8.5	7.2
C	1995	282	2277	17.05	11.80
D	1718	733	2451	12.54	10.2
E	2529	196	2725	15.8	8.78
F	2765	327	3092	11.25	10.3
G	2836	392	3228	8.74	7.05
H	1943	436	2379	12.56	9.9
I	1982	361	2343	16.31	11.86
J	2225	543	2768	15.44	9.2

Figure 6.6: Convergence delay and control overhead in the configuration of Fig. 6.5 for different choices of the root node.

choices of the root node. Both delay components depend on the location of the root and the order with which the end-to-end algorithm satisfies the sessions–in increasing order of MMF rates. According to Fig. 6.5, sessions S_1, S_2, S_3 and S_7 first receive the lowest MMF rate (0.125) due to the first-level bottleneck node A. Then the MMF rates of S_5 (0.315) and S_4,S_6 (0.208) will be allocated by the second-level bottleneck nodes G and B, respectively.

In addition to the location of the root, the delay component D_S depends on the transient states of the TDMA schedule. During the TDMA schedule modifications, some links may be occasionally allocated a few slots. Since slots are shared by link control packets, as well as control and data packets of various sessions, this may delay the circulation of the control packets of some sessions and, consequently, increase the convergence delay of the end-to-end algorithm. This phenomenon was observed in the case of maximum D_S (3145 slots) where node B is the root. Link 3 was allocated 5 slots or less until slot 904; links 4 and 8 were constantly being rescheduled at the expense of link 3 during this period. Link 3 is in the control path of sessions S_1 and S_2, which belong to the set of sessions whose MMF rates must be computed first; this slowed down the end-to-end algorithm convergence. This behavior did not arise for all other root choices. The minimum D_S (1523 slots) was observed when the first-level bottleneck node A was selected as the root.

The delay component D_L depends on the order in which link demands have stabilized and the location of the root with respect to this order. The link demands stabilize in the order they are ”removed” (along with bottleneck nodes and sessions) during the end-to-end algorithm execution. In Fig. 6.5, the first demands to stabilize will be of links 1-4 because these links are crossed by the first-level sessions S_1,S_2,S_3 and S_7 (and only those sessions). Then links 5-9 will follow, due to the second-level sessions S_4-

S_6. This reasoning provides the order for demand stabilization among groups of links. The order within a group depends on the specific experiment run. According to Fig. 6.6, maximum D_L (733 slots) occurred when node D was selected as root. In this run, the last demand to stabilize (at slot $D_S = 1718$) was link 9, the only link adjacent to the root. We observed that, although at slot D_S the link demands at lower tree levels had already been stabilized and satisfied, the entire tree was rescheduled from the root downward. This worst-case global rescheduling did not occur for similar scenarios; for example, when node I was selected as root, it was adjacent to the slowest converging demand (link 6 at slot $D_S = 1982$) but in this run the tree was partially re-scheduled and convergence occurred within 361 slots. Incidentally, the minimum D_L was observed for the root being node B, the case that yielded maximum D_S. In all experiments D_L is much less than $10NT_{system} = 5000$ slots, the delay bound of the tree link scheduling algorithm.

Another quantity of interest during convergence is the control overhead, expressed as the fraction of slots used for control packet transmissions. The control overhead consists of the overhead due to the link coordination mechanism (SC_INFO, SC_UPD, SC_UPD_ACK, STABLE_REQ and STABLE_ACK packets) and the overhead due to the circulating end-to-end control packets. According to Fig. 6.6, the link control overhead is greater during D_S (maximum $O_S = 17.05\%$) because the link demands change constantly during this period. After the link demands stabilize, the link control overhead O_L is in the order of 10% on average. Link control overhead is dominated by the STABLE_REQ/STABLE_ACK packet exchanges: a node must request permission from its parent for each unstable child link it needs to reschedule. Similar to ATM networks, the end-to-end control overhead is regulated at the source by sending 1 control for every P data packets. The parameter P can be adjusted to trade-off increased speed of con-

vergence for increased overhead. In the experiments we use $P = 19$; this yields a fixed overhead of 5%.

After convergence, only end-to-end control overhead exists because the sources are never aware that the MMF rates have been reached. Constant flow of end-to-end control packets is needed for dynamic recomputation of the session MMF rates in presence of network dynamics (session additions and removals). Fig. 6.7 depicts session through-

	Rates (Kbps) and Delay (ms)				
Root	MMF	T	G	D_{avg}	d_{95}
S_1	20.73	20.73	19.69	10.65	$\pm$ 1.25
S_2	20.73	20.73	19.66	10.71	$\pm$ 1.22
S_3	20.73	20.73	19.66	10.69	$\pm$ 1.20
S_4	34.56	34.56	32.83	6.54	$\pm$ 0.73
S_5	51.84	51.84	49.24	4.284	$\pm$ 0.78
S_6	34.56	34.56	32.83	6.54	$\pm$ 0.73
S_7	20.73	20.73	19.68	10.63	$\pm$ 1.21

Figure 6.7: Session throughput (T), goodput(G) and average delay (D_{avg}) with 95% confidence intervals (d_{95}) for the configuration in Fig. 6.5, measured at each session destination after convergence.

put and goodput as well as average delay between data packet arrivals measured at the session destination after convergence. The throughput (goodput) of a session in bps is the number of bits due to data plus control packets (data packets only) the destination receives for this session from the time of convergence ($D_S + D_L$) until the end of the simulation run. The session throughputs exactly match the MMF rates; as expected, the goodput of every session is approximately 5% less than the throughput on account of the

end-to-end control overhead. Sessions within the same MMF group experience similar average delay (D_{avg}) within a small 95% confidence interval (d_{95}); this is due to the TDMA schedule periodicity and the WRR link schedulers employed over each session path.

6.4 Related work

Provision of end-to-end fairness via distributed link scheduling has not been addressed yet for wireless ad hoc networks. Weighted fairness, utility-based fairness and max-min fairness have been defined and addressed for single-hop sessions (links) in [105], [107] and [**?**], respectively. Distributed random access MAC protocols are used to coordinate transmissions–fairness can be realized only in a probabilistic sense. Tassiulas and Sarkar [108] define single-hop max-min fairness in slotted TDMA systems. A distributed token generation mechanism is used to compute the MMF rates for the network links. However, the scheduling needs both global topology information as well as a maximum weighted matching computation at every slot to enforce these rates. In a sequel paper [61], the token generation mechanism is coupled with back-pressure flow control to compute the max-min fair rates for end-to-end sessions. The centralized link scheduling component is still required.

The TDMA link scheduling in [108][61] can only achieve long-term fair rates because operation is considered in a slotted system of infinite horizon. Short term rate guarantees, as well as bounded delay (to the extent of the system period) can be achieved by periodic TDMA schedules. Our approach uses a fixed system period T_{system} combined with a set of local feasibility conditions. Another line of research has focused on link scheduling algorithms for systems of variable period[55][90][81]. These algorithms

try to reach a TDMA link schedule with short period realizing a given link demand slot allocation (finding the minimum period and TDMA schedule is an NP-complete problem). In [55], a centralized heuristic is introduced. A subsequent semi-centralized implementation [90] requires that information about each change in topology or demand to be distributed to all nodes; then each node uses an identical copy of the centralized algorithm [55] to compute the allocation on its adjacent links. This approach would not be practical for settings of frequent changes in topology or traffic demands. DSSA [81], a link scheduling algorithm for Bluetooth scatternets, uses only local information. Nodes start with knowledge of the demands of their adjacent links and use a heuristic to construct a TDMA link schedule realizing the demands in a short period. However, the algorithm termination is not distributed; this renders implementation harder in dynamic settings. In addition to the difficulties specific to each algorithm of [55][90][81], variable-period systems by nature need to suspend communications while a new TDMA schedule is computed in response to changes in topology or traffic demands. Once the computation is complete, a signal must be broadcast to the entire network before operation is resumed.

Distributed TDMA scheduling schemes of lower complexity have been proposed for both slot-synchronized [35][122] and asynchronous TDMA systems [78][79][80]. These schemes are dynamic but do not target specific link demand allocations. Hence, they cannot be used for provision of deterministic bandwidth guarantees to links or end-to-end sessions.

6.5 Conclusions

We presented a framework where end-to-end bandwidth allocation algorithms currently available for wireline networks can be used with certain modifications for wireless ad hoc networks if we can find a set of appropriate local feasibility conditions as well as an underlying distributed, self-stabilizing link scheduling algorithm. The link scheduling is based on an asynchronous TDMA protocol that does not rely on global slot synchronization nor knowledge of the number of nodes in the network.

Using this framework, we proposed an algorithm for provision of end-to-end ABR service to multi-hop sessions. This algorithm can operate for any topology and compute the session MMF rates with respect to a fraction of the network capacity provided by the local feasibility conditions. We showed that, in the case of tree topologies, the network can be fully utilized and a link scheduling algorithm that can enforce the computed end-to-end rates exists. We presented an implementation of this framework over Bluetooth, an existing asynchronous TDMA wireless technology.

A natural extension for the link scheduling component of the framework is the design of converging algorithms that provide rate enforcement in more general topologies than trees (at the inevitable expense of reduced utilization). Such algorithms are the subject of our future research efforts.

Chapter Appendix 6.A—Pseudocodes of Procedure SampleReschedule() and algorithm STABLETREE

```
Procedure SampleReschedule
    begin
SR-1    PrioritizeLinks();
SR-2    l_c = GetMaxUnstableChildLink();
        if (l_c ≠ −1) then
            if (busybit_==0 AND BusyBit(v)==0) then
SR-3            busybit=1;
SR-4            send SC_INFO packet to v;
            end
        end
    end
```

```
Procedure PrioritizeLinks
    Assign priorities to children links in order of appearance after slot e_{l_p}^{(u)}
    local    : CH = set of children links, LINKSET, p, slot
    begin
        p = |CH|; LINKSET = CH; slot = e_{l_p}^{(u)} ⊕ 1;
        repeat
            l_c = local_schedule[slot];
            if (l_c ∈ LINKSET) then
                p_{l_c} = p /*set the priority of l_c to p*/;
                p=p-1;
                LINKSET = LINKSET - {l_c} ;
            end
            slot = slot ⊕ 1;
        until LINKSET is empty;
    end
```

```
Function GetMaxUnstableChildLink
    Return the maximum priority unstable child link or -1 otherwise
    local    : CH = set of my children links, J_k equals 1 if I am slave on child link k and zero otherwise
    begin
        for p=|CH| down to 1 do
            l_c = the child link of priority p;
            if (not satisfied(l_c)) then
                return l_c;
            else
                lpsum = Σ_{k∈CH:p_k<p_l} (τ_k + J_k);
                if (lpsum > |[e_{l_c} ⊕ 1, s_{l_p} ⊖ 1]|)) then
                    return l_c;
                end
            end
        end
        return -1;
    end
```

Figure 6.8: Procedure SampleReschedule()

Algorithm 1: STABLETREE

```
     Data    : Asynchronous events at node u  Parent node(link): p(l_p) (or none if root), Child node
               (link): c (l_c)
     Result  : Corresponding actions
E1   Events:   e1: Any adjacent link becomes non-satisfied;
               OR e2: Scheduling of a link just completed;
     begin
         if (event e2 occured) then
E1-1         busybit_=0;
         end
         if (busybit_==0) then
             if (I am root) then
                 SampleReschedule();
             else
                 if (wait_parent_==0)) then
E1-2                 wait_parent_=1;
E1-3                 send STABLE_REQ packet to parent p;
                 end
             end
         end
     end
E2   Event: STABLE_REQ packet received from child c;
     begin
         if (I am root OR satisfied(l_p)) then
             if (stable(l_c)) then
E2-1             send STABLE_ACK packet to child c;
             else
                 if (busybit_==0 AND wait_parent_==0) then
E2-2                 SampleReschedule();
                 end
             end
         end
     end
E3   Event: STABLE_ACK packet received from parent p;
     begin
E3-1     wait_parent_=0;
E3-2     SampleReschedule();
     end
E4   Event: SC_INFO packet received from node v;
     begin
         if (I am child of v) then
E4-1         busybit_=1;
E4-2         wait_parent_=0;
E4-3         send SC_INFO packet to v;
         else
E4-4         AssignSlots(l_v) /*Determine new slot positions for l_v*/ ;
E4-5         Initiate distributed coordination mechanism by updating v and affected neighbors with
             SC_UPD packets.
         end
     end
```

Figure 6.9: The asynchronous distributed link scheduling algorithm

Chapter 7

Conclusions

The goal of this dissertation was to address two fundamental issues that arise in wireless ad hoc networks: topology organization and transmission scheduling for provision of QoS guarantees. Both were viewed as resource allocation problems where nodes need to reach a global optimality objective using only local information.

In the topology organization problem, topology control was addressed jointly with neighborhood discovery. The symmetric link establishment protocol provided insights for neighborhood discovery mechanics in frequency hopping systems in general and Bluetooth in particular. The channel participation constraints in topology control and the incorporation of network formation delay as an additional performance objective, were unique contributions of this dissertation.

In the transmission scheduling problem we introduced a novel distributed TDMA framework for the realization of various link-level and end-to-end QoS objectives. The fundamental starting point was the recognition that link demands will be generated locally by the nodes. This led to the need for capturing a subset of the (globally) feasible allocations using local conditions. The general bound for arbitrary topologies and the optimal bound for trees on the asynchronicity overhead, derived in Chapters 3 and 4,

respectively, allowed asynchronous TDMA to be naturally incorporated in this framework.

Another fundamental contribution was the fluid distributed algorithm that operates in this well-defined subset of feasible allocations and guides slot reassignments towards the desired objective. Although we could not analytically prove that application of the fluid algorithm to the slotted system converges to the optimal TDMA schedule, simulations for both static and mobile networks demonstrated excellent performance.

The local feasibility conditions and the fluid model allowed viewing the provision of end-to-end QoS as interaction of two separate mechanisms that operate in parallel: a QoS-aware end-to-end bandwidth allocation algorithm operating at the fluid level and computing the optimal session rates and a dynamic link scheduling algorithm that receives feasible link demands and attempts to converge a TDMA schedule realizing the computed rates. This logical separation allows realization of various end-to-end QoS objectives by modiying existing algorithms for wireline networks. The dynamic link scheduling problem was solved for tree networks; however, the end-to-end algorithms can compute the optimal rates for any topology when coupled with the identified feasibility conditions.

Finally, an important common feature of both topology organization and transmission scheduling approaches is that, due to their low complexity and lack of restrictive assumptions, they can be implemented even in current low-end wireless technologies such as Bluetooth.

The next section, outlines the contributions of this dissertation. Section 7.2 discusses some open problems for further study.

7.1 Contributions

Chapter 2:

- A randomized protocol for symmetric discovery and link establishment in frequency hopping systems (two-node case).
- A distributed topology construction protocol for multi-channel frequency hopping systems with channel participation constraints (multiple-node case).

Chapter 3:

- Introduction of the asynchronous TDMA communication model.
- A scheduling algorithm that minimizes asynchronicity overhead for a specific ordering of link activations in the reference synchronized schedule. The overhead of this algorithm never exceeds the period of the reference schedule.
- A heuristic approach to find the minimum-overhead ordering of link activations.

Chapter 4:

- A distributed asynchronous TDMA protocol for multi-channel ad hoc networks that does not rely on any assumptions such as network-wide slot synchronization/enumeration or global topology knowledge.
- Derivation of sufficient local feasibility conditions that depend on existence (or lack thereof) of a topology control algorithm and global slot synchronization.
- An algorithm for optimal scheduling in asynchronous TDMA tree networks: in this case, the entire set of feasible allocations can be captured by local conditions.

Chapter 5:

- A distributed fluid algorithm for computing link MMF rates in an ad hoc network and a slot assignment algorithm aiming their enforcement.
- Extensive performance evaluation for large networks in both static and dynamic settings.

Chapter 6:

- A two-stage framework for provision of end-to-end rate guarantees:
  - Fluid-based end-to-end bandwidth allocation algorithm computes optimal session rates according to a QoS objective.
  - Distributed link scheduling algorithm computing a TDMA schedule realizing these rates.
- Distributed admission control mechanism and MMF rate computation algorithm for realizing CBR and ABR end-to-end QoS objectives, respectively.
- A distributed dynamic link scheduling algorithm for enforcing any set of link rates in tree-structured asynchronous TDMA ad hoc networks.
- Implementation and performance evaluation of the transmission scheduling framework over Bluetooth.

7.2 Suggestions for future work

Throughout the dissertation, we have tried to provide directions toward which our work can be extended. In this section, we will elaborate on a few we believe are most interesting.

7.2.1 Topology organization

In the topology organization problem, the lack of initial proximity information necessitates approaches that are incremental in addition to being distributed. While we addressed this complexity when all nodes are initially within range of each other, the problem needs to be studied extensively for the multi-hop case. In addition, adjusting transmission powers in addition to channel assignments for performing topology control is another interesting research direction to pursue.

7.2.2 Transmission scheduling

In the transmission scheduling problem, it was evident that realization of both link-level or end-to-end QoS objectives relies on the solution of a dynamic link scheduling problem. While this problem was solved for the case of tree topologies, a converging algorithm for arbitrary topologies remains an open issue. A possible approach would be to aim for an approximate solution and use an extension of the fluid MMF algorithm of Chapter 5 to guide the slot reassignments. Such an algorithm could be realized in the context of a generalized link-level MMF model that includes minimum rate requirements for each link. In this model, link demands generated by the end-to-end algorithm would be viewed as minimum link rate requirements; the remaining bandwidth would be shared to the links in a MMF manner. In addition to the generalized link MMF fluid model, various algorithms for the decisions of slot reassignments during a link rate adjustment should be investigated for both synchronized and asynchronous TDMA networks. For example, a slot assignment algorithm that uses two-hop information might yield better convergence properties at the expense of increased complexity.

Our approach for the transmission scheduling problem can be summarized by the following steps:

1. Find the minimum or an upper bound on the TDMA schedule length realizing a set of local slot demands in the network. This bound will determine the local feasibility conditions.

2. Define a (local or end-to-end) QoS objective and find a distributed fluid bandwidth allocation algorithm.

3. Find a dynamic TDMA scheduling algorithm that realizes the local slot demands resulting from the fluid algorithm.

4. Design a distributed TDMA protocol that keeps the network free of transmission conflicts during the slot reassignments.

While these steps were applied to a single-transceiver/multi-channel ad hoc network carrying point-to-point traffic, they also provide a flexible framework for a systematic treatment of distributed transmission scheduling in a wide variety of wireless settings:

- **Single-channel systems:** Single-channel systems carrying point-to-point traffic suffer from secondary interference. A typical example is the hidden-terminal problem that arises in 802.11-based wireless ad hoc networks. Similar to the multi-channel case, finding the minimum length TDMA link schedule is a NP-complete problem [52]; contrary to the multi-channel case, no known upper bounds on the schedule length translatable to local conditions exist, even for slot-synchronized systems. However, for asynchronous TDMA, our general bound on the asynchronicity overhead holds irrespective of the interference constraints.

- **Multi-transceiver systems:** Network throughput can be increased if each node is equipped with multiple communication transceivers. In such a system, each node u can simultaneously communicate (transmit or receive) with a number of

neighbors equal to $\alpha(u)$, its transceiver count. In absence of secondary interference (multi-channel system) and global slot synchronization, Choi and Hakimi have established that if $\alpha(u)$ is even for every node u, the minimum schedule length for a given link slot demand allocation $\boldsymbol{\tau}$ is $L_{min}(\boldsymbol{\tau}) = \max_u \sum_{l \in L(u)} \tau_l / \alpha(u)$ [123]. Hence, the local feasibility conditions multi-channel/multi-transceiver system with even number of transceivers per node will be $\sum_{l \in L(u)} \tau_l \leq \alpha(u) \cdot T_{system}$. Given these conditions, the fluid model developed in this dissertation can be used unmodified to compute link or end-to-end MMF rates. However, enforcing the rates using a slotted schedule may require more sophisticated slot assignment algorithms.

- **Multicast traffic model:** In many envisioned applications an ad hoc network is spontaneously formed when a need for collaborative action exists. Such higher-layer group communications can be captured by the multicast traffic model where each packet transmission is destined to a subset of neighbors. Contrary to point-to-point traffic, here the broadcast nature of the wireless medium is needed to increase performance–while multicasting can be implemented using multiple point-to-point transmissions, multicast scheduling can achieve the same allocations using fewer transmissions. For the multicast traffic model, neither NP-completeness of the optimal scheduling problem nor any upper bounds on the schedule length are known to date.

Summarizing, in the single-channel system and the multicast traffic model cases, optimal centralized offline scheduling (step 1) has yet to be addressed; in multi-transceiver ad hoc networks appropriate bounds on the schedule length do exist under certain conditions. In every case, if local feasibility conditions are determined from step 1, steps 2-4 can be used to generate distributed online scheduling algorithms aiming for various

QoS objectives.

7.2.3 Transmission scheduling and topology discovery

Our distributed TDMA architecture assumes that neighborhood discovery is performed using a separate transceiver and channel. In single-transceiver networks a node will need to coordinate transmissions in the discovery and communication channels. Hence discovery can be seen as part of the transmission scheduling process where discovery and communication become conflicting objectives: if many slots are assigned for discovery, communication throughput decreases. On the other hand, if the discovery channel is used only a very small fraction of time, then topology discovery may not be effective. The investigation of this trade-off in a mobile network setting is another challenging open research issue.

BIBLIOGRAPHY

[1] R.E. Kahn. The organization of computer resources into a packet radio network. 25:169–178, January 1977.

[2] J. Jubin and J.D. Tornow. The darpa packet radio network protocols. *Proc. IEEE*, 75:21–32, January 1987.

[3] D. Beyer. Accomplishments of the darpa suran program. In *Proc. MILCOM*, Monterey, CA, October 1990.

[4] P. Sass. Communications networks for the force xxi digitized battlefield. *Proc. ACM/Baltzer Mobile Networks and Applications Journal (Special Issue, Mobile Ad Hoc Networking)*, 4, Octiber 1999.

[5] J. Strater and B. Wollman. Ospf modeling and test results and recommendations. *Mitre technical report, Xerox Office Products Division*, 96W0000017:http://www.isr.umd.edu/TechReports, 1996.

[6] J.J. Garcia-Luna-Aceves. Wireless internet gateways (wings). In *Proc. MILCOM*, Monterey, CA, November 1997.

[7] M. Steenstrup. Cluster based networks. *Ad Hoc Networking, Addison Wesley*.

[8] B. Leiner, R. Ruth, and A. Sastry. Goals and challenges of the darpa glomo program. *IEEE Personal Communications*, pages 34–43, December 1996.

[9] A. Ephremides, J.E. Wieselthier, and D.J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proc. of IEEE,* 75(1):56–73, 1987.

[10] M. Frankel. Tactical c3 for the ground forces, telecommunications and processing for military command and control: An architecture for the 21st century. *AFCEA International Press*, 1986.

[11] MeshNetworks Inc. Fostering disruptive technologies. In *www.meshnetworks.com*, Maitland, FL, USA, January 2002.

[12] Moteran Systems. Moteran systems: True network mobility. In *www.moteran.com*, Germany, January 2003.

[13] Inc Arcwave. Meshcast urban and suburban network simulations. In *http://www.cowave.com/*, Campbell, CA, USA, January 2003.

[14] Vista Broadband Networks. Vista broadband: Connecting communities. In *http://www.vbbn.com/d6.html*, Santa Rosa, CA, USA, January 2003.

[15] Bluetooth Special Interest Group. Specification of the bluetooth system, version 1.2. In *www.bluetooth.com*.

[16] IEEE 802.15 Working Group for WPAN. Wpan specification. In *http://www.vbbn.com/d6.html*, http://grouper.ieee.org/groups/802/15/, January 2002.

[17] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43:551–558, May 2000.

[18] Inc. Ember Communications. Reliable wireless networks for industrial systems. In *www.ember.com*, Boston, MA, January 2003.

[19] R. Ramanathan and R. Hain. Topology control of multihop radio networks using transmit power adjustment. In *Proc. IEEE INFOCOM*, Tel Aviv, Israel, March 2000.

[20] H. Takagi and L. Kleinrock. Optimal transmission ranges for randomly distributed packet radio terminals. 32(1):246–257, 1984.

[21] J. Wieselthier, G.D. Nguyen, and A. Ephremides. On the construction of energy-efficient broadcast and multicast trees in wireless networks. In *Proc. IEEE INFOCOM*, Tel Aviv, Israel, April 2000.

[22] L. Hu. Topology control for multihop packet radio networks. 41(10), October 1993.

[23] L. Hu. Distributed code assignments for cdma packet radio network. *Proc. IEEE Transactions on Networking*, 1(6), 1993.

[24] T. Salonidis, P. Bhagwat, L. Tassiulas, and R.O. LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[25] G.V. Záruba, S. Basagni, and I. Chlamtac. Bluetrees - scatternet formation to enable bluetooth-based ad hoc networks. In *Proc. International Conference on Computer Communications (ICC)*, St. Petersburg, Russia, June 2001.

[26] C. Law, A. Mehta, and K. Siu. Performance of a new bluetooth scatternet formation protocol. In *Proc. ACM MOBIHOC*, Long Beach, CA, October 2001.

[27] Z. Wang, R.J. Thomas, and Z.J. Haas. Bluenet-a new scatternet formation scheme. In *Proc. HICSS*, Big Island, Hawaii, January 2002.

[28] C. Petrioli and S. Basagni. Degree-constrained multihop scatternet formation for bluetooth networks. In *Proc. GLOBECOM*, Taipei, Taiwan, November 2002.

[29] G. Tan, A. Miu, J. Guttag, and H. Balakrishnan. An efficient scatternet formation algorithm for dynamic environments. In *Proc. IASTED Communications and Computer Networks (CCN)*, Cambridge, MA, November 2002.

[30] X. Li and I. Stojmenovic. Partial delaunay triangulation and degree-limited localized bluetooth scatternet formation. In *Proc. AD-HOC NetwOrks and Wireless (ADHOC-NOW)*, pages 17–32, Toronto, Canada, September 2002.

[31] R. Guerin, J. Rank, S. Sarkar, and E. Vergetis. Forming connected topologies in bluetooth adhoc networks. In *Proc. International Teletraffic Congress (ITC)*, Berlin, Germany, September 2003.

[32] V. Bharghavan R. Sivakumar, B. Das. Spine routing in ad hoc networks. *ACM/Baltzer Publications Cluster Computing Journal, Special Issue on Mobile Computing*, 3, June 1998.

[33] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen. Scalable routing strategies for ad hoc wireless networks. *Proc. IEEE Journal on Selected Areas in Communications*, 3:1369–1379, August 1999.

[34] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality of service support. *Proc. ACM Baltzer Mobile Networks and Applications*, 3:101–119, June 1998.

[35] D.J. Baker and A. Ephremides. The architectural organization of a packet radio network via a distributed algorithm. *Proc. IEEE Transactions on Communications*, 29:1694–1701, 1981.

[36] M. Gerla and T. Tsai. Multicluster, mobile multimedia radio network. *Proc. ACM Baltzer Journal of Wireless Networks*, 1:255–65, August 1995.

[37] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proc. IEEE Vehicular Technology Conference (VTC)*, Amsterdam, The Netherlands, September 1999.

[38] L. Bao and J.J. Garcia-Luna-Aceves. Topology management in ad hoc networks. In *Proc. ACM MOBIHOC*, Annapolis, MD, USA, June 2003.

[39] S. Banerjee and S. Khuller. A clustering scheme for hierarchical control in multi-hop wireless networks. In *Proc. IEEE INFOCOM*, Anchorage, ALASKA, USA, April 2001.

[40] A. Amis, R. Prakash, D. Huynh, and T. Vuong. Max-min d-cluster formation in wireless ad hoc networks. In *Proc. IEEE INFOCOM*, pages 32–41, 2000.

[41] T. Salonidis, P. Bhagwat, and L. Tassiulas. Proximity awareness and fast connection establishment in bluetooth. In *Proc. ACM MOBIHOC*, Boston, MA, August 2000.

[42] T. Salonidis, P. Bhagwat, L. Tassiulas, and R.O. LaMaire. Distributed topology construction of bluetooth personal area networks. In *Proc. IEEE INFOCOM*, Anchorage, ALASKA, USA, April 2001.

[43] G. Alonso, E. Kranakis, R. Wattenhofer, and P. Widmayer. Probabilistic protocols for node discovery in ad-hoc, single broadcast channel networks. In *In WMAN*

(workshop on Wireless Mobile Adhoc Networks), IPDPS, Nice, France, April 2003.

[44] G. Alonso, C. Sawchuk, E. Kranakis, R. Wattenhofer, and P. Widmayer. Randomized protocols for node discovery in ad-hoc multichannel broadcast networks. In *Proc. ADHOCNOW*, Montreal, Canada, April 2003.

[45] C. Law, A. Mehta, and K. Siu. Performance of a new bluetooth scatternet formation protocol. In *Proc. ACM MOBIHOC*, Long Beach, CA, USA, October 2001.

[46] A. Kumar and R. Gupta. Capacity evaluation of frequency hopping based ad-hoc systems. In *Proc. ACM SIGMETRICS*, Cambridge, MA, June 2001.

[47] J.J. Garcia-Luna-Aceves and J. Raju. Distributed assignment of codes for multihop packet radio networks. In *Proc. MILCOM*, Monterey, CA, USA, October 1997.

[48] N. Abramson. The aloha system. *eds. N.Abramson and F. Kuo, Computer-Communication Networks, Prentice Hall, Englewood Cliffs, NJ, 1973*.

[49] L. Kleinrock and F.A. Tobagi. Packet switching in radio channels i. carrier sense multiple-access modes and their throughput-delay characteristics. *Proc. IEEE Transactions on Communications*, 23, December 1975.

[50] IEEE. Wireless lan, medium access control (mac) and physical layer (phy) specifications. In *www.ieee.org*.

[51] A. Nasipuri and S. Das. A multichannel csma mac protocol for multihop wireless networks. In *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, New Orleans, September 1999.

[52] E. Arikan. Some complexity results about packet radio networks. *Proc. IEEE Transactions on Information Theory*, 30:681–685, July 1984.

[53] I. Holyer. The np-completeness of edge coloring. *Proc. SIAM Journal of Computing*, 10:169–197, 1981.

[54] J. Silvester. Perfect scheduling in multihop broadcast networks. In *Proc. International Conference on Computer Communications (ICC)*, London, England, Sepmteber 1982.

[55] M. Post, P. Sarachik, and A. Kershenbaum. A biased greedy algorithm for scheduling multihop radio networks. In *Proc. Annual Conference on Information Sciences and Systems (CISS)*, Johns Hopkins Univ., March 1985.

[56] S. Ramanathan. A unified framework and algorithm for channel assignment in wireless networks. In *Proc. IEEE INFOCOM*, Kobe, Japan, September 1997.

[57] C. Zhu and M.S. Corson. Qos routing for mobile ad hoc networks. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[58] C.R. Lin. On-demand qos routing in multihop mobile networks. In *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

[59] S. Chen and C. Nahrstedt. Distributed quality of service routing in ad hoc networks. *Proc. IEEE Journal on Selected Areas in Communications*, 17, August 1999.

[60] S. Lee and A. Campbell. Insignia: In-band signaling support for qos in mobile ad hoc networks. In *Proc. International Workshop on Mobile Multimedia Communications (MoMuC)*, Berlin, Germany, October 1998.

[61] S. Sarkar and L. Tassiulas. End-to-end bandwidth guarantees through fair local spectrum share in wireless ad-hoc networks. In *Control and Decision Conference (CDC)*, Maui, HI, USA, December 2003.

[62] C. Zhu and M.S. Corson. A five-phase reservation protocol (fprp) for mobile ad hoc networks. *Wireless Networks*, 7:371–384, October 1989.

[63] T. Salonidis, P. Bhagwat, L. Tassiulas, and R.O. LaMaire. Distributed topology construction of bluetooth wireless personal area networks. *Proc. IEEE Journal on Selected Areas in Communications (JSAC)–Special Issue on Wireless Ad Hoc Networks*, January 2005 (to appear).

[64] T. Salonidis and L. Tassiulas. Asynchronous tdma ad hoc networks: Scheduling and performance. *Proc. European Transactions In Telecommunications (ETT)*, 3, May-June 2004.

[65] T. Salonidis and L. Tassiulas. Distributed on-line schedule adaptation for balanced slot allocation in wireless ad hoc networks. In *Proc. IEEE International Workshop on Quality of Service (IWQoS)*, Montreal,Canada, June 2004.

[66] T. Salonidis and L. Tassiulas. Distributed dynamic scheduling for end-to-end rate guarantees in wireless ad hoc networks. *Technical Report, Institute of Systems Research (ISR), University of Maryland*, TR 2004-7:http://www.isr.umd.edu/TechReports, 2001.

[67] T.G. Robertazzi and P.E. Sarachik. Self-organizing communication networks. *IEEE Communications Magazine*, 24(1), 1986.

[68] C.R Lin and M. Gerla. Adaptive clustering for mobile wireless networks. *Proc. IEEE Journal on Selected Areas in Communications*, 15(7):1265–1275, 1997.

[69] A.K. Parekh. Selecting routers in ad hoc wireless networks. In *SBT/IEEE International Telecommunications Symposium*, Acapulco, Mexico, August 1994.

[70] IEEE 802.15 Working Group for WPAN. Ieee 802.15 std. specification, 2002. In *http://grouper.ieee.org/groups/802/15/*.

[71] K. Trivedi. *Probability and Statistics with reliability queuing and Computer Science Applications*.

[72] M. Marsan, C. Chiasserini, A. Nucci, and G. Carello. Optimizing the topology of bluetooth wireless personal area networks. In *Proc. IEEE INFOCOM*, New York City, NY, June 2002.

[73] G. Miklos, Z. Turanyi, A. Valko, and P. Johansson. Performance aspects of bluetooth scatternet formation. In *Proc. ACM MOBIHOC*, Boston, MA, August 2000.

[74] T. Salonidis and L. Tassiulas. Performance issues of bluetooth scatternets and other asynchronous tdma ad hoc networks. In *Proc. International Workshop on Mobile Multimedia Communications (MoMuC)*, Munich, Germany, October 2003.

[75] J. Haartsen. Bluetooth baseband specification, version 1.1. In *www.bluetooth.com*.

[76] K. Fleming. Bluetooth host controller interface functional specification, version 1.1. In *www.bluetooth.com*.

[77] I.A. Cimet C. Cheng and P.R. Kumar. A protocol to maintain a minimum spanning tree in a dynamic topology. In *Proc. ACM SIGCOMM*, Stanford, CA, August 1988.

[78] N. Johansson, F. Alriksson, and U. Jonsson. Jump mode - a dynamic window-based scheduling framework for bluetooth scatternets. In *Proc. ACM MOBIHOC*, Long Beach CA, October 2001.

[79] A. Racz, G. Miklos, F. Kubinszky, and A. Valko. A pseudo random coordinated scheduling algorithm for bluetooth scatternets. In *Proc. ACM MOBIHOC*, Long Beach CA, October 2001.

[80] S. Baatz, M. Frank, C. Kuhl, P. Martini, and C. Scholz. Bluetooth scatternets: An enhanced adaptive scheduling scheme. In *Proc. IEEE INFOCOM*, New York, NY, June 2002.

[81] U. Korner N. Johansson and L. Tassiulas. A distributed scheduling algorithm for a bluetooth scatternet. In *Proc. International Teletraffic Congress (ITC)*, Salvador da Bahia, Brazil, September 2001.

[82] L. Tassiulas. Scheduling packet radio networks, M.Sc. Thesis, May 1990.

[83] B. Hajek and G. Sasaki. Link scheduling in polynomial time. *Proc. IEEE Transactions on Information Theory*, 34:910–917, September 1988.

[84] A. Ephremides and T.V. Truong. Scheduling broadcasts in multihop radio networks. *Proc. IEEE Transactions on Communications*, 38, April 1990.

[85] R. Ramaswami and K. Parhi. Distributed scheduling of broadcasts in a radio network. In *Proc. IEEE INFOCOM*, Ottawa, Ontario, Canada, April 1989.

[86] S. Ramanathan and E. Lloyd. Scheduling algorithms for multihop radio networks. *Proc. IEEE Transactions on Networking*, 1:166–177, April 1993.

[87] N. Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters*.

[88] T. Salonidis and L. Tassiulas. Distributed on-line schedule adaptation for balanced slot allocation in bluetooth scatternets and other ad hoc network architectures. *Technical Report, Institute of Systems Research (ISR), University of Maryland*, TR 2001-24:http://www.isr.umd.edu/TechReports, 2002.

[89] L. Tassiulas and S. Sarkar. Maxmin fair scheduling in wireless networks. In *Proc. IEEE INFOCOM*, New York, NY, USA, October 2002.

[90] M. Post, A. Kershenbaum, and P. Sarachik. A distributed evolutionary algorithm for reorganizing network communications. In *Proc. MILCOM*, Boston, MA, October 1985.

[91] D.J. Baker, J. Wieselthier, and A. Ephremides. A distributed algorithm for scheduling the activation of links in a self-organizing, mobile, radio network. In *Proc. International Conference on Computer Communications (ICC)*, London, UK, June 1982.

[92] D.J. Baker, J. Wieselthier, and A. Ephremides. A channel access protocol for survivable radio networks with frequency hopping spread spectrum signalling. In *Proc. Annual Allerton Conference On Communication, Control and Computing*, pages 50–59, Allerton, IL, October 1984.

[93] I. Chlamtac and A. Lerner. Fair algorithms for maximal link activation in multihop radio networks. *Proc. IEEE Transactions on Communications*, 35, July 1987.

[94] I. Cidon and M. Sidi. Distributed assignment algorithms for multihop packet radio networks. *Proc. IEEE Transactions on Computers*, 38:1353–1361, October 1989.

[95] A. Farago I. Chlamtac and H. Zhang. Time-spread multiple-access (tsma) protocols for multihop mobile radio networks. *Proc. IEEE Transactions on Networking*, 6:804–812, December 1997.

[96] J.H. Ju and V.O.K. Li. An optimal topology-transparent scheduling method in multihop packet radio networks. *Proc. IEEE Transactions on Networking*, 6:298–306, June 1998.

[97] R. Krishnan and J. Sterbenz. An evaluation of the tsma protocol as a control channel mechanism in mmwn. *Technical report, BBN Technical Memorandum*, 1279, 2000.

[98] I. Chlamtac and S. Pinter. Distributed nodes organization algorithm for channel access in a multihop dynamic radio network. *Proc. IEEE Transactions on Computers*, 36, June 1987.

[99] J. Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *In Proc. Journal of Research National Bureau of Standards*, 69(B), 1965.

[100] C. Shannon. A theorem on coulouring lines of a network. *J. Math. Phys.*, 39:148–151, 1948.

[101] V. Bhatnagar and G. Kesidis. Bluetooth scatternet formation using proximity information of an election protocol. In *Proc. joint IEEE Int. Conf. on Networking and IEEE Int. Conf. on Wireless LANs and Home Networks*, Atlanta, GA, August 2002.

[102] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proc. ACM MOBICOM*, Boston, MA, USA, 2000.

[103] W. Zhang and G. Cao. Optimizing tree reconfiguration for mobile target tracking in sensor networks. In *Proc. IEEE INFOCOM*, Hong Kong, March 2004.

[104] M. Cagalj, J.P. Hubaux, and C. Enz. Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues. In *Proc. ACM MOBICOM*, Atlanta, GA, September 2002.

[105] S. Lu H. Luo and V. Bharghavan. A new model for packet scheduling in multihop wireless neworks. In *Proc. ACM MOBICOM*, Boston, MA, USA, August 2000.

[106] X. Huang and B. Bensaou. On max-min fairness and scheduling in wireless ad-hoc networks: Analytical framework and implementation. In *Proc. ACM MOBIHOC*, Long Beach, CA, USA, October 2001.

[107] T. Nandagopal, T. Kim, X. Gao, and V. Bharghavan. Achieving mac layer fairness in wireless packet networks. In *Proc. ACM MOBICOM*, Boston, MA, USA, October 2000.

[108] L. Tassiulas and S. Sarkar. Maxmin fair scheduling in wireless networks. In *Proc. IEEE INFOCOM*, New York, NY, USA, June 2002.

[109] K. Nahrstedt Y. Xue, B. Li. Price-based resource allocation in wireless ad hoc networks. In *Proc. 11th International Workshop on Quality of Service (IWQoS)*, Monterey, CA, USA, June 2003.

[110] D. Bertsekas and R. Gallager. *Data networks*.

[111] M.S. Corson V. Park. A performance comparison of the temporally-ordered routing algorithm and ideal link state routing. In *Proc. International Symposium on Computer Communications (ISCC)*, Athens, Greece, April 1998.

[112] P.S Khedhar and S. Keshav. Fuzzy prediction of timeseries. In *Proc. IEEE Conference on Fuzzy systems*, March.

[113] A. Charny. An algorithm for rate allocation in a packet switching network with feedback, M.Sc. Thesis, May 1994.

[114] L. Kalamboukas. *Congestion Management in High Speed Networks*. PhD thesis, University of California Santa Cruz, September 1997.

[115] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proc. ACM SIGCOMM*, Austin, TX, USA, September 1989.

[116] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. Scalable architecture for integrated traffic shaping and link scheduling in high-speed atm switches. *Proc. IEEE Journal on Selected Areas in Communications*, 15, June 1997.

[117] G. Apostolopoulos, R. Guerin, S. Kamat, A. Orda, T. Przygienda, and D. Williams. Qos routing mechanisms and ospf extensions, 1998.

[118] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, and B. Vandalore. The ERICA switch algorithm for ABR traffic management in ATM networks. *TON*, 8(1):87–98, 2000.

[119] P. Johansson, R. Kapoor, M. Kazantzidis, and M. Gerla. Rendezvous scheduling in bluetooth scatternets. In *Proc. International Conference on Computer Communications (ICC)*, New York, NY, April 2002.

[120] IBMResearch. Bluehoc: Bluetooth performance evaluation tool. In *http://oss.software.ibm.com/bluehoc/*.

[121] Ns notes and documentation. In *http://www.isi.edu/vint/nsnam*.

[122] A. Kershenbaum and M. Post. Distributed scheduling of cdma networks with minimal information. *Proc. IEEE Transactions on Communications*, 39, January 1991.

[123] H. Choi and S. Hakimi. Data transfers in networks with transceivers. *Networks*, 18:223–251, 1988.