DYNAMIC RESERVATION MEDIUM ACCESS FOR MULTIHOP WIRELESS REAL-TIME COMMUNICATIONS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Jun Yi,

____________________________________
Christian Poellabauer, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2012
© Copyright by

Jun Yi

2012

All Rights Reserved
Abstract
by
Jun Yi

Various applications, such as battlefield surveillance, industrial process monitoring and control, civil infrastructure monitoring, etc. are enabled by multihop wireless networking technology. Many wireless multihop networks carry streams of data with time-critical information (e.g., video streams in surveillance networks, sensor streams in monitoring and actuating applications, or command and control streams in factory automation applications). Such data must reach their destinations in a predictable and timely manner. Providing real-time communications in such a multihop wireless network is critical to their success. However, providing timeliness support is challenging, mainly due to (i) the inherently unreliable nature of the wireless medium, (ii) the distributed nature of multihop wireless networks, and (iii) the resource-constrained (mainly bandwidth and energy) environments. As a result, the design of an effective and efficient medium access control layer is especially important since it lays the foundation to provide actual timeliness support for all upper layers.

However, existing solutions are either over-coordinated (fixed-schedule-based schemes) or under-coordinated (prioritized contention-based schemes), failing to address this problem efficiently. This thesis introduces DRAMA, a new distributed, progressive, dynamic slot reservation mechanism, aiming to provide
timeliness support at the medium access control layer. In DRAMA, each node progressively and dynamically makes short-term slot reservations according to the timeliness and bandwidth requirements of its outgoing traffic, thereby quickly adapting to traffic and link dynamics. Potentially interfering nodes reserve slots in a serialized and orthogonal manner, which ensures fast, contention-free slot reservations with high bandwidth utilization and low bandwidth overhead. Similar to fixed-schedule-based approaches, nodes in DRAMA can enter a low-power sleep mode when they do not transmit or receive data.

To validate the design, we implemented a prototype and deployed it for extensive experiments. Our experimental results show that DRAMA is able to meet end-to-end latencies under various traffic and link dynamics when other schemes may not be able to do so, while incurring little energy and bandwidth overhead.
To my wife Suhong and my daughter Jiangnan (Jenna) and my prenatal angel.
CONTENTS

FIGURES ................................................................. vi
TABLES ............................................................... iii
PREFACE ............................................................... xi
ACKNOWLEDGMENTS .................................................. xii

CHAPTER 1: INTRODUCTION ........................................... 1
1.1 Contributions ....................................................... 4
1.2 Organization of the Thesis ....................................... 5

CHAPTER 2: BACKGROUND ............................................. 8
  2.1 Schedule-Based Wireless Medium Access Control .............. 9
    2.1.1 Schedule-Coordination ..................................... 11
  2.2 Contention-Based Wireless Medium Access Control ............ 15
  2.3 Duty-Cycled-Based Wireless Medium Access ..................... 18
  2.4 Summary ........................................................ 21

CHAPTER 3: BASICS OF DRAMA ....................................... 22
  3.1 Basic Ideas ....................................................... 22
  3.2 Modular Architecture .......................................... 25
  3.3 Challenges ....................................................... 27
  3.4 Summary ........................................................ 30

CHAPTER 4: ORTHOGONAL AND SERIALIZE RESERVATION SCHEME 32
  4.1 Flexible Frame Structure ....................................... 33
  4.2 Reservation Interference ....................................... 35
  4.3 Reservation Coordination ...................................... 38
    4.3.1 Orthogonal and Serialized Reservations .................. 38
    4.3.2 The Procedure ............................................. 42
FIGURES

1.1 An illustration of factory automation using multihop wireless network technology. ................................................. 3

2.1 An illustration of static schedule-based solutions. .................... 9
2.2 An illustration of IEEE 802.11 distributed coordination function (DCF) ................................................................. 15
2.3 An illustration of activity cycling of S-MAC [76]. ..................... 19

3.1 An illustration of the dynamic and progressive reservation scheme. 23
3.2 The software architecture where DRAMA works. .................... 26
3.3 An illustration of conflicting reservations using the dynamic and progressive reservation scheme alone. ................. 28

4.1 An illustration of frames, slots, RCSs, DCSs, MCSs, and reservation elements. ......................................................... 34
4.2 An example network where each node can interfere with the reception of any other node within its 2-hop range. ............. 36
4.3 An illustration of conflicting reservations among interfering nodes in an example network in Figure 4.2. We assume that all reservations target for slot (i, u). Signal propagations are indicated by arrows and signal collisions are indicated by crosses. We have DCS_{A→B}(i, u) \parallel DCS_{D→E}(i, u) in (a), DCS_{A→B}(i, u) \parallel RCS_{D}(i, u) in (b), and RCS_{A}(i, u) \parallel RCS_{D}(i, u) in (c). ...................... 37

4.4 An illustration of interference-free reservations among non-r-interfering nodes A and E in an example network in Figure 4.2. We assume that all reservations target for slot (i, u). Signal propagations are indicated by arrows and signal collisions are indicated by crosses. In all cases (a), (b), and (c), any reservation made by node A and E are not possible to be conflicting, although their signals collide at node C—yet node C is not the designated receiver of either node A or node E. ...................... 39
4.5 An illustration of using reservation strides to make reservations by nodes A and B (with $\lambda_A = 1$ and $\lambda_B = 2$) in the example network shown in Figure 4.2. Tall and short bars represent RCSs and DCSs, respectively.

4.6 Serializing reservations among $r$-interfering nodes D ($\lambda_D = 4$), C ($\lambda_C = 3$), B ($\lambda_B = 2$), and A ($\lambda_A = 1$) for the network topology in Figure 4.2. For simplicity, we only plot relevant reservations and only show reservation propagations along path $D \rightarrow C \rightarrow B \rightarrow A$. RCS$_D(i)$ reserves DCS$_{D\rightarrow C}(i+4)$ and RCS$_D(i+4)$, which are overheard by C. RCS$_C(i+1)$ (overheard by B) not only reserves DCS$_C\rightarrow B(i+4)$ and RCS$_C(i+4)$ but also relays reservation DCS$_{D\rightarrow C}(i+4)$ and RCS$_D(i+4)$. Note that RCS$_C(i)$ will relay DCS$_{D\rightarrow C}(i+4)$ and RCS$_D(i+4)$ if RCS$_C(i)$ is after RCS$_D(i)$. RCS$_B(i+2)$ (overheard by A) not only reserves DCS$_{B\rightarrow C}(i+4)$ and RCS$_B(i+4)$ but also relays overheard reservations DCS$_{D\rightarrow C}(i+4)$, RCS$_D(i+4)$, DCS$_{C\rightarrow B}(i+4)$, and RCS$_C(i+4)$. Finally, RCS$_A(i+3)$ can reserve RCS$_A(i+4)$ and relay all these overheard reservations.

5.1 An illustration of the timeliness of the first three packets (in case (a), (b), and (c), respectively) of a stream going from the source node A to B, C, D, and to the destination D. Both the minimum packet inter-release interval of the stream and the local deadline at each hop is 4 time units. The timeliness parameters of each packet at each node is denoted as (expected arrival or release time)/(actual arrival or release time)/(absolute local deadline) and are shown below each hop. The expected arrival or release time of each packet are shown in the case that the packet is a future packet. The expected arrival or release time of each packet in the case that the packet has been received is shown. It is simply the absolute deadline of the packet at the previous hop or the actual release time at the source. The three cases (a), (b), and (c) show three different packet delivery scenarios: right on time, deadline miss, and deadline meet, respectively.
An illustration of dynamic reservations of node B ($\lambda_B = 2$) over $B \rightarrow C$ with reservation tasks $S_1^{B\rightarrow C}$ and $S_2^{B\rightarrow C}$. For simplicity, we only plot relevant reservations in node B’s reservation table, and assume that the actual or expected size of each packet is one slot and that link qualities are perfect. $DCS_{B\rightarrow C}(i+2)$ represents the regions of slots within $F_{i+2}$ which can be reserved over $B \rightarrow C$ without conflicting with entries in B’s reservation table. The number of reserved and demand slots before the first region of $DCS_{B\rightarrow C}(i+2)$ is 2 and 4, respectively. Therefore, B reserves 2 slots in the first region to satisfy the timeliness and load requirement of $S_1^{B\rightarrow C}$ and $S_2^{B\rightarrow C}$. There are 2 future packets arriving after the first region and accordingly node B reserves 2 slots in the second region.

An illustration of the end-to-end admission control of periodic real-time stream $S_1$.

The IntelMote2 hardware platform.

An illustration of deferred procedure processing by TinyOS [23].

An illustration of timeliness-aware augmentation of the run queue.

The network testbed, where the NS is a noise source for nodes 3, 4, and 5.

Real-time performance while varying traffic loads (the noise source is off).

Real-time performance while varying traffic loads (the noise source is active).

Real-time performance while varying end-to-end deadlines (the noise source is off).

Real-time performance while varying end-to-end deadlines (the noise source is active).

Real-time performance while varying frame sizes (the noise source is off).

Real-time performance while varying packet interrelease times (the noise source is off).

Real-time performance while varying inter-period traffic loads (the noise source is off).

Cumulative distribution of end-to-end latencies (the noise source is off).

Energy performance while varying the number of streams (the noise source is off).
7.11 Energy performance while varying the number of streams (the noise source is active). .......................................................... 97

7.12 Energy performance while varying frame sizes (the noise source is off). .......................................................... 98

A.1 The state diagram of reservation stride allocation. The statement at the beginning of an arrow is the condition that makes the indicated state transition and that at the end of the arrow is the action taken before moving to a new state. ........................................ 113

B.1 An example of reservation propagation. ........................................ 117

D.1 The software architecture of TinyOS implementation of DRMAM. The line arrows indicate command calls or event callbacks. ........ 121

D.2 An illustration of the implementation of the PacketQueue module. 125

D.3 An illustration of the implementation of the Reserve module. .... 126
TABLES

7.1 DEFAULT EXPERIMENTAL CONFIGURATIONS ....................... 85
7.2 ROBUSTNESS AND OVERHEAD OF RESERVATION PROPAGATION ........................................ 99
7.3 BANDWIDTH OVERHEAD ........................................ 101
7.4 STORAGE OVERHEAD .......................................... 103
PREFACE

This research was partially funded by NSF under grant numbers CNS-0834180 and CNS-0834230. Views and conclusions contained in this document are those of the authors and should no be interpreted as representing the official policies, either expressed or implied, of NSF.

Keywords: Real-Time Communications, Multihop Wireless Networks, Quality of Service, Medium Access Control, Access Admission Control, Scheduling, Distributed Coordination
I am extremely grateful to Professor Christian Poellabauer, my thesis advisor, for giving me the right amount of freedom and guidance to study in a broad range of areas during my graduate studies. From the very beginning, he was instrumental in training my research skills and scientific writing skills, and constantly reminding me that thinking a problem at the highest level at the beginning, is as important, if not more important, than delving into seeking the right solution. Professor Poellabauer not only taught me how to become a better researcher, but also helped me to become a better person. His manner to training and supporting students and collaborating with colleagues and many of his suggestions will influence me for a long time.

I am grateful to the other members of my committee, Professor Sharon Xiaobo Hu, Professor Liqiang Zhang, Professor Michael Lemmon, and Professor Aaron Striegel, for their feedback and advice that helped to shape my research skills. Professor Hu taught me the art of working on a problem systematically. Her inquisitorial and sharp questions helped me to better understand the limitations of my research and motivated me to find better problems to solve and better solutions to these problems. Professor Zhang provided the right balance to my research by not only constantly challenging me to every detail of my work, but also trying to reminding me of ongoing efforts by other research groups. Professor Lemmon showed me how to rigorously define a problem and then pursue its solution. He
also brought new perspectives to my thesis (e.g., using control-based approaches to solve real-time communication problems). Professor Striegel taught me the graduate network course at the beginning of my graduate study, which built the foundation for my research afterward.

I owe a lot to Professor Leonid Faybusovich, Professor Julia Knight, and Professor Peter Cholak at the Mathematics Department. Not having any prior knowledge in these fields, I learned mathematical logic, computability, and universal algebra from Professor Knight and Professor Cholak. They helped me to master many prerequisite courses in very short time with their incisive and many hours of tutoring. They always encouraged me to pursue further study in these areas. Professor Faybusovich introduced me to the world of mathematical optimization and structuring and he guided me through some preliminarily mathematical research projects. There are some traces of using the structuring approach to generate better solutions in this thesis. Although I have not applied a lot what I learned from them in many areas so far, I believe I will benefit from this knowledge greatly in my future work.

I am grateful to all the friends and colleagues with whom I spent my time as a graduate student at the University of Notre Dame. Thanks to my friends, Thida-pat (Tam) Chantem, Dinesh Rajan, and Shengyan Hong. Tam spent many hours discussing the most diverse and exciting topics with me. Dinesh discussed many aspects of real-time systems research with me in my first two years of graduate study. Shengyan always looked at every aspect of my research work and asked me critical questions when most unexpected. Thanks to my DARTS group colleagues and officemates at 214 Cushing, Hongsheng Lu, Shu Liu, Pramita Mitra, Chris Miller, Yihua Zhang, Nikhil Yadav, Dirk Van Bruggen, Mehrdad Aliasgari, and
Salvador Aguinaga. I treasure our daily foosball games that always relaxed me on a busy day. Special thanks to Tam for her help and patience with my countless questions. She was the default good-to-answer-all-questions person whom I asked everything. Special thanks to Hongsheng as well for his insightful comments and guidance on how to present this thesis work clearly and smoothly.

I would like to express my earnest gratitude to my parents and my brother for their love and support, without which any of my achievements would not have been possible. Thanks to my parent for their love and countless sacrifices to raise me and to give me the best possible education. I am deeply indebted to my dear wife Suhong for her love and understanding through my graduate years. She was always behind me and gave her unconditional support even if that meant to sacrifice the time we spent together and sacrifice her educational opportunities. I thank my mother, mother-in-law, sister-in-law, and brother-in-law who devotedly took care of my daughter for a long time. Without their help, it would not have been possible to reach this stage in my career. Finally, thanks to my daughter Jiangnan (Jenna) for the joy and the happiness she brings to me during our many moments together.
CHAPTER 1

INTRODUCTION

A multihop wireless network is an inter-network where each node can forward packets wirelessly. Many networks in research or industry, e.g., wireless mesh networks [4], wireless sensor networks [3, 13, 63], and wireless ad hoc networks [16], belong to this category. Various applications, such as battlefield surveillance [22], industrial process monitoring and control [62], civil infrastructure health monitoring [32, 36], etc. are enabled by multihop wireless networks. Compared to its wired counterpart, a multihop wireless network has the following main advantages [16]:

- It can be deployed rapidly as a lower-cost back-haul.

- It is able to provide coverage in hard-to-wire areas.

- It extends coverage and enhances throughput due to multihop forwarding and wireless broadcast.

- It extends battery life due to lower-power and short-range transmission.

Many wireless multihop networks carry streams of data with time-critical information (e.g., video streams in surveillance networks [5, 32], sensor streams in monitoring and actuating applications [72], or command and control streams in factory automation applications [62]). Such data must reach their destinations in
a predictable and timely manner [62, 64, 77]. As one example, an electric power
grid or water distribution network usually involves controlling some physical pro-
cesses, which demand some type of hard real-time support, i.e., each packet of
control or feedback data must be serviced within a specified deadline to ensure
the overall control application’s performance level [13, 38, 64]. As another exam-
ple, in the factory automation application shown in 1.1 all command or feedback
packets from each field device must reach their destinations before their end-to-end
deadlines. Missing deadlines not only slows down assembling/production line and
therefore decreases productivity, but also degrades product quality since delayed
feedback less accurate control over product operation.

However, providing timeliness support is challenging, mainly due to the fol-
lowing characteristics of multihop wireless networks:

- The inherently unreliable nature of the wireless medium is unfavorable to
  providing timeliness support [13, 64]. Specifically, wireless interferences can
  introduce significant levels of unpredictability if not addressed appropriately.

- The distributed nature of multihop wireless networks is an obstacle to coor-
dinating nodes’ activities in a timeliness-aware, decentralized, coordinated,
  and scalable manner [13].

- Providing timeliness support while being resource (e.g., energy and band-
  width) efficient are often conflicting objectives [64, 74, 77]. Multihop wireless
  networks often operate in resource-constrained environments, where energy
  and bandwidth are typically the most important resources. On the one
  hand, keeping nodes continuously active allows them to adapt quickly to
  network dynamics (e.g., changes in traffic and link quality), yet inevitably
  incurs unnecessary energy drain. On the other hand, powering off nodes
Figure 1.1. An illustration of factory automation using multihop wireless network technology.

intermittently will save energy but will also prolong latencies and reduce the nodes’ responsiveness to network dynamics. Further, providing timeliness support typically has some bandwidth overhead (e.g., to coordinate nodes’ activities), which then reduces the bandwidth available to the actual real-time traffic.

To support real-time communication in a multihop wireless network, the design of the medium access control layer (MAC) is especially important since it lays a foundation to provide actual timeliness support for all upper layers. The medium access control layer should therefore have the following characteristics:
• It is operated in a distributed and scalable manner, e.g., it is able to operate effectively in a network of thousands of nodes.

• It is bandwidth- and energy-efficient, e.g., most of the available bandwidth can be used to deliver actual data traffic, while extra energy consumption is minimized.

• It adapts rapidly to local network dynamics, e.g., traffic fluctuation, link quality deterioration and improvement, and external short-term unpredictable noise, etc.

• It provides upper layers with sufficient functionalities and easy-to-use interfaces to support network-wide real-time communications.

1.1 Contributions

The main contribution of this dissertation is to provide the first distributed solution that aims to support hard real-time communication in a multihop wireless network. In particular, we propose a new MAC-layer protocol, called dynamic reservation medium access (DRAMA), which consists of several schemes to provide timeliness support in a resource-efficient manner. The key ideas that allow us to achieve this goal are:

• The Progressive Reservation Scheme. Each node progressively makes slot reservations for short periods of time, which include slot reservations for both data traffic delivery and future reservation points. The node makes further short-term slot reservations using these future reservation points. Since each future point provides opportunities to reserve appropriate number
of slots at appropriate slots, each node is able to quickly adapt to traffic and link dynamics.

- **The Orthogonal and Serialized Reservation Scheme.** This scheme enforces a serialized reservation order among potentially interfering nodes and achieves fast, negotiation-free, and interference-free reservations with high bandwidth utilization and low bandwidth overhead. However, potentially interfering nodes are allowed to make slot reservations simultaneously within different (i.e., orthogonal) future time intervals, thereby preventing undue delays in the reservation scheme.

- **Real-Time Reservation Policy.** To satisfy the timeliness requirements of real-time traffic, each node dynamically makes reservations that ensure that these requirements are met under various traffic and link conditions.

1.2 Organization of the Thesis

The rest of this thesis is organized as followsings. We discuss related work in Chapter 2. We present overviews of DRAMA in Chapter 3, the orthogonal and serialized reservation scheme in Chapter 4, the real-time dynamic reservation policy for real-time traffic in Chapter 5, and our implementation of a prototype in Chapter 6. We evaluate the performance and overhead of DRAMA in Chapter 7 and conclude our thesis in Chapter 8.

Chapter 2 presents the background information about wireless medium access techniques. We roughly classify existing techniques into three categories: schedule-based, contention-based, and duty-cycle-based schemes. We analyze their advantages and disadvantages and describe the scenarios in which these schemes perform
It then becomes apparent that none of them is a good fit for real-time communications in a multihop wireless network and a new design is therefore needed.

Chapter 3 describes the motivation and the basic idea of DRAMA, i.e., dynamic and progressive reservations scheme, and provides some simple examples to illustrate this scheme. This scheme strikes a balance between fixed-schedule-based and contention-based schemes and retains the best of both worlds. We also describe the software architecture and supportive components to implement such a scheme. At the end of this chapter, an open research problem is brought up, i.e., nodes may make conflicting reservations using the dynamic and progressive reservation scheme alone. This open problem calls for a fast, bandwidth-efficient, and interference-free reservation coordination scheme, which is discussed in the next chapter.

Chapter 4 presents the orthogonal and serialized reservation scheme to address the open problem described in Chapter 3. The basic idea is to introduce a flexible structure to organize interfering nodes’ reservation behaviors. In the first part, we present a flexible frame structure where a node is allowed to reserve different numbers of slots at different positions from frame to frame. We also examine the node-to-node relationship for making conflicting reservations. In the second part, we show how interfering nodes make their reservations within a frame in a serial order to achieve interference-free reservations.

Chapter 5 describes a specific real-time dynamic reservation policy for a typical real-time traffic model. The traffic model is explained in detail and the basic idea and algorithm of the policy are presented thereafter. An accompanying admission control algorithm is presented at the end of this chapter.

Chapter 6 describes the challenges that the hardware and software platforms
impose upon implementing a prototype and then discusses how we address these challenges. We also discuss the implementation of some auxiliary components of DRAMA in this chapter. The lessons and experiences we learned from the implementation are briefly described at the end of this chapter.

In Chapter 7, we present measurements of the real-time performance, energy consumption, storage overhead, and bandwidth overhead for typical real-time streams on the Intel Mote2 platform. The preliminary results show that DRAMA achieves its main goal, i.e., to meet the maximum number of end-to-end deadlines of all streams when traffic and network conditions change frequently.

Chapter 8 concludes the dissertation by summarizing our contributions, exposing some limitations of our solutions, and proposing several directions for future work.
CHAPTER 2

BACKGROUND

Over the past decades, two classes of solutions have been proposed to provide wireless medium access control [37, 44] for various applications: scheduled-based and contention-based. While schedule-based solutions can provide exact access opportunities that match perfectly the requirements of a given scenario, they are inflexible and susceptible to traffic and network dynamics. On the other hand, while contention-based solutions are more flexible, they can only provide unpredictable wireless medium access control.

The remainder of this chapter is organized as follows. Section 2.1 discusses schedule-based solutions. In particular, we first illustrate in general why they are inflexible in response to traffic and network dynamics and then discuss some existing schedule-based solutions in the literature. Section 2.2 discusses contention-based solutions. In particular, we first illustrate in general why they can only provide unpredictable wireless medium access control and then discuss some existing contention-based solutions. Section 2.3 introduces duty-cycle-based solutions, which can be treated as a combination of schedule-based and contention-based solutions. The main purpose of duty-cycle-based solutions is to conserve energy at the cost of throughput and timeliness.
Figure 2.1. An illustration of static schedule-based solutions.

2.1 Schedule-Based Wireless Medium Access Control

Schedule-based schemes [9,11,19,53,58,62,71] are based on deterministic transmission schedules, where a periodic frame is divided into slots and slot allocations determine when a node can transmit or receive data and when a node can power down its network card to preserve energy. Figure 2.1 illustrates such a scheme.

A challenge in schedule-based solutions is that the frame size and all slots assignments must be determined network-wide such that all bandwidth and latency requirements are met [11]. This means that fixed frame and slot sizes are only efficient for very concrete traffic patterns [70], inevitably reducing bandwidth efficiency and throughput for varying traffic workloads and network topologies. Frequent adaptations of frame and slot sizes necessitate network-wide slot re-allocations, thereby resulting in significant communication and computational overhead [62].

Consider a system of an in-phase stream $S_1 = (p = 3, e = 1, d = 3)$, where $p, e,$
and $d$ are the period, transmission time, and deadline (in units of slots) of stream $S_1$. The stream releases its first packet at the beginning of slot 1 and releases more packets every 3 slots thereafter, e.g., releasing the second packet and the third packet at the beginning of slot 4 and 7, etc. Every packet takes 1 slot to be transmitted and the deadline of each packet is 3 slots ahead from its release time, e.g., the deadlines of the first packet and the second packet are at the ends of slot 3 and slot 6, respectively. Figure 2.1(a) shows the slot allocations using the fixed frame size of 7 slots. In order to meet every packet’s deadline, slots 1, 4, and 7 in every frame are allocated to the stream. However, in this scenario, bandwidth is over-reserved. For example, slot 1 of the second frame will be idle since $S_1$ will not have a packet ready for transmission by that time. The reserved bandwidth utilization is $\frac{3}{7}$, whereas the demanded bandwidth is $\frac{1}{3}$.

Further, using static schedule-based solutions, the end-to-end latency of a stream is tightly coupled to the stream’s slots allocated in each frame along the stream’s route [11]. A potential transmission failure in a slot at a node, due to unpredictable external or internal noise and interference, may delay the transmission of all subsequent packets significantly and cause deadline misses. To counteract this effect, a stream may require a large extra share of slots to satisfy its end-to-end latency constraints [70]. Figure 2.1(b) shows that the transmissions for the second and the third packets of $S_1$ fail at slot 4 and slot 7 in the first frame. These two packets are queued up until the second frame and are transmitted at slot 1 and slot 4 in the second frame. Neither of them met their respective deadlines. Even worse, the fourth packet of $S_1$, which is released at the beginning of slot 3 in the second frame, is transmitted at slot 7 in the second frame and therefore missed its deadline, which is the end of slot 6 in the second frame.
In summary, static reservation-based approaches usually suffer from several problems:

- They are inflexible in response to changes in network topology, link quality, timeliness requirements, and bandwidth requirements. Such dynamics are typically addressed using expensive network-wide re-negotiation and redistribution of reservations.

- They incur large bandwidth and computation overheads, e.g., to obtain a contention-free schedule using a multi-party multi-round negotiation-based approach or using a centralized approach, both of which waste bandwidth and limit the network’s responsiveness to traffic changes.

- They limit actual bandwidth utilization. Static reservation-based techniques usually arrange slot reservations into a table of fixed length based on worst-case workload assumptions, therefore over-reserving the medium and wasting bandwidth.

However, static schedule-based solutions usually have the following advantages:

- They are energy efficient since the radio of each node can be switched off within the slots where the node is neither a receiver or transmitter.

- They are bandwidth efficient in the case of heavy traffic, comparing to the contention-based solutions [57] where large portion of bandwidth is wasted due to idle sensing, collisions, backoffs, and retransmissions.

2.1.1 Schedule-Coordination

Various coordination approaches are used to allocate interference-free slots to nodes, e.g., each slot allocated to a node is guaranteed to be interference-free
from any other node’s transmission. These approaches can be roughly classified as centralized coordination, random-function-based coordination, and distributed negotiation-based coordination.

**Centralized Coordination.** In a scheme based on centralized coordination, a special node (usually the control or administrative node) is served as the coordinator. The coordinator collects traffic demands of all streams, communication qualities of all links, and link/node interference information of the entire network. Then, it runs a centralized algorithm to generate a schedule for each node, which indicates which slots the node shall transmit and receive. The generated schedules are communicated to the corresponding nodes. Nodes follow their respective schedules to transmit or receive packets from their neighbors.

WirelessHART \[62\] is an example of centralized coordination. It is a TDMA-based wireless mesh networking technology and the first open wireless communication standard specifically designed for process measurement and control applications. The wireless mesh network includes three main elements: wireless field devices, gateways, and a network manager. Wireless field devices are linked to processes or plant equipments with WirelessHART protocol stack enabled. Gateways enable communication between wireless field devices and host applications connected to a high-speed backbone or other existing plant communications network. The network manager has the responsibility for configuring the network, scheduling communications between devices, managing message routes, and monitoring network health. This element can be integrated into gateways, host applications, or process automation controllers. The network manager has to assign time slots to each device and distributes them to the corresponding devices.

**Random-Function-Based Coordination.** Using a random-function-based co-
ordination, all nodes in a network share a random function, whose purpose is to compute the priority value of each node for each slot. The node with the highest priority has the right to transmit within the corresponding slot, which may result in “priority chains” in a network and thereby greatly wastes bandwidth. For example, consider a chain of nodes and the computed priority values decrease sequentially from the head node to the tail node. In this case, only the head node can transmit since it has a higher priority value than all of its neighbors (i.e., the node next to the head node). All other nodes can not transmit since each of them has a lower priority than the node proceeding it. Contending nodes are guaranteed to have different priorities for any given slot, therefore achieving contention-free medium access. Moreover, the function is random, e.g., the function returns different priorities for different slots, thus avoiding starving a node.

The function’s input parameters usually consists of node identification and slot index. The function’s return value is the priority value of the input node at the given slot. The following is the random function used by TRAMA \cite{55} to define node $u$’s priority at slot $t$:

$$prio(u, t) = hash(u \oplus t),$$  \hspace{1cm} (2.1)

which is a pseudo-random hash function of the concatenation of node $u$’s identification and $t$.

Assuming that node identifications are unique and nodes’ clocks are synchronized, all nodes compute the same priority value at any given time slot. At each slot, a node runs the random function with the identifications of all of its contending nodes. If the node itself has the highest priority (the largest return value) among all of its contending nodes, then it starts to transmit queued packets. How-
ever, if the selected node does not have any data to send, the slot is wasted. To achieve higher bandwidth utilization and efficiency, instead, the slot will be re-used to satisfy other nodes’ traffic demands. The selected node may indicate to other nodes that it can give up this slot (if it does not have any packets to send); this slot could then be used by another node. Nodes exchange current traffic information with their neighbors to accomplish slot re-use. The bandwidth cost of exchanging traffic information is high. Moreover, due to the function’s random nature, random-function-based coordination is not able to provide deterministic latencies and to actively reserve slots according to quality of service requirements.

**Distributed Negotiation-Based Coordination.** Using a distributed negotiation-based coordination, a node indicates its reservation attempt for a slot based on its own traffic requirements and communicates this attempt to all of its contending nodes. Only after the node receives acknowledgments from all of its contending nodes, this reservation attempt is treated as successful and the node can use this slot for actual traffic delivery. If multiple conflicting reservation attempts from contending nodes take place at the same time, then these contending nodes must resolve these conflicting reservations usually using multi-round or multi-phase negotiations.

The coordinated distributed scheduling (CDS) scheme of the optional WiMAX MeSH mode [29] is an example of distributed negotiation-based coordination. CDS uses a multi-stage multi-party reservation negotiation to achieve contention-free reservation, which reacts slowly to traffic changes and wastes bandwidth.
2.2 Contention-Based Wireless Medium Access Control

In a basic form of a contention-based scheme (e.g., [14, 21, 26, 27, 45]), a node starts to transmit packets while the shared wireless medium is sensed as idle. Only the acknowledgement from the receiver to the transmitted packet can indicate the success of the corresponding transmission. Retransmissions are required if the acknowledgement has not been received, which not only increases delivery latencies but also brings large variations to delivery latencies. In general, a contention-based scheme has the following advantages:

- It is flexible, robust, and scalable.
- It has a very high bandwidth utilization and efficiency when contention is light or infrequent, for example, only one node is transmitting [57].
- It is simple to implement and completely distributed.

However, it suffers from the following disadvantages:
• It is energy inefficient since it requires nodes to stay active to continuously monitor or listen to the channel (“idle listening”) and retransmit failed packets, therefore leading to excessive energy consumption [6, 60].

• It has low bandwidth utilization when contention is intensive since large portions of bandwidth are wasted due to frequent collisions, backoffs, and retransmissions [60].

• It brings large variation to packet delivery latency since it uses random backoff techniques to respond to collisions, potentially leading to unpredictable latencies [60].

Perhaps the simplest form of contention-based medium access control involves carrier sense multiple access with collision avoidance (CSMA/CA) [26, 27], where wireless networks attempt to avoid collisions instead of detecting them as in its wired counterpart CSMA/CD [28]. CSMA/CA attempts to avoid collisions by using a control message exchange to reserve the wireless channel before each data message transmission. A device with a message to send first performs the CSMA algorithm to find an appropriate transmission time. Once the CSMA algorithm determines a transmission time, the source device transmits a request to send (RTS) control message to the intended receiver. If the receiver can receive the transmitted RTS control message it responds with a clear to send (CTS) control message. The source device retries the transmission of RTS control messages at a later time if it does not receive a CTS within a certain time. The receiver device does not respond with a CTS if it can not safely transmit. For example, if the receiver detects a transmission in its neighborhood, but the source does not detect this transmission, then the receiver will defer the pending communication and not send a CTS back to the source. After a successful reception of a CTS,
the source transmits the data message. Neighboring nodes that receive an RTS or CTS message know a data transfer will occur soon and delay attempting any message transmissions until a later time. While CSMA/CA reduces the effect of hidden terminals and associated energy losses in wireless networks, it requires devices to listen to the channel constantly and transmit multiple messages for each data message.

A popular variant of CSMA is distributed coordination function (DCF) as shown in Figure 2.2, which uses a channel access mechanism similar to slotted CSMA/CA and uses acknowledgments for reliability. In addition to the physical carrier sensing according to the CSMA algorithm, devices using DCF perform virtual carrier sensing by tracking channel utilization with control messages. Each device maintains a counter, called the network allocation vector (NAV), that indicates the channel has activity on it whenever the NAV has a non-zero value. Devices update the NAV based on the data length present in control messages they receive. Periodically, each device decrements its NAV so that the current transmission ends when the NAV reaches zero. Using the NAV allows a device to quickly check for possible channel activity without constantly listening to the channel (therefore saving energy). A device using DCF considers the channel busy when either the physical channel sensing detects a transmission or the NAV is non-zero. When the channel is not considered as busy, the node may start to transmit queued packets.

Some prior work on contention-based scheduling combined with priority scheduling have resulted in probabilistic solutions to limit latencies within certain ranges (e.g., [21, 34, 35, 45, 67, 75]), but they do not attempt to meet every packet’s deadline. They usually provide timeliness support by adaptively adjusting the
contention window sizes and backoff times according to the urgency level of the outstanding packet. They can only provide soft-real-time performance for packet scheduling locally.

2.3 Duty-Cycled-Based Wireless Medium Access

Using a duty-cycle-based solution [8, 52, 57, 68, 76], a node cycles its radio between the high power active state and the low power sleep state. This solution trades performance characteristics, such as throughput and latency, for a decrease in energy consumption to prolong a node’s lifetime. There are many variants of duty-cycle-based solutions. In general, a duty-cycle-based solution has the following advantages:

- It is energy efficient since nodes sleep at most time and only switch on occasionally.
- It is ease to implement and completely distributed.

However, it suffers from the following disadvantages:

- It leads to large end-to-end latency since nodes sleep most of the time while a transmission succeeds only when both the sender and the receiver are active.
- It leads to large variation in packet delivery latency since the time for the sender and the receiver to be active at the same time is highly unpredictable.

One example of duty-cycle-based solution is S-MAC [76], as shown in Figure 2.3. It forms virtual clusters (where nodes in a cluster do not strictly synchronize their sleep schedules) of neighboring nodes for communications. A node periodically transmits SYNC messages at the beginning of its active frames. The SYNC
messages allow their neighbors to learn its schedules so that they can wake up at the proper time to transmit or receive a message. To better synchronize nodes’ schedule within a cluster or across clusters, nodes adopt the schedule of their neighbors in several cases. First, if a node currently does not have a schedule and hears a SYNC message, it adopts the schedule and joins the virtual cluster. Second, if a node hears multiple yet different schedules, it adopts them all so as to allow communications between different virtual clusters. Third, a node that does not hear any SYNC messages from neighbors chooses its own schedule.

Figure 2.3. An illustration of activity cycling of S-MAC [76].

Message transfer occurs using the traditional RTS/CTS/DATA/ACK procedure [26, 27] to limit collisions and the hidden terminal problem. Nodes transmit the RTS and CTS messages during the active time period, but the data message gets transferred during the inactive period so the uninvolved sensor nodes may sleep. Nodes that overhear an RTS or CTS message for another node can enter the sleep state to conserve energy. To prolong sleep times and ensure that other nodes do not corrupt a transmission, all nodes perform both physical and virtual carrier sensing. The RTS and CTS messages contain the message transmission time, including time for the ACK message, so that nodes may sleep until the
transmission completes. Nodes that wake up with data to send sense the channel for a random time and only transmit if they do not detect any activity.

B-MAC [52] is another example of duty-cycle-based solution, where nodes independently follow a sleeping schedule based on the target duty cycle for the network. Since the schedules of a pair of communicating nodes are independent, the transmitting node uses very long packet preambles so that the receiver can catch the transmission with high probability. Nodes that sense activity on the channel remain awake to receive the message following the preamble or return to sleep if they do not detect activity on the channel. Before transmitting, nodes delay a random time and sense the channel to prevent corrupting an ongoing transmission. Comparing to S-MAC, B-MAC is less coordinated and therefore more flexible, but it needs the supports from applications to specify the target duty-cycle.

B-MAC introduces excess latency at each hop and is suboptimal in energy consumption due to long preamble sampling, which are overcome by X-MAC [8]. X-MAC employs a shortened preamble approach to allow pairs of transmitters and receivers to have more opportunities to engage earlier, therefore reducing per-hop latency. The shortened preambles also significantly reduce energy consumption by saving preamble sampling times.

Z-MAC [57] combines the strengths of static schedule-based and contention-based approaches while offsetting their weakness. Similar to the schedule-based protocols, Z-MAC assigns nodes time slots, but easily allows them to utilize slots they do not own through contention-based approaches with prioritized back-off times. This provides Z-MAC with the capability to perform well similar to contention-based approaches when applications generate less traffic, but approxi-
mates static schedule-based approaches when traffic requirements increase. In the worst case, the performance of Z-MAC falls back to that of CSMA.

2.4 Summary

Wireless medium access control schemes have received much attention in the past decades from researchers and commercial vendors. Unfortunately, these advances do not directly apply to wireless real-time networks because their goals and constraints differ from those of wireless real-time networks. The largest difference comes from the conflicts between responsiveness to network dynamics and timeliness predictability, which is not the design goal of traditional wireless medium access control schemes. They are either over-coordinated (schedule-based schemes) or under-coordinated (contention-based schemes), failing to provide hard real-time traffic delivery in response to traffic and network dynamics. As a consequence, a new scheme is needed to strike a balance between them.
CHAPTER 3

BASICS OF DRAMA

In this chapter, we will first describe the basic idea (i.e., the dynamic and progressive reservation scheme) of DRAMA, then introduce the software architecture where DRAMA fits in, and at the end bring up an open problem of achieving contention-free reservations using this basic idea.

3.1 Basic Ideas

To provide timeliness support in a resource-efficient manner, both sender and receiver should wake up on demand to communicate while all other potentially interfering nodes power off their radios, which favors a static schedule-based solution. However, to provide fast and reliable packet delivery with time-varying external and internal traffic and environment interferences, both sender and receiver should act flexibly and only use acknowledgement to guarantee delivery reliability, which favors a contention-based solution.

We propose a unique solution, called dynamic reservation medium access (DRAMA), which makes use of the best sides of the two aforementioned solutions. In our own view, static schedule-based solutions are over-coordinated, yet prioritized contention-based solutions are under-coordinated. Both of them fail to efficiently support real-time communications in a multihop wireless environment.
Using DRAMA, a node not only reserves slots for actual data transfer to its next-hop neighbors, but also for future channel reservation points which reserve slots for data transfer and further reservations. This dynamic and progressive reservation process continues until there is no more traffic to be sent by the node. In Figure 3.1 a tall bar indicates a reservation point while a short bar indicates a reservation for actual data transfer. During the first reservation point shown in this graph, one future reservation for data transfer and one future reservation point are reserved. This reservation point is also used to reserve two more reservations for actual data transfer and one further future reservation point.

While the dynamic and progressive reservation scheme ensures that a single node is able to quickly adapt to changes in the network by frequently adjusting their reservations accordingly, the proposed protocol must also ensure that no conflicting reservations for the same slot can occur among multiple nodes. Toward this end, DRAMA introduces a serialized and orthogonal reservation scheme, built upon the dynamic and progressive reservation scheme, to reserve slots in a dis-
tributed manner among potentially interfering nodes. Given a certain future time interval, using this scheme a node is only allowed to reserve slots within the time interval if the node is aware of all previous reservations of other potentially interfering nodes for the same time interval. That is, DRAMA enforces a serialized reservation approach among potentially interfering nodes. However, potentially interfering nodes are allowed to make slot reservations simultaneously for different (i.e., orthogonal) future time intervals, thereby preventing undue delays in the reservation scheme. This serialized and orthogonal reservation technique is discussed in detail in Chapter 4.

Finally, to satisfy the timeliness requirements of real-time traffic, nodes making slot reservations have to consider link status as well as the arrival times and deadline requirements of received and upcoming packets. Wireless links are typically unstable (e.g., due to usually uncontrolled and unpredictable environment noises) and therefore the predicted traffic and link status is not always accurate. DRAMA introduces a real-time reservation policy, built upon the serialized and orthogonal reservation scheme, to compensate for these inaccuracies by dynamically and periodically adjusting the number of slots to reserve in a short term. This real-time reservation scheme is discussed in Chapter 5.

In summary, DRAMA strikes a balance between contention-based and static schedule-based solutions:

- DRAMA makes short-term reservations (as opposed to long-term reservations used in static schedule-based solutions), allowing a node to quickly adapt to changing traffic requirements or network conditions without the need for an expensive network-wide re-design and re-distribution of schedules like contention-based solutions. However, unlike ad hoc and random
reservations used by contention-based solutions (e.g., using RTS/CTS pairs by CSMA/CA), this short-term progressive reservation scheme allows a node to reserve slots according to its time-varying traffic timeliness and load requirement.

- Like static schedule-based solutions, nodes in DRAMA only transmit and receive packets in their reserved slots and go to sleep in all other slots and avoids continuously monitoring or listening to the channel (“idle listening”) used by contention-based solutions, therefore reducing large portion of energy consumption.

- Transmissions in DRAMA are contention-free and no bandwidth is wasted due to collisions, backoffs, and retransmissions as in contention-based solutions, therefore DRAMA has higher bandwidth utilization than contention-based solutions. Moreover, slot reservations for a stream in DRAMA vary from frame to frame according to actual traffic demands, whereas using static schedule-based solutions they are fixed and reserved in advance according to the worst-case traffic demand. Consequently, DRAMA has higher bandwidth utilization and traffic throughput than static schedule-based solutions.

3.2 Modular Architecture

To implement the dynamic and progressive reservation idea at the medium access layer, DRAMA needs supports from other components from other layers or the medium access layer itself [33], as shown in Figure 3.2. We assume that (i) the physical layer provides time-slotted packet and radio on/off primitives (e.g., [23, 49, 65]), which are used by DRAMA to wake up the radio whenever
packets are to be transmitted or received, (ii) the link layer provides link quality monitoring (e.g., [2, 50]), link interference detection (e.g., [42, 46, 50, 53, 61, 80]), and time synchronization (e.g., [12, 18, 47, 66, 79]), and (iii) the routing and application layers have already established a route for a given real-time stream [31, 51, 73] and that the stream’s end-to-end deadline has been decomposed into local deadlines for each hop along that route (e.g., [34, 39, 43, 59]). We discuss our own implementations of these components in Chapter 6. The main goal of DRAMA at the medium access layer is to ensure that assigned local deadlines are met in a resource-efficient manner.
DRAMA lies at the wireless medium access layer. It relies on three auxiliary functionalities, whose implementations are discussed in detail in Chapter 6. First, it needs to synchronize neighboring nodes to a common clock so nodes know when to sleep and wake up. Second, DRAMA needs a link monitor to tell the transmission rate and packet reception ratio of a link. Third, DRAMA needs an interference manager to tell which node’s transmission interfere with which other nodes’ reception. DRAMA is responsible for (i) exploiting the interference relationship (provided by the interference manager) and the link quality (provided by the link monitor) to achieve contention-free multiple access and to satisfy the quality of service requirements of all admitted streams; and (ii) switching on or off the wireless radio based on the slot schedule. While the routing and end-to-end layer adapts to network dynamics at the path and network level, DRAMA adapts to network dynamics locally at the link level, but may also trigger adaptations at the routing and end-to-end layer, if necessary.

3.3 Challenges

The aforementioned dynamic and progressive reservation scheme allows nodes to adapt their reservations whenever traffic or network conditions change. However, it presents two main challenges to implement such an approach efficiently.

The first challenge is how to coordinate nodes’ reservations. Figure 3.3 illustrates the conflicting reservations problem. If interfering reservations coexist, then the communications using these reservations will collide, leading to non-deterministic packet delivery latency, energy inefficiency, and bandwidth inefficiency, which is the opposite side of real-time communications. In the example
network, each node can communicate with its direct neighbor and can interfere with the reception of any other node within its 2-hop range. In each case two indicated transmitters reserve overlapped time interval for their respective transmissions. We assume that both indicated transmitters in each case transmit all the time within their reserved time intervals. In (a), node A and node C transmit to node B. Node B can not decode packets from the two transmitter at the same time and thus the actual communications using the two reservations fail. In (b), node A’s signal interferes with node D’s signal at node C, therefore C can not
successfully decode the packet from node D, which happens to node B as well—it can not decode the packet from node A due to node D’s interference. In this case, the actual communications using the two reservations fail. In (c), node B can not decode the packet from node A due to node C’s interference, however node D can successfully decode the packet from node C without interference from node A. In (d), node B can not decode the packet from node A due to node D’s interference, however node E can successfully decode the packet from node D without interference from node A.

In summary, a potential coordination scheme is desirable to have the following characteristics:

- It ensures that all reservations (including reservations for both actual data transfer and further reservation point) must be contention-free, which prohibits reserving conflicting reservations.

- It uses an appropriate amount of bandwidth for coordinating nodes’ reservations or resolving conflicting reservations.

- It coordinates nodes’ reservations in a timely manner, which does not counteract the responsiveness characteristic of the dynamic and progressive reservation scheme.

- It is distributed and scalable so that it can operate well in a network of thousands of nodes.

We address this challenge in Chapter 4.

The second challenge is how to reserve an appropriate amount of time intervals for each node at appropriate positions under changing link and traffic conditions, in order to satisfy the timeliness requirements of real-time traffic. A reservation
policy is needed to address this challenge and desirable to have the following characteristics:

- It should meet as many packets’ deadlines as possible. For example, in order to meet more packet deadlines, it should be able to slow down less urgent packets in favor of highly urgent packets and to use every available time slot to meet every packet’s deadline.

- The computational complexity of the reservation policy must be computationally efficient, leading to small energy consumption.

- An accompanying local admission control is needed to limit local traffic ingestion, preventing local neighborhood from overloading. The admission control shall be effective, e.g., it has modest computation complexity and is not too pessimistic.

We address this challenge in Chapter 5.

### 3.4 Summary

We use a technique called the dynamic and progressive reservations scheme to strike a balance between contention-based and schedule-based schemes. In this basic scheme, a node progressively reserves time slots to be used for (i) data communications to next hops and (ii) future reservation points that will be used to reserve slots for the next time interval. This dynamic approach allows the network to rapidly adapt to traffic load and network changes without the need for an expensive network-wide re-design and re-distribution of the schedule. Dynamic reservations can therefore provide real time support with high link utilization, similar to contention-based methods, but without experiencing the non-deterministic
consequences and energy-costly idle listening of contention-based channel access schemes.

Although the dynamic and progressive reservation scheme is flexible and adaptive, there is a potential risk for conflicting reservations among interfering nodes, e.g., interfering nodes may attempt to make reservations that overlap in time. This situation necessitates a mechanism to coordinate the reservations among interfering nodes in a bandwidth-efficient and expedite manner, we address this problem in Chapter 4. Moreover, reserving appropriate length of time interval at appropriate positions under changing link and traffic conditions is critical to satisfy the timeliness requirements of real-time traffic, we address this problem in Chapter 5.
CHAPTER 4

ORTHOGONAL AND SERIALIZIED RESERVATION SCHEME

To achieve fast, bandwidth efficient, and contention-free reservations based on the dynamic and progressive reservation scheme, an appropriate level of structure to coordinate interfering nodes’ reservations is needed. A rigid and complicated reservation structure will be inevitably inflexible and incur large communication and computation overhead to maintain the structure, especially at the time when network and traffic conditions frequently change. A random or ad hoc reservation structure may not be able to bring the desirable reservation properties. We introduce a flexible yet well-defined structure to coordinate interfering nodes’ reservations.

First, on the one hand, we adopt the frame and slot concept from TDMA-based solutions to facilitate structuring nodes’ reservations. On the other hand, unlike TDMA-based solutions, nodes can reserve slots at any position within a frame and vary their reservations from frame to frame according to their traffic demand and network conditions. This flexibility allows DRAMA to easily implement the dynamic and progressive reservation scheme and also lays a foundation to achieve contention-free reservations efficiently. We discuss the flexible frame structure in Section 4.1.

Second, to avoid conflicting reservations, we need, in the first place, to identify which nodes may make these conflicting reservations, i.e., which node’s trans-
mission interferes with which other nodes’ reception. We introduce the concept of reservation interference \((r\text{-interference})\) to identify the nodes that could make conflicting reservations with one another. We discuss \(r\text{-interference}\) in Section 4.2.

Third, we use reservation strides to characterize the interference relationships between interfering nodes. Reservation strides are integers and the difference between the reservation strides of two interfering nodes measures the length of the communication duration to coordinate the two nodes’ reservations. Combining the frame structure and the concept of reservation stride, we make a balance between reservation speed and reservation order. On the one hand, potentially interfering nodes are allowed to make slot reservations simultaneously within different (i.e., orthogonal) future frames, thereby preventing undue delays in the reservation scheme. On the other hand, reserving slots within any given frame is serialized among interfering nodes (e.g., a node is only allowed to reserve slots within any given frame if the node is aware of all previous reservations of other potentially interfering nodes within the same frame), leading to negotiation-free, interference-free reservations with high bandwidth utilization and low bandwidth overhead. We discuss reservation interference in Section 4.2 and reservation stride and the resulting orthogonal and serialized reservations in Section 4.3.

4.1 Flexible Frame Structure

Similar to other TDMA-based approaches, DRAMA uses the concepts of frames and slots, as illustrated in Figure 4.1(a). Unlike them, a node in DRAMA not only reserves slots for actual data transfer to its next-hop neighbors, but also slots that serve to make further reservations in the future. To implement this progressive reservation approach, DRAMA uses the concept of interference-free
communication session (CS). A communication session is a reservation of one or more consecutive slots, which will be used for two purposes. First, a data communication session (DCS) is used to transmit actual real-time data between a sender and one or more receivers (i.e., unicast or multicast). Note that a multicast DCS can be treated as multiple unicast DCSs sharing the same transmitter and therefore, we only consider unicast DCSs in the following sections. Second, a reservation communication session (RCS) is used to broadcast new reservations of future RCSs and DCSs, but also to inform receivers about existing reservations made previously by neighboring nodes. Finally, we also introduce a contention-based communication session, called management communication session (MCS), which is used for network management purposes (e.g., nodes can compete for these slots to join the network and to communicate interference measurements).

For simplicity, we assume unidirectional communications (i.e., DCS communi-
cations do not require immediate acknowledgments); however, our approach can also be applied to bidirectional communications (e.g., by treating a bidirectional communication as two unidirectional communications). In the following sections, if we do not need to distinguish between RCS and DCS, we will simply refer to both of them as CS. Also, we will sometimes say that an RCS (instead of a node) makes reservations, if there is no ambiguity.

Further, we denote the $i$th frame as $F_i$ and the $u$th slot in $F_i$ as $(i, u)$. The RCS at $(i, u)$ reserved by A is denoted as $\text{RCS}_A(i, u)$. The DCS over link $A \rightarrow B$ at $(i, u)$ reserved by A is denoted as $\text{DCS}_{A \rightarrow B}(i, u)$. If the subscripts or slot indices are irrelevant in the context, we will omit them. A CS that occupies multiple consecutive slots can be treated as multiple single-slot CSs corresponding to these constituent slots, therefore the aforementioned notations are applicable in general.

In DRAMA, reservation information is encapsulated within an RCS payload as a collection of reservation elements, each of which is in the form of (transmitter, receiver, first slot index, number of slots, frame index). The structures of RCSs and reservation elements are shown in Figure 4.1(b) and 4.1(c), respectively. Fields other than reservation elements within an RCS are mainly used for time synchronization and slot and frame alignment (as discussed in Section 6.2.1) in Chapter 6.

4.2 Reservation Interference

To ensure that all DCS and RCS reservations are interference-free (i.e., will not conflict with existing reservations), we introduce the concept of reservation interference ($r$-interference) to identify the nodes that could make conflicting reservations with one another. Using this concept and the aforementioned frame struc-
Figure 4.2. An example network where each node can interfere with the reception of any other node within its 2-hop range.

ture, we then discuss how to coordinate reservations among r-interfering nodes in next section.

We denote the set of the neighbors of node A as $N_A$ and the set of nodes that potentially interfere with node A’s reception from its neighbors as $I_A$. For example, in Figure 4.3, $N_A = \{B\}$ and $I_A = \{B, C\}$ (assuming that the interference range is twice the transmission range). As described in Section 4.1, we require that all RCSs and DCSs are interference- and conflict-free, i.e., the receiver(s) of each DCS or RCS transmission will receive the transmission without collision.

The potential interference relationships can be divided into three categories:

- for DCS$_{A \rightarrow X}(i, u)$ and DCS$_{B \rightarrow Y}(i, u)$ to be interference-free (denoted as DCS$_{A \rightarrow X}(i, u) \parallel$ DCS$_{B \rightarrow Y}(i, u)$), we require that $(X \notin I_B) \cap (Y \notin I_A)$,

- for RCS$_A(i, u) \parallel$ DCS$_{B \rightarrow Y}(i, u)$, we require that $(Y \notin I_A) \cap (\forall X \in N_A(X \notin I_B))$, and

- for RCS$_A(i, u) \parallel$ RCS$_B(i, u)$, we require that $(\forall X \in N_A(X \notin I_B)) \cap (\forall Y \in N_B(Y \notin I_A))$.

Figure 4.3 shows the aforementioned three categories of reservation interferences. We have the following relationship among these three categories: RCS$_A(i, u) \parallel$ RCS$_B(i, u)$ $\Rightarrow$ RCS$_A(i, u) \parallel$ DCS$_{B \rightarrow Y}(i, u)$ $\Rightarrow$ DCS$_{A \rightarrow X}(i, u) \parallel$ DCS$_{B \rightarrow Y}(i, u)$.
The key to avoiding interference is to ensure that two nodes cannot make interfering RCS reservations, i.e., if all RCS communications are interference-free, all DCS communications will also be interference-free. We call two nodes that can make interfering RCS reservations as *reservation interfering* (or *r-interfering*), which is formally defined as follows:

**Definition 1. R-Interference.** Nodes $A$ and $B$ r-interfere with each other,
denoted by \( A \sim B \) if \( (\mathbb{I}_B \cap \mathbb{N}_A) \cup (\mathbb{I}_A \cap \mathbb{N}_B) \neq \emptyset \).

In Figure 4.2 we have \( A \sim D \), even though they may reserve non-conflicting DCSs. The combinations of nodes that are not r-interfering are nodes A and E. From Definition 4.1, we can see that a reservation conflict only happens among r-interfering nodes. As shown in Figure 4.4, any reservation made by node A and E are not possible to be conflicting. To avoid reservation conflicts without sacrificing reservation speed, we must efficiently synchronize reservation behaviors among r-interfering nodes, which is discussed in next section.

4.3 Reservation Coordination

We describe how r-interfering nodes coordinate to achieve fast, negotiation-free, interference-free reservations in this section, exploiting the flexible frame structure and the r-interference relationship described aforementioned in Section 4.1 and Section 4.2 respectively.

4.3.1 Orthogonal and Serialized Reservations

The key to ensuring interference-free reservations within a given frame is to separate the reservations made by r-interfering nodes in time, such that it is possible for a node to learn about all existing reservations made by its r-interfering nodes before making its own reservations within that same frame. Toward this end, we associate a reservation stride (or shortly stride) to each node (which is done in a distributed manner and discussed later), denoted as \( \lambda_A \) for node A. To make reservations progressively, a node (say node A) uses an RCS (say RCS\(_A(i)\)) in each frame (say \( F_i \) for RCS\(_A(i)\)) to make new reservations only within \( F_{i+\lambda_A} \).

To obtain orthogonal reservations among r-interfering nodes, two r-interfering
Figure 4.4. An illustration of interference-free reservations among non-r-interfering nodes A and E in an example network in Figure 4.2. We assume that all reservations target for slot \((i, u)\). Signal propagations are indicated by arrows and signal collisions are indicated by crosses. In all cases (a), (b), and (c), any reservation made by node A and E are not possible to be conflicting, although their signals collide at node C—yet node C is not the designated receiver of either node A or node E.

Nodes with different strides will make reservations within different frames (using their respective RCSs within the same frame), which is called reservation orthogonality. For example, in Figure 4.5, node B has a reservation stride of 2 and RCS\(_B(i)\) reserves both DCS\(_{B\rightarrow C}(i + 2)\) and RCS\(_B(i + 2)\). At node A with \(\lambda_A = 1\), RCS\(_A(i)\) reserves both DCS\(_{A\rightarrow B}(i + 1)\) and RCS\(_A(i + 1)\). RCS\(_A(i + 1)\), which is itself reserved by RCS\(_A(i)\), reserves DCS\(_{A\rightarrow B}(i + 2)\) and RCS\(_A(i + 2)\).

While reservation orthogonality accelerates reservation speed, it does not au-
automatically lead to interference-free reservations. For example, assume two nodes D and A with $\lambda_D > \lambda_A$ and any given $F_i$. Then, both $RCS_A(i + \lambda_D - \lambda_A)$ and $RCS_D(i)$ reserve slots within $F_{i+\lambda_D}$. Specifically, node D reserves slots within $F_{i+\lambda_D}$ before node A does, which is denoted as $D < A$, since $RCS_D(i)$ is before $RCS_A(i + \lambda_D - \lambda_A)$. The corresponding reservation time gap $\Delta_{D<A}$ for D and A to reserve slots within $F_{i+\lambda_D}$ is equal to the time interval from $RCS_A(i + \lambda_D - \lambda_A)$ to $RCS_D(i)$, which is $\lambda_D - \lambda_A$ (in units of frames). Now suppose that the reservation propagation latency of a new reservation from D to A is $\delta_{D<A}$ (in units of frames), which measures the length of time interval from the moment that the reservation was made by node D to the moment that it is received by node A. The reservation propagation latency is incurred by other nodes relaying the reservation information via their own RCSs. Node A may not be able to receive D’s reservation within $F_{i+\lambda_D}$ before A makes its own reservations within $F_{i+\lambda_D}$, i.e.,
Figure 4.6. Serializing reservations among r-interfering nodes D (λ_D = 4), C (λ_C = 3), B (λ_B = 2), and A (λ_A = 1) for the network topology in Figure 4.2. For simplicity, we only plot relevant reservations and only show reservation propagations along path D → C → B → A. RCS_D(i) reserves DCS_D→C(i + 4) and RCS_D(i + 4), which are overheard by C. RCS_C(i + 1) (overheard by B) not only reserves DCS_C→B(i + 4) and RCS_C(i + 4) but also relays reservation DCS_D→C(i + 4) and RCS_D(i + 4). Note that RCS_C(i) will relay DCS_D→C(i + 4) and RCS_D(i + 4) if RCS_C(i) is after RCS_D(i). RCS_B(i + 2) (overheard by A) not only reserves DCS_B→C(i + 4) and RCS_B(i + 2) but also relays overheard reservations DCS_D→C(i + 4), RCS_D(i + 4), DCS_C→B(i + 4), and RCS_C(i + 4). Finally, RCS_A(i + 3) can reserve RCS_A(i + 4) and relay all these overheard reservations.

Δ_D<A < δ_D<A. Therefore, A may be unaware of D’s reservations and may reserve slots that conflict with D’s reservations.

The key to achieve interference-free reservations among r-interfering nodes
turns to assigning appropriate strides to different r-interfering nodes, such that within any given frame $F_i$, each node in an r-interfering set starts to reserve slots within $F_i$ only after it has learnt about all prior reservations made by other r-interfering nodes, which is called reservation serialization. That is, to avoid conflicting reservations, we must require that the reservation time gap between two r-interfering nodes is not less than the reservation propagation latency between them (i.e., $\Delta_{D_{<A}} < \delta_{D_{<A}}$).

Reservation propagation latencies are affected by the reservation relay policies and the reservation time gaps are affected by the reservation stride assignment rules. To achieve a robust interference-free communication, we need an appropriate reservation relay policy and an accompanying reservation stride assignment rule, which are discussed in Section 4.3.3 for both ideal and practical environments.

Before we delve into the design of concrete reservation stride assignment rules and reservation relay policies and their tradeoffs in Section 4.3.3, we will present a general orthogonal and serialized reservation procedure in next section, where any specific reservation stride assignment rules, reservation relay policies, and slot reservation policies (a concrete example for real-time traffic is presented in Chapter 5) can be plugged in.

4.3.2 The Procedure

We summarize the aforementioned orthogonal and serialized reservation scheme in detail in this section. We assume that each node uses a reservation table to track both its reserved CSs and all other CSs made by other nodes this node knows about (via overhearing past RCS broadcasts). We consider this table implemented
Algorithm 1: The reservation and channel access procedure of node $B$, $\lambda_B$ has been assigned by some reservation stride assignment rule, as discussed in Section 4.3.3.

```
1: for each slot $(i, u)$ do
2:   switch $(i, u)$ is marked in its reservation table as
3:     case DCS$_{B\rightarrow A}(i, u)$:
4:       DCSTransmit($B \rightarrow A$)
5:     case DCS$_{A\rightarrow B}(i, u)$:
6:       DCSReceive($A \rightarrow B$)
7:     case RCS$_{A}(i, u)$:
8:       RCSReceive($A \rightarrow B$) /*record overheard reservation elements in its reservation table*/
9:     case RCS$_{B}(i, u)$:
10:    ReserveRCS($F_{i+\lambda_B}$) /*reserve an interference-free RCS within $F_{i+\lambda_B}$ based on entries in the reservation table (Section 5.2 in Chapter 5)*/
11:   ReserveDCS($F_{i+\lambda_B}$) /*reserve interference-free DCSs dynamically for real-time traffic over all outgoing links based on entries in its reservation table (Section 5.2 in Chapter 5)*/
12:  RelayCS() /*relay overheard reservation elements to appropriate regions (Section 4.3.3)*/
13:  default:
14:   Sleep() /*sleep until the end of the slot*/
```

as a ring buffer, where each entry points to a linked list of reservation elements (either RCS or DCS) that correspond to this entry.

Algorithm 1 shows the main algorithm, by which each node manages the reservation and channel access decisions based solely on the contents of its reservation table. At each slot, a node performs one of the following actions:

- data transmission (using a DCS) to a neighboring receiver (lines 3-4);
- data reception (using a DCS) from a neighboring sender (lines 5-6);
- listening to the channel in order to overhear one of its neighbor’s RCS broadcasts and recording any overheard new reservations using its reservation table (lines 7-8);
- making reservations (using an RCS) for future RCSs and DCSs by selecting
available slots based on its reservation table and relay overhead reservations to nodes in their respective interference sets (lines 9–12); and

- enter a low-power sleep mode otherwise (lines 13–14). As an enhancement, the node may determine the next slot where one of aforementioned cases will be true and sleeps until then.

When a node starts to run the algorithm, it has been assigned a reservation stride by some reservation stride assignment rule to guide its reservation behaviors. Some concrete reservation stride assignment rules are discussed in Section 4.3.3. In Algorithm 1, any specific reservation relay policies can function in the place of RelayCS and any specific slot reservation policies can function in the place of ReserveRCS and ReserveDCS. A concrete reservation relay policy is discussed in Section 4.3.3 and a concrete slot reservation policy for real-time traffic is discussed in detail in Chapter 5.

4.3.3 Reservation Stride Assignment Rule and Reservation Relay Policy

4.3.3.1 Ideal Case

Ideally, if every interference-free communication succeeds, the reservation relay policy is simple for a node:

- It needs to relay each newly overheard reservation element only once and the element will be reliably received by its neighbors.

- It needs to only relay a reservation element that has not expired and whose transmitter r-interferes with the node (since reservations made by non-r-interfering nodes cannot be conflicting with one another).
In this ideal case, reservation elements can be propagated reliably from one node to another via the shortest path and the propagation latency (in units of frames) in the worst case is equal to the number of hops of the shortest path. Therefore, we have the following theorem (with the proof provided in Appendix B):

**Theorem 1.** Assuming that every interference-free communication succeeds, if
\[ \lambda_D - \lambda_A \geq H_{D \rightarrow A} \]
for every \( D \rightarrow A \) with \( \lambda_D > \lambda_A \), where \( H_{D \rightarrow A} \) is the smallest number of hops from \( D \) to \( A \), then DRAMA achieves interference-free slot reservations and channel access.

To assign stride values to r-interfering nodes according to Theorem 1, we rely on a distributed reservation stride assignment protocol based on DRAND [58]. In DRAND, nodes select their individual color (representing a stride in our case) in a serialized order among their neighbors. Based on this idea, nodes in DRAMA select their reservation strides that satisfy Theorem 1. By the protocol one node’s reservation stride can be changed at a time while keeping all other nodes’ reservation strides unchanged, since the protocol serializes r-interfering nodes’ decisions. The concrete reservation stride assignment protocol is discussed in detailed in Appendix A.

### 4.3.3.2 Realistic Case

Compared to the aforementioned ideal scenario, in a realistic environment, some nodes may not be able to successfully overhear their neighbors’ RCS broadcasts, since not every interference-free communication will succeed (e.g., due to unpredictable noise, broken links, etc.). As a result, continuing the example from before, some reservation elements reserved by D cannot propagate to node A in time (i.e., \( \delta_{D < A} > \Delta_{D < A} \)) or they may even be lost. We therefore propose two
methods to improve the propagation robustness: (i) RCS rebroadcast in the side of reservation propagation latency control and (ii) reservation strides adjustment in the side of reservation stride assignment rule.

Using Algorithm 1, each node is able to implicitly acknowledge received RCS broadcasts by relaying the corresponding RCS reservation elements in its next RCS broadcast. For example, suppose that reservation element RCS\(_D(i + \lambda_D)\) is broadcast by RCS\(_D(i)\), which is overheard by D’s neighbor A. The next RCS (say RCS\(_A(i')\)) of node A will relay reservation element RCS\(_D(i + \lambda_D)\), which will be overheard by node D. The detection of reservation element RCS\(_D(i + \lambda_D)\) embedded within RCS\(_A(i')\) indicates node A’s acknowledgement of broadcast RCS\(_D(i)\). Note that \(t' = t\) in the best case where RCS\(_A(i)\) is after RCS\(_D(i)\), or \(t' = t + 1\) in the worst case. If an RCS broadcast has not been acknowledged by a neighbor, it will be rebroadcast. As a result, reservation elements of node D can be more reliably propagated to node A. However, \(\delta_{D\prec A}\) may be prolonged and exceed \(\Delta_{D\prec A}\).

To achieve timely reservation propagation from D to A, the prolonged reservation propagation latency \(\delta_{D\prec A}\) may necessitate a reservation stride adjustment. In this case, node D triggers the distributed reservation stride assignment protocol to increase \(\lambda_D\) such that \(\delta_{D\prec A} \leq \lambda_D - \lambda_A\) still holds. The two aforementioned methods can be adaptively chosen according to collected propagation timeliness statistics. For example, if A has not received reservation element RCS\(_D(i + \lambda_A)\) before F\(_i\) starts, A counts this as a late propagation and reports the resulting timeliness statistics to D. Nodes along the path from D to A can choose to use the aforementioned two methods in combination or alone to improve the propagation robustness according to the timeliness statistics, thereby adapting to link qualities.
and node densities. Both of these methods increase the propagation reliability and
timeliness at the cost of communication overhead (we show their effectiveness and
overhead under various link qualities and node densities in Section 7.4 in Chapter
7). In addition to these two proposed methods, other techniques (e.g., as proposed
in [29, 55, 57]) to improve propagation reliability and timeliness can be added.

4.4 Discussion

We will discuss bootstrapping (Section 4.4.1), recovery (Section 4.4.2), bidi-
rectional reservations (Section 4.4.3), and frame and slot size selection (Section
4.4.4) of the orthogonal, serialized reservation scheme in this section.

4.4.1 Node Joining

A node joins DRAMA (e.g., at the time of deployment and restart) via a multi-
stage procedure, which basically is a combination of the procedures described in
prior sections. First, the node listens to its neighbors to synchronize its clock
and build its interference map by communicating with its neighbors via MCSs. It
also needs to indicate the intention to join the network to its r-interfering nodes
by transmitting during MCSs and waiting to hear acknowledgments from its r-
interfering neighbors. The purpose of acknowledgments is to prevent concurrent
join operations from other nodes. The node also needs to allocate a reservation
stride for itself by triggering the distributed stride allocation protocol (as discussed
in Section 4.3 and Appendix A). Then, the node listens to its neighbors’ RCSs for
at least $\lambda_{\text{max}}$ frames (Section 4.3), where $\lambda_{\text{max}}$ is the maximum reservation stride
among its r-interfering nodes, before it makes any reservations. Since the non-
expired reservations of the node’s r-interfering nodes can span within at most $\lambda_{\text{max}}$
frames, by listening for $\lambda_{\text{max}}$ frames, the node can obtain all current reservations of its r-interfering nodes and learn exactly which slots within the current frame are reserved by whom. Next, the node starts to reserve its initial RCSs. Toward this end, it chooses an un-reserved slot during the current frame as its own RCS and to reserve a future RCS within the $\lambda$th future frame. It does so for $\lambda$ consecutive frames and therefore reserves $\lambda$ future RCSs, one in each of the $\lambda$ consecutive future frames. These initially reserved RCSs will be propagated to the node’s r-interfering nodes. From that time on, the node uses these initial RCSs to make further reservations and the join operation is completed. The node indicates the completion to every node which has acknowledged the join request initially.

4.4.2 Recovery

We discuss how DRAMA acts in the face of node failure, link failure, traffic congestion, and RCS propagation failure.

In the case of node failure, all queued packets and the chosen stride value may be invalid. The node needs to re-start the protocol processing using the join procedure described in Section 4.4.1. During the join procedure, this node is not available for traffic delivery. In the case of link failure (e.g., a neighbor crashed or the link to a neighbor is severely interfered), the node will experience frequent retransmissions and trigger a traffic re-routing to circumvent these failed neighbors or failed links. In the case of traffic congestion where the number of available slots is insufficient for the traffic demanded, packets will be dropped locally and a reroute or multiple path load balance will be triggered to the upper layer. An RCS propagation failure is observed when the node overhears conflicting reservations that are made by two r-interfering nodes. In the case of RCS propagation failure,
the reservation stride values and the RCS propagation policy need to be adjusted as in Section 4.3.3.2.

4.4.3 Bidirectional Reservations

The orthogonal and serialized reservation scheme described previously only supports reservations of unidirectional DCS. To support reservations of bidirectional DCS, we need only to make appropriate modification to the definition of r-interference (Definition 1) and the structure of reservation strides (Theorem 1). Algorithm 1 is still valid.

Since a bidirectional DCS reservation is interfered if either end of the DCS communication is interfered, the definition of r-interference is modified as

**Definition 2. R-Interference for bidirectional communications.** Nodes $A$ and $B$ r-interfere with each other, denoted by $A \leftrightarrow B$, if $(\mathbb{I}_B \cap (\bigcup_{C \in N_A} N_C)) \cup (\mathbb{I}_A \cap (\bigcup_{C \in N_B} N_C)) \neq \emptyset$.

As in the definition, the r-interference for bidirectional communications has a broader range than that for unidirectional communications.

The structure of reservation strides needs to be modified for bidirectional communications. In the serialized and orthogonal reservation scheme (Section 4.3), reservation elements are propagated in one way, i.e., from nodes with greater reservation strides to nodes with smaller reservation strides. Nodes with greater reservation strides may not know the reservations made by nodes with smaller strides. Therefore, they cannot transmit in the slots which they have not reserved since these slots may be reserved by other nodes with smaller strides. As a remedy, we need a two-way reservation propagation scheme so that nodes with greater reservation strides also know the reservations made by nodes with smaller strides.
To achieve this goal, we need to enlarge nodes’ reservation strides to accommodate the additional reservation propagation time in the other way (i.e., from nodes with smaller strides to nodes with greater strides). Suppose $\delta_{\text{worst-case}}$ is the worst-case propagation time (in units of frames) for a reservation element of a node to all of its $r$-interfering nodes. Then the node’s reservation stride must be no less than $\delta_{\text{worst-case}}$ to ensure all of its reservations will be known by all of $r$-interfering nodes before actual communications of any reservation take place. Adding this constraint to the structure of reservation stride (Theorem 1) will achieve this goal and the modified theorem is shown as follows

**Theorem 2.** Assuming that every interference-free communication succeeds, if $\lambda_D - \lambda_A \geq H_{D \to A}$ for every $D \leftrightarrow A$ with $\lambda_D > \lambda_A$ (where $H_{D \to A}$ is the smallest number of hops from $D$ to $A$) and $\lambda_B \geq \delta_{\text{worst-case}}$ for every node $B$ (where $\delta_{\text{worst-case}}$ is the worst-case reservation propagation latency (in units of frames) for a reservation element of node $B$ to all of its $r$-interfering nodes, then DRAMA achieves interference-free slot reservations and channel access in a bidirectional communication environment.

4.4.4 The Sizes of Slot and Frame

The size of a single slot is chosen according to the traffic and application requirements. An event communication network may choose small slot size to reduce unused time within individual slots. A data communication network may choose large slot size to reduce packet fragmentation and packet encapsulation overhead. Discussion on this topic can be found in literature (e.g., [70]). The frame size affects how quickly nodes react to network dynamics. A larger frame size may reduce bandwidth overhead at the cost of slow reaction to network dynamics. With
a larger frame, the number of RCSs in a certain time interval decreases (although the size of individual RCSs may increase), the aggregate bandwidth overhead may decrease slightly. Since a node can only react to network dynamics by adjusting its reservations within the frame a certain number of frames ahead, a larger frame is less responsive to network dynamics. However, frame size has little impact on the real-time performance (e.g., the ratio of packet deadline meets), since nodes in DRAMA can reserve any slot for their communications according to the need of real-time traffic, as supported by our experiments reported in Chapter 7.

4.5 Summary

We use a flexible frame structure and the concept of reservation stride to obtain the orthogonal and serialized reservations among interfering nodes, while retaining the dynamic and progressive reservation idea. The serialized and orthogonal reservation scheme achieves partial of the design requirements stated in Chapter 1:

- It achieves interference-free reservations in a distributed and scalable manner since interfering nodes’ reservations are serialized and each node makes reservations solely based on its own reservation table.

- It has a high bandwidth utilization since all slots can be reserved within each interference region.

- It has a light bandwidth overhead since all reservations are negotiation-free and reservations are only propagated to their respective interference regions.

- It reacts to network dynamics rapidly since interfering nodes’ reservations are orthogonal and negotiation-free.
However, the orthogonal and serialized reservation scheme only provides the mechanism to support real-time communications in a multihop wireless network. The concrete real-time reservation policy is needed (using the orthogonal and serialized reservation scheme) to reserve an appropriate number of slots at appropriate positions, in response to changing network and traffic conditions. A concrete real-time reservation policy is discussed in next chapter.
CHAPTER 5

REAL-TIME RESERVATION POLICY

Due to the fact that wireless links are not always stable, previously reserved slots based on past link quality may not be sufficient to transmit the required amount of data in time. Furthermore, some reserved communications may fail at some intermediate nodes and retransmissions may increase end-to-end latencies and result in end-to-end deadline misses. For these reasons, nodes need to dynamically decide how many and which slots to reserve to meet the timeliness requirements of queued and future packets. In this section, we first present the real-time traffic model we base our work on, and then introduce a dynamic reservation policy (using the orthogonal and serialized reservation scheme in Chapter 4) and its accompanying admission control mechanism. At the end of this chapter, we also discuss how to support both real-time and best-effort traffic in Section 5.4.1.

5.1 Real-Time Traffic Model

We assume that a real-time stream has already been decomposed into a collection of periodic reservation tasks (or shortly reservation tasks or just tasks) by some end-to-end decomposition technique (e.g., [7, 25, 43, 59]). Each reservation task corresponds to a single hop traffic delivery of a stream along the stream path.
Figure 5.1. An illustration of the timeliness of the first three packets (in case (a), (b), and (c), respectively) of a stream going from the source node A to B, C, D, and to the destination D. Both the minimum packet inter-release interval of the stream and the local deadline at each hop is 4 time units. The timeliness parameters of each packet at each node is denoted as (expected arrival or release time)/(actual arrival or release time)/(absolute local deadline) and are shown below each hop. The expected arrival or release time of each packet are shown in the case that the packet is a future packet. The expected arrival or release time of each packet in the case that the packet has been received is shown. It is simply the absolute deadline of the packet at the previous hop or the actual release time at the source. The three cases (a), (b), and (c) show three different packet delivery scenarios: right on time, deadline miss, and deadline meet, respectively.

The reservation task for the traffic delivery of stream $S_k$ over $A \rightarrow B$ is denoted as $S^A\rightarrow B_k$ with period (or minimum packet inter-release interval) $p^A\rightarrow B_k$, transmission time $e^A\rightarrow B_k$ per period, and local deadline $d^A\rightarrow B_k$. To simplify the presentation, we refer to all timing parameters in terms of the number of slots based on a default transmission rate. Collectively, we use $S_k = \{\cdots, S^A\rightarrow B_k, \cdots\}$ to represent the end-to-end decomposition of $S_k$.

Then, it is DRAMA’s responsibility to ensure that each node along the path of a real-time stream makes slot reservations that meet the timeliness needs of
the stream. The model described here is a very common and general model to
describe real-time workload [40, 41], allowing us to reuse existing schedulability
analysis techniques.

We assume that each node schedules queued packets according to the earliest
deadline first (EDF) scheduling policy [41]. Failed transmissions can be detected
via missing acknowledgments (e.g., an acknowledgement can be returned within
the same slot for a bidirectional DCS communication or otherwise piggybacked
onto an MCS or RCS for a unidirectional DCS communication) and retried for a
certain number of times.

To facilitate the EDF scheduling, the absolute deadline of each queued packet
at each hop must be known. For this purpose, an absolute deadline field is embed-
ded into each data packet like Delay EDD [15] or Jitter EDD [69]. The absolute
deadline of a packet at a source is equal to the release time of the packet plus
the local deadline of the reservation task. The absolute deadline of a packet at
the $j^{th}$ hop ($j = 1$ for the source) is then the release time of the packet at the
source plus the sum of all $j$ local deadlines. To facilitate the computation of the
absolute deadlines, each node increases the absolute deadline field of each packet
upon reception by the local deadline of the reservation task at this node.

To facilitate each node’s reservation, the expected arrival times and the trans-
mission times of both received and future packets must be known. The trans-
mission time of a future packet is conservatively estimated to be the transmission
time of the corresponding reservation task. The expected arrival times of received
and future packets are computed in different ways. The expected arrival time of
a released packet at a source is equal to the release time. The expected arrival
time of a received packet at an intermediate node is equal to the value of the
absolute deadline field of the packet upon reception (prior to the increase of this field by this intermediate node), i.e., the expected arrival time of a packet at an intermediate node is simply the absolute deadline of the packet at the previous hop. The expected arrival time of the \(i\)th future packet is \(i\) periods of the reservation task corresponding to the packet far from the expected arrival time of the latest received (or released at the source) packet of the same stream. The absolute deadline of \(i\)th future packet at a node is equal to the expected arrival time of the packet plus the local deadline of the reservation task corresponding to the packet.

Figure 5.1 illustrates the local timeliness of the first three packets of a real-time stream at each hop. From this figure, we can see the actual arrival time of a packet be earlier or later than either its expected arrival time or even the absolute deadline without appropriate slot reservations.

5.2 Dynamic Real-Time Reservations

The previous chapter described how slots have to be reserved to ensure interference-free communications. Now we describe how slots are reserved to ensure real-time communications. Within each RCS, a node makes reservations for future RCSs and DCSs as described in lines 9-12 of Algorithm 1 in Chapter 4. When making reservations, a node first selects slots for a future RCS before making DCS reservations. This is because RCSs carry reservation information that is critical for future reservations of other nodes (which will overhear these RCSs). Next, a node selects as many slots as necessary for its DCS communications to satisfy the bandwidth needs of its real-time streams. This section describes this process in more detail and the algorithm is shown in Algorithm 2.

Assume a node \(B\) that uses its RCS\(_B(i, u)\) to make new reservations within
Figure 5.2. An illustration of dynamic reservations of node B ($\lambda_B = 2$) over $B \rightarrow C$ with reservation tasks $S_{1B \rightarrow C}$ and $S_{2B \rightarrow C}$. For simplicity, we only plot relevant reservations in node B’s reservation table, and assume that the actual or expected size of each packet is one slot and that link qualities are perfect. $DCS_{B \rightarrow C}(i + 2)$ represents the regions of slots within $F_{i+2}$ which can be reserved over $B \rightarrow C$ without conflicting with entries in B’s reservation table. The number of reserved and demand slots before the first region of $DCS_{B \rightarrow C}(i + 2)$ is 2 and 4, respectively. Therefore, B reserves 2 slots in the first region to satisfy the timeliness and load requirement of $S_{1B \rightarrow C}$ and $S_{2B \rightarrow C}$. There are 2 future packets arriving after the first region and accordingly node B reserves 2 slots in the second region.

$F(i + \lambda_B)$. Node B chooses the earliest available slot for its new RCS reservation (where slot availability can be obtained from the reservation table). Next, node B will select slots for its DCS communications for all of its real-time streams. As
Algorithm 2 Real-time reservations of node B at every RCS$_B(i, u)$ under traffic and link dynamics

1: **ReserveDCS**(F$_{i+\lambda_B}$):
2:   for each outgoing link $B \to C$ do
3:     for each CS($i + \lambda_B, v$), $0 < v \leq |F|$ do
4:       /*Demand$_{B \to C}(i + \lambda_B, v)$ is the number of demanded slots for all queued or future packets whose expected arrival times are before slot $(i + \lambda_B, v)$*/
5:       /*Supply$_{B \to C}(i, u; i + \lambda_B, v)$ is the number of reserved slots by node B over link $B \to C$ within interval $[(i, u), (i + \lambda_B, v)]$*/
6:       if Demand$_{B \to C}(i + \lambda_B, v) >$ Supply$_{B \to C}(i, u; i + \lambda_B, v)$ then
7:         Reserve DCS$_{B \to C}(i + \lambda_B, v)$

Illustrated in Figure 5.2, node B (using RCS$_B(i)$) makes DCS reservations within F$_{i+\lambda_B=i+2}$ for S$_1^{B \to C}$ and S$_2^{B \to C}$ over $B \to C$. It is up to node B to decide which slots are assigned to a specific stream as long as all streams’ deadlines are met.

For an outgoing link $B \to C$, while using RCS$_B(i, u)$ to reserve DCSs over $B \to C$, B needs to identify the slots within F$_{i+\lambda_B}$ that can be reserved over $B \to C$ without conflicting with existing entries in its reservation table. We denote these available slots as DCS$_{B \to C}(i + \lambda_B)$, which initially includes all slots except MCSs within F$_{i+\lambda_B}$. While making or overhearing a new reservation that conflicts with DCS$_{B \to C}$, B removes the corresponding slots from DCS$_{B \to C}(i + \lambda_B)$. Node B attempts to reserve slots for each packet as early as possible after the packet’s expected arrival time. It examines each slot $(i + \lambda_B, v) \in$ DCS$_{B \to C}(i + \lambda_B)$ in ascending order to check whether this slot should be reserved as DCS$_{B \to C}(i + \lambda_B, v)$. It computes the number of reserved slots over $B \to C$ within $[(i, u), (i + \lambda_B, v)]$, denoted as Supply$_{B \to C}((i, u), (i + \lambda_B, v))$, and the number of demand slots over $B \to C$ by $(i + \lambda_B, v)$, denoted as Demand$_{B \to C}(i + \lambda_B, v)$. Supply$_{B \to C}((i, u), (i + \lambda_B, v))$ is readily available by consulting B’s reservation table. Demand$_{B \to C}(i + \lambda_B, v)$ is equal to the sum of the sizes of all queued or future packets whose expected arrival times are before $(i + \lambda_B, v)$. Note that the size of each queued or future packet is computed using current link quality. If Supply$_{B \to C}((i, u), (i + \lambda_B, v)) <$
Demand\(_{B\rightarrow C}(i + \lambda_B, v)\), slot \((i + \lambda_B, v)\) is reserved as DCS\(_{B\rightarrow C}(i + \lambda_B, v)\), and the reservation table is updated.

5.3 Distributed Admission Control

When a source node initiates a new real-time stream (e.g., by performing a route discovery process), an end-to-end admission control mechanism is needed to ensure that the stream’s deadlines can be met without violating the timeliness needs of existing streams. We assume that end-to-end admission control is the responsibility of the routing layer (as admission control is often tightly integrated with route discovery).

However, DRAMA lies at the medium access control layer and can only be responsible to provide the routing layer with any information needed to make admission decisions. Specifically, DRAMA must provide the upper layers with a local admission control mechanism that checks whether a given reservation task of a new stream is locally admissible. The new stream is end-to-end admissible if all of its reservation tasks are locally admissible. The local admission control mechanism is summarized in Algorithm 3 and explained in the following paragraphs. The basic idea of Algorithm 3 is to treat the packets at a node as reservation tasks (see Section 5.1) and perform real-time schedulability analysis among these reservation tasks.

The local admission control is executed by local-admission-test at node \(A\) for a new reservation task \(S_k^{A\rightarrow B}\). Since the introduction of \(S_k^{A\rightarrow B}\) may affect already admitted reservation tasks of \(A\)’s r-interfering nodes, these reservation tasks must also be re-checked (lines 3–5) to ensure they are still satisfiable (i.e., the required number of interference-free slots in every period can be successfully reserved before
Algorithm 3 Local admission control for $S_k^{A \rightarrow B}$ of a new stream $S_k = \{ \cdots, S_k^{A \rightarrow B}, \cdots \}$. $S_k$ is end-to-end admissible if every $S_k^{A \rightarrow B}$ is locally admissible.

1: /*check whether $S_k^{A \rightarrow B}$ is locally admissible*/
2: boolean local-admission-test($S_k^{A \rightarrow B}$):
3: for $C \leftrightarrow A$ do
4: if check-r-interfering-node($C$) == false /*assume that $S_k$ were admitted*/ then
5: return false
6: return true

7: /*check whether all admitted reservations tasks of a node are still satisfiable due to the joining of a new stream*/
8: boolean check-r-interfering-node($C$):
9: for each reservation task $S_{C \rightarrow D}^j$ do
10: $T_{C \rightarrow D}^j = \emptyset$ /*initialize the task set corresponding to $S_{C \rightarrow D}^j$*/
11: /*take real-time streams into account*/
12: for each link $E \rightarrow F$ interfering link $C \rightarrow D$ do
13: $\Pi_{E \rightarrow F}^j = (\lambda_E, \pi_E(E \rightarrow F))$ /*$\pi_E(E \rightarrow F)$ is the local link priority of $E \rightarrow F$ of node $E$ among all of its outgoing links*/
14: $T_{E \rightarrow F}^j = (p_{E \rightarrow F}^j, \epsilon_{E \rightarrow F}^j, d_{E \rightarrow F}^j, \Pi_{E \rightarrow F}^j)$
15: $T_{C \rightarrow D}^j = T_{C \rightarrow D}^j \cup T_{E \rightarrow F}^j$
16: /*take RCS traffic into account*/
17: for each RCS $E$ with which $C \rightarrow D$ may interfere do
18: $T_{C \rightarrow D}^j = T_{C \rightarrow D}^j \cup T_E$
19: /*take MCS traffic into account*/
20: $T_0 = (F, |MCS|, F, (+\infty, +\infty))$ /*$|MCS|$ is the number of MCS slots per frame*/
21: $T_{C \rightarrow D}^j = T_{C \rightarrow D}^j \cup T_0$
22: $w_{C \rightarrow D}^j = \text{CalcWCRspTimeFixedPriority}(T_{C \rightarrow D}^j, T_{C \rightarrow D}^j)$ /*compute the worst-case response time $w_{C \rightarrow D}^j$ of $T_{C \rightarrow D}^j$ in $T_{C \rightarrow D}^j$ using the fixed-priority worst-case response time analysis in [20]*/
23: if $w_{C \rightarrow D}^j > d_{C \rightarrow D}^j$ then
24: return false
25: return true

The function check-r-interfering-node at node $C$ performs real-time schedu-
lability analysis for each reservation task $S_{j}^{C\rightarrow D}$ (lines 9–23). To check whether $S_{j}^{C\rightarrow D}$ is still satisfiable, we rely on two observations. One observation is that we only need to consider the set $T_{j}^{C\rightarrow D}$ of reservation tasks interfering with $S_{j}^{C\rightarrow D}$, which includes those corresponding to real-time streams (lines 11–15) and those corresponding to RCS (lines 16–18) and MCS (lines 19–20). The other observation is that these reservation tasks can be considered as being scheduled by a fixed-priority scheduler, where tasks’ priorities are predefined and higher-priority tasks are favored while contending for the same slot. Their reservation behaviors manifest their compliance to the fixed-priority scheduling policy: (i) nodes with greater reservation strides have a higher priority for reserving slots (see Section 4.3 in Chapter 4) than nodes with smaller strides, and (ii) the priorities of reservation tasks of the same transmitter take a link-by-link priority order (see Section 5.2) and thus all reservation tasks inherit the priorities of their associated links.

For these reasons, each reservation task $S_{i}^{E\rightarrow F} \in T_{j}^{C\rightarrow D}$ is augmented to the tuple $(p_{i}^{E\rightarrow F}, e_{i}^{E\rightarrow F}, d_{i}^{E\rightarrow F}, \Pi_{i}^{E\rightarrow F})$. The priority $\Pi_{i}^{E\rightarrow F}$ is expressed as a pair $(\Pi_{i}^{E\rightarrow F}(1), \Pi_{i}^{E\rightarrow F}(2))$, where $\Pi_{i}^{E\rightarrow F}(1)$ represents the reservation priority of node $E$ among its $r$-interfering nodes and $\Pi_{i}^{E\rightarrow F}(2)$ represents the reservation priority of link $E \rightarrow F$ among all outgoing links of node $E$. We let $\Pi_{i}^{E\rightarrow D}(1)$ be equal to $\lambda_{E}$ and $\Pi_{i}^{E\rightarrow F}(2)$ be equal to the link order value $\pi_{E}(E \rightarrow F)$ of $E \rightarrow F$ by node $E$. Therefore, for priorities $\Pi$ and $\Pi'$, priority $\Pi$ is higher than priority $\Pi'$ if $\Pi(1) > \Pi'(1)$ or $\Pi(1) = \Pi'(1)$ and $\Pi(2) < \Pi'(2)$. The reservation task corresponding to an RCS is assigned the highest priority among all reservation tasks at the transmitter of the RCS since it is critical in achieving reliable and timely reservation and propagation. The reservation task corresponding to the MCS is assigned the highest priority among all reservation tasks in the network.
since it is pre-allocated and shared by all nodes.

The function `check-r-interfering-node` also invokes a fixed-priority response time analysis routine (e.g., \[20, 41\]) with \(T_{k}^{A\rightarrow B}\) as input to compute the worst-case response time \(w_{k}^{A\rightarrow B}\) of \(S_{k}^{A\rightarrow B}\) (lines \[21, 23\]). If some other reservation tasks have the same priority as \(S_{k}^{A\rightarrow B}\), the tie is broken in their favor. This tie-break rule guarantees that all of the reservation tasks over the same link can be scheduled without missing any deadline by any work-conserving policy (EDF in our case, see Section \[5.2\]) if all of them pass the aforementioned worst-case response time test \[41\]. If \(w_{k}^{A\rightarrow B} \leq d_{k}^{A\rightarrow B}\), then \(S_{k}^{A\rightarrow B}\) is still satisfiable after the introduction of \(S_{k}^{E\rightarrow F}\); otherwise, the local admission control fails.

The aforementioned admission control mechanism is summarized as a theorem below (with the proof provided in Appendix \[C\]).

**Theorem 3.** Assume that every interference-free communication succeeds. For a given new stream \(S_{k} = \{\cdots , S_{k}^{A\rightarrow B}, \cdots \}\), if Algorithm \[3\] returns true with every input \(S_{k}^{A\rightarrow B}\) of \(S_{k}\), then \(S_{k}\) is end-to-end admissible.

We summarize the admission control from the global point of view, as illustrate in Figure \[5.3\] where \(S_{1}\) goes through nodes A, B, C, D, and E with \(\lambda_{A,B,C,D,E} = \{1, 2, 3, 4, 5\}\). In the first step as shown in (a), \(S_{1}\) is decomposed into periodic reservation tasks, one over each forwarding link. In the second step as shown in (b), each periodic reservation task is expanded into a group of fixed-priority periodic reservation tasks (higher priority tasks proceeds lower priority tasks in each task set) which interfere with it, including implicit periodic reservation tasks related to RCS and MCS as well as explicit periodic data reservations tasks. In the last step as shown in (c), the classical fixed-priority schedulability test is applied on each generated group in the second step. If all of the expanded task sets pass
Figure 5.3. An illustration of the end-to-end admission control of periodic real-time stream $S_1$.

the test, $S_1$ is end-to-end admissible.
5.4 Discussion

5.4.1 Support Best-Effort Traffic

DRAMA can support best-effort traffic in a reservation-based or a non-reservation-based (i.e., contention-based) manner along with real-time traffic.

To support unpredictable best-effort traffic in a reservation-based manner, nodes reserve slots for packets only when the packets have arrived or the packets’ loads and arrival times are known. Due to RCS reservation propagation, downstream nodes can know the load and arrival time of imminent traffic from upstream nodes. Therefore, unlike the store-forward procedure in which packets contend for the medium after their arrival, downstream nodes in DRAMA can make reservations for these packets in advance. Consequently, the reservations are pipelined along the traffic path, which greatly reduces the end-to-end latency without additional energy consumption.

To support best-effort traffic in the non-reservation-based manner, we need the bidirectional reservations in Section 4.4.3 in the first place. The unidirectional reservations enable a node to know if the current slot has been reserved by itself or its r-interfering nodes whose reservation strides are greater than its. It does not enable the node to tell whether the slot is reserved by any of its interfering nodes. Therefore, if the node attempts to transmit using this slot, a collision may happen. However, bidirectional reservations allow a node to tell who has reserved the current slot, be itself, some of its interfering nodes, or nobody. If none has reserved this slot, the node can use this slot to transmit data traffic in a contention-based manner.

A node can indicate to the potential receivers (using its RCS) its intention to start and stop using this kind of non-reservation-based communication. Once a
node has been indicated by one of its neighbors (which is the transmitter) as a potential receiver, it can wake up in slots during which no reserved communications can interfere with its reception from the transmitter. Since the non-reservation-based communication is contention-based and opportunistic, it provides a means to transmit bursts and unpredictable best-effort traffic in a timely manner at the cost of additional energy consumption.

5.5 Summary

We propose a real-time reservation policy for a typical real-time traffic model, exploiting the underlying orthogonal and serialized reservation scheme. This reservation policy is able to reserve an appropriate number of slots at appropriate positions, according to traffic timeliness, load demand, and network conditions. The reservation policy is computationally simple and manifests that the orthogonal and serialized reservation scheme is able to support various traffic patterns and provide various qualities of service.

An accompany distributed admission control is proposed along with the real-time reservation policy. Since the orthogonal and serialized reservation scheme naturally introduces priorities among interfering nodes for reserving slots, the admission control can be transformed into the classic periodic real-time schedulability test by taking traffic interference into account.
CHAPTER 6

PROTOTYPE IMPLEMENTATION DESCRIPTION

As shown in the proceeding chapters, the correct working of DRAMA relies on other auxiliary components. First, it needs to synchronize neighboring nodes to common clock so nodes know when to wake up at a slot boundary to be ready for transmission and reception and when it is safe to sleep. Second, DRAMA needs a link monitor to tell the transmission rate and packet reception ratio of a link, based on which a node can reserve slots accordingly. Third, DRAMA needs the interference manager to tell which node’s transmission interferes with which other node’s reception, based on which the r-interference relationship is established. In this chapter, we discuss our implementation of these components and DRAMA itself in a specific hardware and software platform, which brings both disadvantages and advantages to their implementations.

We first briefly introduce the hardware and software platform we use in Section 6.1. Then, we discuss our implementation of the auxiliary components and DRAMA itself in Section 6.2. The detailed design and implementation are discussed in Appendix D. Further, we present some lessons and experiences we learn from the prototype implementation in Section 6.3.
6.1 Platform

The Intel Mote2 hardware platform we use delivers a high level of integration as well as low-power operation in a small physical size. It uses PXA271 XScale Processor \[30\] and has 32MB flash and 32MB SDRAM on board. It has a more powerful processor and larger memory than other typical motes (e.g., Mica \[24\]). We choose this platform mainly since it has a sufficient memory to store a reservation table. The platform uses a ZigBee (IEEE 802.15.4 \[27\] radio ChipCon CC2420 \[62\] from TI, which can be powered on/off in less than several milliseconds.

We use the TinyOS operating system \[23\], which has one thread, one stack in the system. TinyOS is a free and open source component-based operating system and platform targeting wireless sensor networks. TinyOS is an embedded operating system written in the nesC programming language \[11\] as a set of cooperating tasks. TinyOS programs are built out of software components, some of which present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage.
TinyOS is completely non-blocking: it has one stack. Therefore, all I/O operations that last longer than a few hundred microseconds are asynchronous and have a callback. To enable the native compiler to better optimize across call boundaries, TinyOS uses nesC’s features to link these callbacks, called events, statically. While being non-blocking enables TinyOS to maintain high concurrency with one stack, it forces programmers to write complex logic by stitching together many small event handlers. To support larger computations, TinyOS provides tasks, which are similar to a deferred procedure call and interrupt handler bottom halves. A TinyOS component can post a task, which the OS will schedule to run later. Tasks are non-preemptive and run in FIFO order. This simple concurrency model is typically sufficient for I/O centric applications, but its difficulty with CPU-heavy applications and . TinyOS code is statically linked with program code, and compiled into a small binary, using a custom GNU toolchain. Associated utilities are provided to complete a development platform for working with TinyOS.

The hardware and software platform presents some challenges to our prototype implementation, mainly due to the following reasons:

- The purpose of our prototype implementation is to provide real-time communication to the network, which necessitates individual nodes to perform all of their event processing in a real-time manner. For example, if a node can not catch up to transmit packets in a slot it has reserved previously due to prolonged running of other functionalities, the neighbor at the next hop can not receive the packets in time and may lead to missing their end-to-end deadlines. The single-thread and single-stack operating system running environment further exacerbates the problem since there is no way to wake up a separate thread to handle an urgent event.
We require neighboring nodes to align at slot boundaries for packet transmission and reception. The simple radio, hardware, and software platform require us to exploit every feature to synchronize nodes sufficiently accurately to meet this requirement.

Debugging the protocol implementation is difficult. Logging various events in flash memory often interferes with the regular protocol operation and occasionally makes the system miss some reception and transmission deadlines. Moreover, detecting the failure or fault causes of the protocol requires correlating multiple nodes’ operations, which demands a special attention and design.

6.2 Implementation

We describe our implementation of time synchronization, interference management, link monitor, and special design for real-time event processing in this section. A more detailed design description on TinyOS is presented in Appendix D.

6.2.1 Time Synchronization

DRAMA coordinates neighboring nodes’ radio activities in a time-slotted manner. For this purpose, it has to (i) synchronize neighboring nodes’ clocks and align them at slot and frame boundaries and (ii) power on and off radios accurately at slot boundaries.

We augment RCS with a timestamp (Figure 4.1 in Chapter 4) to ease achieving time synchronization among neighbors since each node can overhear its neighbors’ RCS without interference. Each RCS has four fields related to time synchro-
organization: transmitter identifier, current frame index, current slot index, and the
timestamp at transmitting instance. A node can synchronize its clock based on
the timestamp in the received RCS, as in [47, 66], to the clock of the transmitter
of the RCS. The synchronization accuracy is less than 100 microseconds in our
experiments.

Moreover, the node can estimate the slot duration and the frame size as well.
Suppose that the slot indices and the timestamps of three consecutive RCSs re-
ceived from the transmitter are $I_1, I_2, I_3$ and $T_1, T_2, T_3$, respectively. Let $S$ and $F$
be the duration of a single slot and the number of slots per frame, respectively.
We then have the equations $(I_2 - I_1 + F) \times S = T_2 - T_1$ and $(I_3 - I_2 + F) \times S = T_3 - T_2$.
From these equations, we can derive the values of both $S$ and $F$. When more data
is available, we can use some filtering and regression techniques to obtain more
accurate synchronization (e.g., [12]). The start times of frames can be estimated
as well, i.e., the start time of the immediate upcoming frame is just $(F - I) \times S$
ahead, where $I$ is the slot index of the RCS just received.

We use a 32KHz precision timer [49] to indicate the transmission/reception
start times at slot boundaries. In order for a receiver to catch up with a trans-
mitter, the receiver must be active no later than the start frame delimiter (SFD),
whose duration is 128$\mu$s for CC2420 [65] in our experiment. We have experimen-
tally observed that the maximum jitter among the transmitter and the receiver is
less than this value, and it is extremely rare that the receiver misses the transmit-
ter when it becomes active tens of microseconds earlier than scheduled wake-up
times. To avoid the occasional overlapping of two interfering transmissions at
a slot boundary due to small synchronization inaccuracy, a guard region, whose
length is equal to the maximum synchronization inaccuracy, is used at end of each
slot. Nodes avoid transmitting packets whose transmissions may fall within the guard region.

6.2.2 Interference Management

The basic idea to determine which node’s transmission interferes with which other nodes’ reception is to correlate transmission and reception records. We currently using MCSs to exchange interference-related information. To accelerate the exchange of these information, we may use RCSs to carry them.

We suppose each node only needs to collect the SINR/PRR (signal-to-interference-and-noise-ratio/packet-reception-ratio) data from nodes within its 3-hop distance. There are two forms of the collected data: transmitter data and receiver data. Each transmitter data record is in the form of (transmitter ID, transmitter power level, timestamp, duration). The timestamp is sampled at the time of SFD (start of frame delimiter). The duration is the length of the time interval of the corresponding packet transmission. The transmitter ID and the transmitter power level are filled out by the transmitter and can be extracted from the received packet head by the receiver. Each receiver data record is in the form of (receiver ID, RSSI, transmitter ID, timestamp), where RSSI stands for received signal strength indicator and can be extracted at a special hardware register immediately after the completion of the reception. If a node does not receive the packet correctly at the time indicated by the timestamp, the transmitter ID field of the receiver data record is null.

Having both the transmitter data records from its neighbors and the receiver data records from itself, a node can correlate transmitter data records with receiver data records using timestamps and durations fields. If the time interval of a
transmitter data record and the time interval of a receiver data record overlap, these two records are said paired and the resulting paired record is in the form of (transmitters’ IDs, transmitters’ transmission powers, receiver’s RSSI, the ID of the transmitter whose packet is corrected received). Each paired record describes the concurrent transmitters and the receiver and their respective transmission powers and received power at a specific time. We do not need to include timestamp information in paired records and from that step on we do not need timestamp information. Note that if node A transmits from time 2 to time 5 and node B transmits from time 4 to 7, the receiver samples its RSSI at time 4.5, then the three nodes are correlated and the transmitters’ IDs of the paired record consists of the IDs of both A and B.

Having a collection of paired records, a node needs to compute the reception power of individual transmitters on itself for each paired record, i.e., the contributing factor of each transmitter to the received RSSI of each paired record. For example, suppose the received RSSI is 10 at a node, node A is 10 meters away from the node and transmits at power 10DBm, it may contribute 2 units to the RSSI. Another transmitter B is 20 meters away from the node and transmits at power 10DBm, it may contribute 1 unit to the RSSI. However, the RSSI contributions are not proportional to physical communication distance. For example, node C is 5 meters away and transmits at power 20DBm and there is a barrier between node C and the receiver (in this case, the node itself), so its contribution to the RSSI is only 1. The reason of this kind of complication is mainly due to complicated external environment. Therefore, we need a measurement-based approach [42, 46, 50, 53, 61], any analytical approach may fail.

We need to find some special pairs to compute the contributing factors. For
example, if a node is the receiver of two paired records and the sets of the transmitters of the two paired records only differ by a single transmitter. For example, the transmitter set of the first paired record is \( \{A, B, C\} \), and the second is \( \{A, B\} \), then C is a singular transmitter to these two paired records. We can compute the contributing factor of the singular transmitter by differing the RSSIs of the two paired records (e.g., if the RSSI of the first paired record is 10 and the second is 8, then, continuing from the aforementioned example, the contributing factor of the singular transmitter C on the receiver is 2). Such singular transmitters may not exist everywhere, however, we can reduce other complicated paired records to simpler forms, e.g., if we know the combined contribution of node C and D on the receiver is 5, then we can know the contributing factor of node D is 3 (assuming that we already knew the contributing factor of node C is 2). Using these methods, we can also filter out some irrelevant nodes. If a node is transmitting, but the RSSI of a receiver is the same as when the node is not transmitting. Then we can be somewhat sure that the node’s activity is irrelevant to the receiver’s communication activities and these two nodes should be non-interfering.

While computing the contributing factor of each transmitter on the receiver on each paired record on the one hand, we compute the SINR threshold for the receiver on the other hand. The SINR threshold is a watermark such that when the received SINR is greater than it the corresponding packet will be received correctly with probability \( P \) (e.g., 95%), which is specified by applications. We denote the set of paired records sharing the receiver as \( R \). The minimum SINR value that the same receiver can receive packets over \( R \) with probability at least \( P \) is recorded as the SINR threshold for the receiver.

Having both the contributing factors and the SINR thresholds, we now can
determine whether a node’s transmission will interfere another node’s reception. That is actually simple, we compare the estimated SINR (denoted as \( \text{SINR}_{\text{est}} \)) perceived by the receiver when the transmitter is a singleton in a paired record

\[
\text{SINR}_{\text{est}} = \frac{\text{the transmitter’s contributing factor}}{N_0},
\]

(6.1)

where \( N_0 \) is the contributing factor of background noise. If the estimated SINR is less than the SINR threshold, then the transmitter is considered as a singleton interferer of the receiver.

If the transmitters field of the paired record consists of several transmitters, each of them can be considered as a designated transmitter to the receiver, the others are considered as potential interferers. In this case, the \( \text{SINR}_{\text{est}} \) of the transmitter to the receiver is computed as

\[
\text{SINR}_{\text{est}} = \frac{\text{the transmitter’s contributing factor}}{N_0 + \text{sum of interferers’ contributing factors}},
\]

(6.2)

where \( N_0 \) is the contributing factor of background noise. If the estimated SINR is less than the SINR threshold, then these potential interferers are considered as true interferers to the transmitter (i.e., if they transmit simultaneously, the receiver can not correctly decode packets transmitted by the transmitter), based on which we can build the required interference map for a node. Each interference element of the map is in the form of (transmitter, receiver, the set of interferers), where an interferer can be a singleton or a set of nodes which are transmitting simultaneously. For example, (B, A, \{C, D\}) is an interference element in Figure 4.2 in Chapter 4. In realistic networks, non-singleton interferer is rare [42, 80].
6.2.3 Link Quality Monitor

We use a link monitor (as indicated in Figure 3.2 in Chapter 3) at the MAC layer to both provide reliable DCS transmissions and measure link qualities. For reliable unidirectional DCS transmissions, we augment RCS packets of a node to piggyback packet reception acknowledgments to its neighbors (since all of the neighbors of an RCS must listen to it). For reliable bidirectional DCS transmissions, packet reception acknowledgement can be a part of a DCS communication itself. Nodes retry failed transmissions, thereby providing reliable data traffic delivery. Link qualities are measured by packet reception ratios (PRRs). This link quality indicator can be measured either actively by the transmitter or passively by the receiver. If the receiver needs to acknowledge every received data packet in each RCS transmission, the transmitter can actively count the number of attempted and succeeded (i.e., acknowledged) transmissions. Otherwise, the transmitter only knows the number of attempted transmission in a certain time interval (e.g., frames) and therefore the receiver needs to send a feedback (via either MCS or RCS piggybacks) to the transmitter the number of successfully received packets in the time interval. In our implementation, we use the active measurement method and use consecutive 3 frames as the time interval. Other methods to estimate link quality based on SINR (signal to interference and noise ratio) are more popularly used [2, 46, 56]. They can be alternative implementations of the link monitor and still fit within DRAMA.

6.2.4 Real-Time Event Processing

TinyOS uses a single-thread and single-stack programming execution model. The run queue consists of an array of deferred procedures. The thread runs these
deferred procedures in the FIFO order. Each procedure runs to completion (i.e., the processor can not suspend a running procedure in the middle and turns to run other procedures). Only interrupts can suspend a running procedure and after the completion of an interrupt processing the suspended procedure will then resume. A procedure may post another procedure to the tail of the run queue, which will be running later. This execution model imposes difficulty to time-critical operations described in Chapters 4 and 5.

The most time-critical events in our implementation are the RCS broadcasting, RCS overhearing, DCS transmitting, and DCS receiving. Missing those events will reduce the reliability of the protocol and prolong packet delivery latencies. For example, as shown in Figure 6.2 when a node receives an RCS broadcast from one
of its neighbor, which says that the neighbor reserves a future slot for transmitting data to that receiver. Upon processing the received RCS, the node finds that it itself is an intended receiver of a future DCS from that neighbor and immediately posts a deferred procedure to handle this future data reception. If the deferred procedures in the run queue run slowly and have not finished by the time when the reserved DCS expires, then the node will miss the DCS packet reception.

To prevent from missing time-critical events, we organize the run queue as a link list and augment each deferred procedure with its scheduled start time and its worst-case execution time (as shown in Figure 6.3). The deferred procedures in the run queue are linked from the head to the tail in the order of their expected start times. There are some deferred functions whose scheduled start times are afloat. For example, when a node overheard its neighbors’ RCS broadcasts, it may update its reservation table immediately or defer to some later time (but not after its next RCS broadcast since this broadcast may have to relay some of overheard reservations). In this case, we treat the latest time to run the deferred procedure as the nominal expected start time.

The running interval of a deferred procedure is equal to [its scheduled start time, its scheduled start time + its worst-case execution time]. If the running time intervals of two consecutive deferred procedures overlap, i.e., the proceeding procedure may run across the scheduled start time of the following procedure, then we need to adjust the execution order of some deferred procedures or re-design the algorithms and data structures used by these procedures to shorten their worst-case execution times.

First, we aggressively run every deferred procedure whose scheduled start time is afloat, as long as the execution of such a procedure will not run across any
Each procedure is tagged with start time and worst-case execution time \((s, e)\)

- **(a) run queue**
  \[
  \begin{array}{c}
  (1, 3) \\
  \end{array} \rightarrow \begin{array}{c}
  (20, 3)
  \end{array}
  \]

- **(b) run queue**
  \[
  \begin{array}{c}
  (1, 3) \\
  \end{array} \rightarrow \begin{array}{c}
  (10, 2)
  \end{array} \rightarrow \begin{array}{c}
  (20, 3)
  \end{array}
  \]

Post a deferred procedure

Figure 6.3. An illustration of timeliness-aware augmentation of the run queue.

defined procedure whose scheduled start time is fixed. By doing this, we significantly avoid the risks to miss the operation deadlines of these deferred procedure whose scheduled start time is afloat.

Second, we re-design the data structure and the processing procedure of the reservation table. Our basic method is to split the processing of the reservation table into multiple finer-grained procedures, each of them can be deferred by some time and has smaller variation of execution times (i.e., the difference between the worst-case execution time and the best-case execution time). As an example, instead of using one reservation table to contain the reservation information over all adjacent links for a node, we use one reservation table for each adjacent link. When a node tries to reserve slots for a link, the node only needs to check the reservation table of the link instead of scanning the single conglomerate reservation table. By doing so, the total and the variation of execution times of reservation table
processing are significantly reduced. As another example, there is a deferred procedure associated with each link that updates the link’s reservation table. Upon overhearing an RCS broadcast from a neighbor, the node always defers the procedures associated with its adjacent links to a later time. The aforementioned methods allow more flexibility to meet time-critical operations.

6.3 Lesson and Experience

During the implementation and deployment of the prototype, we had some unexpected findings and valuable lessons.

- We find that achieving a required time synchronization accuracy on the hardware and software platform is not as hard as on other platforms (e.g., WiFi radios [26] used on personal computers and tablets). Standard off-the-shelf WiFi radios are designed to use a big on-board buffer (as large as several megabytes) to maximize throughput. A packet timestamped at the MAC layer is queued in the on-board buffer and waits for a unpredictable time before being transmitted. The uncertainty of the queuing time significantly decreases the time synchronization accuracy. However, the CC2420 radios [65] we use have a small buffer, which can accommodate only a single packet. Moreover, the CC2420 radios have a transmission speed of 250kbps and use preamble sampling techniques to synchronize a pair of transmitter and receiver, which can tolerate relatively large time synchronization inaccuracy (e.g., up to tens of microseconds). The approach of time synchronization we use will be problematic on other radio models.

- We find that scheduling time-critical events/operations is difficult for single-thread single-stack execution model. Keeping the running time of each de-
ferred procedure as short as possible and separating large procedures into multiple small deferred procedures are effective solutions. We also believe that the problem will be even more difficult if we use multi-threading execution model.

- Setting up a testbed network for DRAMA is much more difficult than imagined due to the required coordination among interfering nodes. Though we believe that we only introduce a light structure for coordination, this structure still demands large effort for implementation and deployment. Therefore, we believe that using a lighter coordination structure may be the future direction to support real-time communications over multihop wireless networks.

6.4 Summary

We discuss the challenges to implement a prototype on the hardware and software platform we use. Meeting the timeliness requirements of time-critical operations/events is the most challenging task due to the single-task and single-stack execution model of the software platform. We augmented the execution model to be timeliness-aware and we tune our algorithms, data structures, and protocol processing flow to introduce more flexibility. As a result, the execution time of most auxiliary functionalities are afloat in certain time ranges, allowing more opportunities for time-critical operations to meet their timeliness requirements.
CHAPTER 7
EXPERIMENTS

We conducted experiments to validate the design and implementation on the prototype we have implemented on the IntelMote2 platform [49] running TinyOS [23], as described in Chapter 6. In our experiments, we vary traffic and network configurations to measure end-to-end deadline meet ratios (Section 7.2), energy consumption (Section 7.3), robustness performance (Section 7.4), bandwidth overhead (Section 7.5), storage overhead (Section 7.6), and computation overhead (Section 7.7) under link quality and traffic changes at the MAC layer. DRAMA shows its superiority in terms of real-time performance, energy consumption, and bandwidth efficiency, comparing to both contention-based and static schedule-based solutions.

7.1 Experiment Setup

We introduce the platform, the network, and the traffic configurations we use and their variations in this section. We also explain the comparisons we use, the performance measurements, and performance metrics in detail.

7.1.1 Platform Configuration

The IntelMote2 [49] is an advanced wireless sensor node platform developed at Intel Research. It is built around the low-power PXA271 XScale CPU and
integrates a commodity low-power low-rate ZigBee (IEEE 802.15.4) compliant radio (TI CC2420 [65]), which can be powered on and off [19] within several milliseconds with negligible energy consumption. The transmission rate of the radio is 250kbps and this radio does not support automatic rate control with varying signal and noise strengths. The CPU constantly runs at the speed of 104MHz although it can run at either higher or lower speeds using some dynamic voltage scaling techniques (e.g., [54]). We choose the constant running speed to filter out the energy consumption overhead brought by dynamic voltage switching.

7.1.2 Network Configuration

We tune the transmission powers and node positions to generate the test network topology, as shown in Figure 7.1. Neighboring nodes are connected by solid lines and a node’s transmission can only interfere with its neighbor’s receptions from other nodes. We set the radio frequency to an unused channel to minimize interferences from 802.11-based traffic in the area where the experiment was conducted. The observed packet loss rates of all links vary from 0% to 4%. In order to introduce further environmental interference, we use an extra node as a noise source (NS) (as shown in Figure 7.1), whose transmissions interfere the receptions of nodes 3, 4, and 5 from other nodes, as indicated by dashed lines. When the NS is active, it turns on within a slot with a probability of 1/4 and the three nodes cannot receive packets from any other nodes correctly (with high probability). We did not turn on the NS always or at most time since a broken or low-quality link might trigger a path-wide rerouting, which is not the focus of DRAMA. The NS is off by default.

In our experiments, DCS communication is bidirectional, i.e., the transmitter
will receive a hardware acknowledgement after every transmission. The time slot is 2.5 milliseconds in length, during which a DCS transmitter can send a payload of 42 bytes, which are used to record the attributes of a monitored event (e.g., time, location, and object features) and receive a hardware acknowledgement. Within a slot, an RCS broadcaster can transmit a payload of 52 bytes, plus a little margin. The radio is powered off only if at least two consecutive slots are marked as sleep. We can vary the size of either slot or frame to measure the impact of reservation periodicity on the performance and overhead of DRAMA. We choose to vary frame size from 100 to 250 slots. The default frame size is 100 slots. The size of an RCS are 2, 3, 4, and 5 in the case of frame size 100, 150, 200, 250, respectively, which are sufficient to encapsulate new and relayed reservations. The first 2, 3, 4, and 5 slots of every frame is reserved a-priori as MCS in the case of frame size 100,
150, 200, 250, respectively. The RCS rebroadcast is disabled and the reservation strides of nodes 1, 2, ⋅⋅⋅, 10 are 2, 4, 6, 14, 10, 4, 2, 12, 16, 8, respectively.

7.1.3 Traffic Configuration

We vary traffic in terms of number of streams, stream periods, stream inter-release times, loads from period to period, and deadlines. Each stream goes through a circular route 1-2-3-⋯-10-1 (Figure 7.1). We deliberately design the network topology in such a way so that we can evenly distribute the end-to-end deadline to local deadlines over the 10 hops for every stream. The default stream period, workload per period, and end-to-end deadline is 5.0, 1000, and 1000 milliseconds, respectively, which is used to model a typical surveillance or control application. We inject traffic of various levels of load into the network, in order to evaluate the performance under both admissible and overloaded workloads as well as the effectiveness of the local admission control (Section 5.3). The default traffic configuration contains 10 default streams, while the maximum admissible number of default streams is 8.

7.1.4 Default Configuration

We summarize the network, the traffic, and the platform configurations we use in default in the following experiments in Table 7.1.

7.1.5 Comparison

We compare the performance of DRAMA with CSMA/CA [27], as an example of contention-based schemes, and Static-BestSlotted, which emulates an ideal spatial slotted communication scheme which perfectly matches the default net-
### TABLE 7.1: DEFAULT EXPERIMENTAL CONFIGURATIONS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>radio transmit power</td>
<td>19.4 mA</td>
</tr>
<tr>
<td>radio sleep power</td>
<td>20 uA</td>
</tr>
<tr>
<td>radio receive power</td>
<td>12.7 mA</td>
</tr>
<tr>
<td>transmit rate</td>
<td>250 kbps</td>
</tr>
<tr>
<td>link quality PRR (NS off)</td>
<td>96%-100%</td>
</tr>
<tr>
<td>link quality PRR (NS active)</td>
<td>75% for nodes 3, 4, and 5</td>
</tr>
<tr>
<td>default NS state</td>
<td>off</td>
</tr>
<tr>
<td>max re-transmit number</td>
<td>7</td>
</tr>
<tr>
<td>$\lambda_{1,\ldots,10}$</td>
<td>2 4 6 14 10 4 2 12 16 8</td>
</tr>
<tr>
<td>frame size</td>
<td>100 slots</td>
</tr>
<tr>
<td>slot length</td>
<td>2.5 ms</td>
</tr>
<tr>
<td>stream route</td>
<td>1 $\rightarrow$ 2, 2 $\rightarrow$ 3, $\cdots$, 10 $\rightarrow$ 1</td>
</tr>
<tr>
<td>number of streams</td>
<td>10</td>
</tr>
<tr>
<td>admissible number of streams</td>
<td>8</td>
</tr>
<tr>
<td>stream period</td>
<td>1000 ms</td>
</tr>
<tr>
<td>stream end-to-end deadline</td>
<td>1000 ms</td>
</tr>
<tr>
<td>stream load per period</td>
<td>1 slot</td>
</tr>
</tbody>
</table>

work and traffic configurations. CSMA/CA uses all default parameters as in [27].

On the one hand, we compare DRAMA to Static-BestSlotted in order to measure how far DRAMA performs from an ideal scheme with the default network and traffic configurations. On the other hand, we measure how DRAMA and Static-BestSlotted perform when a little disturbance (e.g., noise) is introduced into the network and traffic has small variations (e.g., varying the number of streams, stream periods, stream inter-release times, loads from period to period, and deadlines). In Static-BestSlotted, the network is slotted with the same slot size as DRAMA, but does not have the concept of framing. A node occupies the slots whose strides modulo the maximum coloring value in the network is equal to its own coloring value [58], which ensures that each node has the same amount of bandwidth and takes turns to access the medium. When a node has the
right to access a slot, it transmits packets in the earliest-deadline first manner. When a packet’s retransmission number is greater than 7, the packet is dropped as in DRAMA. Nodes’ coloring values are determined using DRAND, which ensures approximately maximum spatial reuse in the network. Moreover, since each node undertakes the same amount of traffic and has similar neighborhood, Static-BestSlotted approximately has the maximum throughput, maximum deadline meet ratio, and minimum energy consumption with the network topology (without the NS) and the default traffic configuration in all possible slot-based schemes. Static-BestSlotted is ideal since it does not introduce any communication overhead (there is no management or control slot) and it allocates slot schedules which match perfectly with the network topology (without the NS) and the default traffic configuration.

7.1.6 Performance Measurement and Metrics

We use our experiments to measure real-time performance and energy efficiency. All the data for each network and traffic configuration are collected over a 30-minute duration after the system is stable. We measure end-to-end delays at node 1, which serves as both the origin and the destination of the real-time traffic. The end-to-end deadline meet ratio is equal to the ratio of the number of packets whose end-to-end latency is no greater than the corresponding end-to-end deadlines to the number of released packets. All dropped packets count as deadline miss. We choose to measure energy consumption at node 4 since it experiences the severest interference from the NS and thus consumes the largest amount of energy to deliver the same traffic load. We measure the idle system power consumption using an oscilloscope when both the radio is turned off and
the CPU is idle looping. The net energy consumption (involving both computation and communication) of an experiment is computed as the difference between the actual and the idle system power consumptions. We take the net energy consumption when the radio is transmitting as a denominator and normalize all net energy consumptions to it.

7.2 Real-Time Performance

We vary traffic in terms of number of streams, stream periods, stream inter-release times, loads from period to period, and deadlines to measure the ratio of deadlines meets.

7.2.1 Varying Network Workloads

Figures 7.2 shows the real-time performance when the noise source is off as the number of streams in the network changes. The deadline meet ratio of CSMA/CA is constantly below 30% due to the unpredictable medium access contentions and backoffs. Both DRAMA and Static-BestSlotted meet above 93% of deadlines when the number of streams is less than 10. However, the ratios of deadline meets in both DRAMA and Static-BestSlotted drop rapidly when the number of streams is greater than 10 and 12, respectively, where the bandwidth is saturated. Since DRAMA uses RCSs to communicate reservation information, it has approximately 80% throughput of Static-BestSlotted. Figures 7.2 also demonstrates the effectiveness of the local admission control (Section 5.3 in Chapter 5): (i) the ratio of deadline meets is high (above 95%) when the workload is admissible, and (ii) the ratio of deadline meets drops rapidly when the actual number of streams exceeds the admissible number of streams.
Figure 7.2. Real-time performance while varying traffic loads (the noise source is off).

Figure 7.3 shows the real-time performance when the interference is active as the number of streams in the network changes. DRAMA meets slightly less ratio of deadlines (<3% and <8% in unsaturated and saturated scenarios, respectively) than when the noise source is off (Figure 7.2). We observed that during the experiment the links which are interfered by the noise source reserved much more slots than the other links, to compensate for the degraded link quality. The ratio of deadline meets by Static-BestSlotted drops much fast ([5%, 60%]) and falls below that of DRAMA in unsaturated scenario, since Static-BestSlotted uses a static slot allocations targeted for static link quality.
7.2.2 Varying Stream Deadlines

Figure 7.4 and 7.5 show the deadline meet ratios when the noise source is off and active, respectively, as the end-to-end deadlines change. The changes in deadlines does not affect the behavior of CSMA/CA. In both Figure 7.4 and 7.5, the deadline meet ratio is constantly below 30% and does not increase rapidly as the deadline increases since about 70% packets are dropped in intermediate nodes in the long path. Each node in Static-BestSlotted delivers packets at a constant speed, and the deadline meet ratios rapidly decrease when the deadline becomes shorter (e.g., the deadline meet ratio is about 40% when the deadline is 500 milliseconds while the period is 1000 milliseconds). However, DRAMA still
Figure 7.4. Real-time performance while varying end-to-end deadlines (the noise source is off).

meets > 78% deadlines when the deadline is 500 milliseconds since DRAMA can accurately predict the expected arrival times and local deadlines of packets at each hop and make reservations accordingly. DRAMA also adapts its reservations according to the actual link status and has slightly lower deadline meet ratios when the noise source is active, which is true when we change other traffic and network configurations and therefore we only show the real-time performances in the following when the noise source is off.
Figure 7.5. Real-time performance while varying end-to-end deadlines (the noise source is active).

7.2.3 Varying Frame Sizes

Figures 7.6 shows deadline meet ratios of DRAMA while varying frame sizes. As the frame size increases, the frame within which a node reserves slots will be further away in time from the current frame, which reduces their responsiveness to link quality and traffic changes. However, Figure 7.6 shows that the deadline meet ratio only drops by up to 3%, mainly because the link quality changes slowly. When the external interference is highly unpredictable and changes rapidly, the deadline meet ratio may drop more.
7.2.4 Varying Stream Inter-Release Times

Figure 7.7 shows the real-time performance when the intervals between consecutive packet releases are varied. The inter-release times are randomly and uniformly distributed within the corresponding ranges, as shown in the x-axis. The deadline meet ratio of CSMA/CA decreases negligibly when the inter-release time range increases. Each node in Static-BestSlotted maintains a constant pace to delivery packets. When the traffic bursts, packets released later need to wait to access the medium. When the traffic becomes light, nevertheless, some allocated slots will be idle. Consequently, the deadline meet ratio of Static-BestSlotted decreases up to 12% when the inter-release times fluctuate within [700,1300]. How-
ever, DRAMA can predict the upcoming traffic load, arrival time, and deadlines at each intermediate node and make appropriate reservations adapting to link quality changes. Consequently, the deadline meet ratio decreases slightly (< 2%) when the inter-release times fluctuate within [700,1300] milliseconds.

7.2.5 Varying Transmission Loads From Period to Period

Figure 7.8 shows the real-time performance when the actual workload per period is varied. The workload per period is randomly and uniformly distributed within the range shown in the x-axis. As in Figure 7.7, DRAMA shows its superiority in adapting to traffic and link dynamics. The deadline meet ratio decreases
Figure 7.8. Real-time performance while varying inter-period traffic loads (the noise source is off).

slightly (< 1%) when the workload per period (1000 milliseconds) fluctuates within [2.5,12.5] milliseconds.

7.2.6 End-To-End Latency Distribution

Figure 7.9 shows the latency distribution of DRAMA with the default traffic setting (i.e., 10 default streams) when the noise source is off and active. It indicates that DRAMA can effectively regulate end-to-end latencies under various link dynamics: (i) the end-to-end latencies of more than 70% of the packets fall just before the deadlines (i.e., [900,1000] milliseconds) and (ii) the latencies of packets that missed their deadlines fall mostly within [1000, 1200] milliseconds. The short
Figure 7.9. Cumulative distribution of end-to-end latencies (the noise source is off).

and narrow trail indicates that we can force the actual latency to fall before a desired deadline with a high probability by specifying the end-to-end deadline to be slightly smaller than the desired deadline.

7.3 Energy Consumptions

Figure 7.10 shows the energy performance when the noise source is off while varying the number of streams in the network. It shows that CSMA/CA has the highest energy consumption since nodes need to constantly sense the channel, which consumes comparable energy as transmission and reception. Both DRAMA and Static-BestSlotted consume significantly less energy than CSMA/CA since
nodes in both schemes can go to sleep within the slots they have not reserved or allocated. DRAMA consumes extra energy to communicate reservation information and therefore consumes more energy than Static-BestSlotted. However, the energy consumption of DRAMA in all workloads is at most twice of that of Static-BestSlotted.

Figure 7.11 shows the energy performance when the noise source is active while varying the number of streams in the network. It shows that CSMA/CA consumes negligibly less energy than in Figure 7.10 since nodes have fewer opportunities to transmit or receive packets due to the additional interference from the noise source. Both DRAMA and Static-BestSlotted consume (4% to 8%) more energy when the
noise source is active, since more retransmissions are needed to deliver packets in time.

We show the impact of frame sizes on the energy performance of DRAMA in Figure 7.12 when the noise source is off. As the frame size increases, the number of slots that an RCS occupies will increase linearly with the frame size, yet the ratio of slots which are reserved for RCSs are approximately constant. Therefore, the energy consumption overhead on average is approximately constant (as shown in Figure 7.12) since the only energy consumption overhead comes from RCS broadcasts. The energy consumption of both Static-BestSlotted and DRAMA is closely related to the traffic load and the number of retransmissions due to the
external interference, and is less influenced by other factors (e.g., frame sizes or the length of slots). Therefore, we skip showing the energy performance in other scenarios.

7.4 Robustness and Overhead of Reservation Propagation

We measure the timeliness and overhead of reservation propagation under various link qualities and node connectivities using simulations. The main reason of using simulations instead of experiments is that link qualities usually have a very narrow transitional region between good and poor link qualities. As a result, it is super difficult to tune radios toward a specific link quality. The reservation strides
TABLE 7.2: ROBUSTNESS AND OVERHEAD OF RESERVATION PROPAGATION

<table>
<thead>
<tr>
<th>conn. degree</th>
<th>stride diffs</th>
<th>RCS reicast</th>
<th>PRR</th>
<th>timely prop.</th>
<th>Avg. Fwds</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>75%</td>
<td>93%</td>
<td></td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>90%</td>
<td>98%</td>
<td></td>
<td>4.7</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>75%</td>
<td>99%</td>
<td></td>
<td>6.2</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>90%</td>
<td>100%</td>
<td></td>
<td>5.5</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>75%</td>
<td>96%</td>
<td></td>
<td>4.6</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>90%</td>
<td>99%</td>
<td></td>
<td>4.9</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>75%</td>
<td>100%</td>
<td></td>
<td>6.8</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>90%</td>
<td>100%</td>
<td></td>
<td>5.7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>75%</td>
<td>97%</td>
<td></td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>90%</td>
<td>99%</td>
<td></td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>75%</td>
<td>100%</td>
<td></td>
<td>9.2</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>90%</td>
<td>100%</td>
<td></td>
<td>8.3</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>75%</td>
<td>100%</td>
<td></td>
<td>7.3</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>90%</td>
<td>100%</td>
<td></td>
<td>7.9</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>75%</td>
<td>100%</td>
<td></td>
<td>11.2</td>
</tr>
<tr>
<td></td>
<td>Y</td>
<td>90%</td>
<td>100%</td>
<td></td>
<td>9.9</td>
</tr>
</tbody>
</table>

of nodes 1, 2, ⋯, 10 are X * [1, 2, 3, 7, 5, 8, 4, 6, 8, 4] respectively, where X represents the reservation stride difference between two r-interfering nodes that consecutively make reservations within the same frame. We choose X to be either 2 or 3. We vary packet reception ratios (PRRs) of wireless links from 75% to 90%. Each node records the number of reservations actually received in time and the simulation program calculates the number of reservations that should be received in time. We use the average ratio among all nodes as the measure of timeliness performance. The propagation overhead is measured by the average number of relays in the network for each new reservation element.

Table 7.2 shows the effectiveness and overhead of RCS rebroadcast and reservation stride adjustment techniques under various link qualities and degrees of
connectivity: (i) either increasing the reservation strides or enabling RCS rebroadcast improves propagation timeliness while incurring more overhead, (ii) a greater degree of connectivity improves propagation timeliness at the cost of increased communication overhead, and (iii) for a given network, when the link qualities are almost perfect, all reservation elements can be propagated in a timely fashion.

7.5 Bandwidth Overhead

In DRAMA, the bandwidth overhead is introduced by RCS slots and all other slots can be reserved as DCS slots. Table 7.2 in Section 7.4 also gives a measure of bandwidth overhead in terms of the average number of relays for each new reservation element. In this section, we give a more intuitive measure of the percentage of bandwidth used by RCS broadcasts.

An RCS broadcast of a node conveys two kinds of information: (i) relayed reservations of its r-interfering nodes and (ii) its own new reservation. We assume that every available slot is either reserved as a DCS or an RCS to maximize the load of RCS broadcasts. In our implementation, an RCS only relays the overheard reservations within the frame where it reserves new reservations. In the worst-case, the RCS needs to announce or relay at most F reservations in total, assuming that all slots are reserved and consecutive slots are reserved by different nodes. Each reservation describes the reserved transmitter and receiver and takes E bytes. In our implementation, E is 1 and can support a network with up to 15 nodes. Therefore, each RCS needs to include $F \times E$ bytes in the worst-case.

We represent the bandwidth overhead as the ratio of the number of RCS slots to the total number of slots within an interference region per frame. In the worst-case, the number of nodes within an interference range is $\lambda_{\text{max}}$, where
TABLE 7.3: BANDWIDTH OVERHEAD

<table>
<thead>
<tr>
<th>number of r-interfering nodes</th>
<th>communication rate</th>
<th>slot duration</th>
<th>bandwidth overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>250kbps</td>
<td>2.5ms</td>
<td>6.8%</td>
</tr>
<tr>
<td>8</td>
<td>1Mbps</td>
<td>2.5ms</td>
<td>1.8%</td>
</tr>
<tr>
<td>8</td>
<td>10Mbps</td>
<td>1.0ms</td>
<td>0.45%</td>
</tr>
<tr>
<td>16</td>
<td>250kbps</td>
<td>2.5ms</td>
<td>13.6%</td>
</tr>
<tr>
<td>16</td>
<td>1Mbps</td>
<td>1.0ms</td>
<td>3.6%</td>
</tr>
<tr>
<td>16</td>
<td>10Mbps</td>
<td>1.0ms</td>
<td>0.90%</td>
</tr>
</tbody>
</table>

$\lambda_{\text{max}}$ is the maximum reservation stride among its r-interfering nodes. Each node reserves an RCS at each frame. A byte in an RCS consumes slightly more bandwidth than a byte in a DCS since an RCS broadcast of a node requires all of its neighbors to be interference-free while a DCS communication only requires the two partners to be interference-free. However, an RCS typically consumes less than twice the bandwidth required by a DCS and we conservatively assume so. Therefore, the RCS bytes per frame within an interference range is at most $2 \times \lambda_{\text{max}} \times F \times E$. Suppose that each slot can transmit $R$ bytes of reservation information, which is 52 bytes in our experiment. The number of slots for RCSs per frame is $\frac{2 \times \lambda_{\text{max}} \times F \times E}{R}$. Thus, the bandwidth overhead is $\frac{2 \times \lambda_{\text{max}} \times F \times E}{F \times R} = \frac{2 \times \lambda_{\text{max}} \times E}{R}$, which shows the bandwidth overhead is irrelevant to frame sizes. Table 7.3 shows the measured and estimated bandwidth overhead in various configurations. The measured bandwidth overhead matches well with the aforementioned analytical results. As the communication rate increases or the slot duration increases, the number of bytes (i.e., $R$) which can be transmitted within a slot increases and the bandwidth overhead decreases rapidly and becomes negligible.
7.6 Storage Overhead

A node’s storage overhead (excluding packet storage) mainly consists of three parts: (i) its reservation table which records its own reservations and overheard reservations from its r-interfering nodes, (ii) the interference map constructed online, and (iii) the reservation stride table of its r-interfering nodes. Parts (i) requires the most storage and we only account for it. In the worst-case, the reservation table can contain $\lambda_{\text{max}} \ast F$ slots/entries, where $\lambda_{\text{max}}$ is the maximum reservation stride among its r-interfering nodes. Each entry may associate with $\lambda_{\text{max}}$ reservation elements from its r-interfering nodes in the worst case. Each reservation element describes the reserved transmitter and receiver and takes E bytes. In our implementation, it takes 1 byte, which is sufficient for a network of up to 15 nodes. Therefore, the overheard table takes $\lambda_{\text{max}} \ast F \ast \lambda_{\text{max}} \ast E$ bytes at most. The actual storage is much less than the result in the worst case. In particular, each entry/slot in the reservation table is usually only associated with only one reservation element since usually only one node among r-interfering nodes can reserve the slot/entry without interference. Therefore, the actual size of reservation table is approximately linear to $\lambda_{\text{max}}$. Table 7.4 lists the storage overhead in total for various configurations in our experiments, where the first four rows are for nodes in our experiment and the others are estimated. Since the storage is approximately linear in both the number of r-interfering nodes and the frame size, we estimate the storage requirements linearly. The table shows that DRAMA is a viable in highly resource-constrained platforms.
### TABLE 7.4: STORAGE OVERHEAD

<table>
<thead>
<tr>
<th># of r-interfere nodes</th>
<th>F</th>
<th>storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100</td>
<td>12.5KB</td>
</tr>
<tr>
<td>8</td>
<td>150</td>
<td>15.0KB</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>15.0KB</td>
</tr>
<tr>
<td>8</td>
<td>250</td>
<td>20.0KB</td>
</tr>
<tr>
<td>16</td>
<td>100</td>
<td>24.0KB</td>
</tr>
<tr>
<td>16</td>
<td>150</td>
<td>30.0KB</td>
</tr>
<tr>
<td>16</td>
<td>200</td>
<td>35.0KB</td>
</tr>
<tr>
<td>16</td>
<td>250</td>
<td>40.0KB</td>
</tr>
</tbody>
</table>

7.7 Computation Overhead

For each slot, a node’s computation mainly occurs (see Section 4.3) when the node reserves new slots for its outgoing traffic and for its new RCS. In the worst case, the node needs to scan all entries within the frame where it attempts to reserve slots. Each entry may be associated with $\lambda_{\text{max}}$ reservation elements from its r-interfering nodes in the worst case, where $\lambda_{\text{max}}$ is the maximum reservation stride among its r-interfering nodes. Since the node may choose a slot for communication over any one of its L outgoing links, the node may need to query the interference map whether each reservation element corresponding to an entry interferes with each of its outgoing links. Therefore, the computation is $O(\lambda_{\text{max}} \times L \times F)$ for each RCS, if the query time for each pair of links is constant (e.g., using a hash table). The complexity can be further reduced with more storage consumption. Since this computation happens once in each frame, the computation energy would be negligible comparing to communication energy consumption.
7.8 Summary

We measured the real-time performance, energy consumption, robustness, storage overhead, bandwidth overhead, and computation overhead for typical real-time streams on the Intel Mote2 platform. The results show that DRAMA achieved its main goals, i.e., to meet the maximum number of end-to-end deadlines of all streams in the situation of changing traffic and network conditions. However, this benefit is achieved at the cost of large storage consumption and modest level of bandwidth and computation overhead. It shows that DRAMA is extremely effective and resource efficient to applications where constant real-time traffic exists, however, it may not be as resource efficient as in applications where traffic is sporadic and extremely light.
CHAPTER 8

CONCLUSION AND FUTURE DIRECTION

In this chapter, we conclude this thesis by (1) summarizing our contributions, (2) exposing some fundamental limitations of current solutions, and (3) proposing several directions for future work.

8.1 Contribution

Various applications, such as battlefield surveillance, industrial process monitoring and control, civil infrastructure monitoring, etc. are enabled by multihop wireless networking technology. Many wireless multihop networks carry streams of data with time-critical information (e.g., video streams in surveillance networks, sensor streams in monitoring and actuating applications, or command and control streams in factory automation applications). Such data must reach their destinations in a predictable and timely manner. Providing real-time communications in such a multihop wireless network is critical to their success. However, providing timeliness support is challenging, mainly due to (i) the inherently unreliable nature of the wireless medium and (ii) the distributed nature of multihop wireless networks, and (iii) the resource-constrained (mainly bandwidth and energy) environments. As a result, the design of an effective and efficient medium access control layer is especially important since it lays the foundation to provide actual timeliness support for all upper layers.
Over the past decades, two classes of solutions have been proposed to provide wireless medium access for various applications: scheduled-based and contention-based. While schedule-based solutions can provide the exact access opportunities which match perfectly with the requirement of given traffics, they are inflexible and susceptible to traffic and network dynamics. On the other hand, while contention-based solutions are more flexible, they can only provide unpredictable wireless medium access. Therefore, they are either over-coordinated (schedule-based solutions) or under-coordinated (contention-based solutions), failing to provide hard real-time traffic delivery in response to traffic and network dynamics. As a consequence, a new scheme is needed to strike a balance between contention-based and schedule-based approaches.

The main contribution of this dissertation is to *systematically design and evaluate an innovative medium access layer technique to support real-time communications in a multihop wireless networks*, which brings the best of both worlds of contention-based and schedule-based approaches. To achieve this goal, we use three reservation schemes (each one is built based on the proceeding one):

- **The Dynamic and Progressive Reservation Scheme.** Each node *progressively* makes slot reservations for *short periods of time*, which include slot reservations for both data traffic delivery and future reservation point. The node makes further short-term slot reservations using these future reservation point dynamically. Since each future point provides opportunities to reserve appropriate number of slots at appropriate slots, each node is able to quickly adapt to traffic and link dynamics.

- **The Orthogonal and Serialized Reservation Scheme.** While the dynamic and progressive reservation scheme ensures that a *single* node is able
to quickly adapt to changes in the network, the proposed protocol must also ensure that no conflicting reservations for the same slot can occur among multiple nodes. Toward this end, DRAMA introduces a \textit{serialized and orthogonal reservation scheme}, built upon the progressive reservation scheme, to enforce a \textit{serialized} reservation order among potentially interfering nodes and achieves \textit{fast, negotiation-free, and interference-free} reservations with \textit{high bandwidth utilization} and \textit{low bandwidth overhead}. However, potentially interfering nodes are allowed to make slot reservations simultaneously within different (i.e., \textit{orthogonal}) future time intervals, thereby preventing undue delays in the reservation scheme.

- \textbf{Real-Time Dynamic Reservation Scheme}. Finally, to satisfy the timeliness requirements of real-time traffic, nodes making slot reservations have to consider link status as well as the arrival times and deadline requirements of received and upcoming packets. Wireless links are typically unstable (e.g., due to usually uncontrolled and unpredictable environment noises) and therefore the predicted traffic and link status is not always accurate. DRAMA introduces a \textit{real-time reservation policy}, built upon the serialized and orthogonal reservation scheme, to compensates for these inaccuracies by dynamically and periodically adjusting the number of slots to reserve in a short term.

To validate the aforementioned design, we implemented a prototype and deployed it for extensive experiments. We discuss the challenges to implement a prototype on the hardware and software platform we use. Meeting the timeliness requirements of time-critical operations/events is the most challenging task due to the single-task and single-stack execution model of the software platform. We
measured the real-time performance, energy consumption, storage overhead, and bandwidth overhead for typical real-time streams on the Intel Mote2 platform. The results show that DRAMA achieved its main goals, i.e., it is energy-efficient, bandwidth efficient, and effectively supports end-to-end timeliness. We also analyze the cost and overhead for adopting our techniques.

8.2 Limitation

While in this thesis we have shown that by using the dynamic and progressive reservation techniques it is effective and efficient to support typical real-time communications in a multihop wireless network, there are several questions remains:

- Can DRAMA meet all end-to-end deadlines in a multihop wireless network? Wireless links are inherently unreliable and sometimes unpredictable. The design objective of DRAMA is to adapt to these changes rapidly and cohesively among interfering nodes. DRAMA can only increase the number of deadlines met under changing conditions and maximize it when the system is stabilized.

- Can DRAMA perform well in a mobile network? DRAMA is designed to work well in a static network with modest external and internal interferences and dynamics. When nodes are frequently moving, links will be broken and re-organized frequently. To adapt to these changes, DRAMA needs to frequently re-assign reservation strides, which greatly degrades the performance of DRAMA. The deeper reason is that DRAMA uses some structure—though flexible and light—to coordinating interfering nodes’ reservations and that maintaining such a structure costs bandwidth and time.
• What applications does DRAMA perform inefficiently? DRAMA can support best-effort traffic with high bandwidth utilization and efficiency and high energy efficiency. DRAMA is extremely effective to applications where constant real-time traffic exists, however, it may not be as effective as in applications where traffic load is light and sporadic since light communication traffic is needed to constantly maintain the structure.

• Can a node turn off if there is no data traffic going through it? Since the correct working of DRAMA relies on the relaying of reservations, the answer depends on whether the node has to relay overheard reservations. If there are no data traffic going through any of its interfering nodes, then the node can turn itself off, otherwise it must keep active.

In spite of these limitations, as we have demonstrated by analysis and experimental results, our solutions are powerful enough to approaching the performance of any wired solution in a multihop wireless network.

8.3 Future Direction

In the next sections, we identify several research directions for future work. Whenever possible, we try to emphasize the main difficulties and possible solutions to address the proposed problems.

8.3.1 Slot Clustering

The basic scheme assumes a perfect and hypothetical on-off radio, i.e., radios can switch on and off instantly on-demand without any warm-up and extra energy cost. Actual radios have warm-up interval ranging from tens of microseconds (e.g., CC2420 on MicaZ [24] or TelosB [48] platforms) to several seconds (typical WiFi
radios on laptops). Interspersed slots reserved by the same node will lead to frequent switching of the radio status and the incurred energy consumption cannot be ignored. Therefore, block slot reservation is preferred over intermittent slot reservation. However, block slot reservation tradeoffs link/network communication capacity and timeliness with energy efficiency.

Various block slot reservation schemes may be used for better energy-timeliness tradeoff, which includes: (1) each RCS of a node is immediately followed by a block of DCS where the node participates as either transmitter or receiver, i.e., each block of slots is grouped around a single node; (2) each slot block is grouped around a single node but it can be floating in frames; (3) each block of DCS is grouped around a single link, and at most one such block is allowed in each frames. To address the timeliness and energy tradeoff, we can extend our previous work \[78\] on analyzing minimal bandwidth reservation in periodic frames to satisfy real-time traffic.

### 8.3.2 Local Deadline Assignments

As a medium access control technique, each node only knows the local timeliness requirements (e.g., local deadlines), which are decomposed from global timeliness requirements (e.g., end-to-end deadlines) by the upper layers (e.g., routing or transportation layers). At the medium access control layer, DRAMA can provide flexibility to the decomposition and routing procedure of the upper layers by two means.

First, nodes’ reservation strides can be adjusted to cater for the decomposition and routing since reservation stride values are directly mapped into traffic forwarding priorities in DRAMA. As one example, heavily loaded nodes can be
assigned greater stride values than those of their neighboring nodes, to increase
the throughput and accelerate the traffic forwarding. As another example, to meet
the end-to-end deadline of an urgent stream (e.g., traversing many hops with a
short end-to-end deadline) can choose a route with nodes of greater stride values.

Second, a real-time stream can adjust the local deadline assignments according
to reservation stride values along its path. The forwarding nodes with greater
stride values are assigned with shorter local deadlines since traffic at these nodes
are forwarded with higher priorities than other nodes. If a forwarding node with a
greater stride value is assigned with a longer local deadlines than other forwarding
nodes, then we can take it as a hint to either (i) assign a smaller stride value to
the node and (ii) reroute the stream through other nodes.

8.4 Final Remarks

In this thesis, we have presented an innovative solution that supports real-
time communications in a multihop wireless network and evaluated it extensively.
While it is hard to predict the exact use of this solution, I believe that the door
has been opened to wireless real-time communications. I also believe that a good
solution will place a balance between flexibility and structuring. Although I always
believe that a little flexibility may go a long way, nevertheless only by appropriate
level of structuring the flexibility will be channeled into a successful design and
implementation.
APPENDIX A
THE DISTRIBUTED PROTOCOL FOR RESERVATION STRIDE ASSIGNMENT

We assume that the routing layer keeps track of the number of hops between r-interfering nodes (information which is typically readily available to most routing protocols). We extend the distributed node coloring scheme of DRAND [58] to assign reservation strides to r-interfering nodes in order to satisfy Theorem 1 or other reservation stride structure. In DRAND, requests for a color value assignment from a node are propagated to all of its neighbors. Grants are given to the requester node to decide a color value only none of its neighbors is requesting for a color value assignment; otherwise, these requester back off their requests randomly in a manner similar to the carrier sense multiple access with collision avoidance scheme (CSMA/CD) [26]. The duration for a node to decide its color value in DRAND will be exponential in the number of neighbors. DRAND only requires neighbors to have different color values while DRAMA requires the reservation stride values of r-interfering nodes satisfy Theorem 1. Since r-interference relationships usually involve many more nodes than there are neighbor nodes, the duration for a node to decide its reservation stride will be unacceptably long in a dynamic wireless environment if we just use the distributed node coloring scheme of DRAND.
Figure A.1. The state diagram of reservation stride allocation. The statement at the beginning of an arrow is the condition that makes the indicated state transition and that at the end of the arrow is the action taken before moving to a new state.

To accelerate the reservation stride assignment process, we parallelize the reservation stride requests and grants among r-interfering nodes. Each request in DRAMA is embedded within a reservation stride proposed by the requester, provided that Theorem 1 holds among the requester and its r-interfering nodes which have already decided their reservation stride values. Multiple requests for reservation stride assignments from r-interfering nodes can be processed and decided simultaneously, provided that Theorem 1 holds among the embedded reservation stride values of these requests. Nodes with which Theorem 1 holds do not need to change their reservation strides while other nodes are requesting their reservation strides. We briefly describe the protocol for the distributed reservation stride
assignment in the following.

In the protocol, each node can be in one of the three states: IDLE, REQUEST, and RELEASE, as shown in Figure A.1. A node stays at the IDLE state when (i) it has not decided on its reservation stride (e.g., when a node starts to join the network) or (ii) Theorem 1 does not hold among the node and some of its r-interfering nodes whose states are RELEASE. The node stays in the RELEASE state if neither (i) nor (ii) is true. A node in the RELEASE state sends out an idle message to all of its r-interfering nodes and transitions to the IDLE state if the condition (ii) is not true. Upon reception of an idle message, a node changes the recorded state of the sender of the idle message to IDLE. The node in the IDLE state uses a lottery to determine if it should initiate requests of reservation stride assignment. The winning probability of the lottery is equal to the inverse of the number of r-interfering nodes in the IDLE state. A node which won the lottery enters into the REQUEST state and sends out a request message to all of its r-interfering nodes; otherwise it stays in the IDLE state and waits for $T$ time for another lottery, where $T$ is equal to three times of the estimated largest communication latency to its r-interfering nodes. The node in the IDLE state sends out grant messages upon reception of request messages from its r-interfering nodes and stays in the IDLE state. The request message embeds the smallest reservation stride value which Theorem 1 holds among the requester (i.e., the node initiated the request) and all of its r-interfering nodes (i) which are in the RELEASE state or (ii) the requester has sent out grant messages to which yet has not received fail or release messages from which.

When a node in the RELEASE state receives a request (with a reservation stride embedded), it sends out a grant message with its reservation stride embed-
ded. When a node in the REQUEST state receives a request from another node, the node sends out a grant message in response if Theorem 1 holds among the two nodes; otherwise, it sends out a reject message. Upon reception of a reject message, a node in the REQUEST state sends out a fail message to all of its r-interfering nodes and thereby enters into the IDLE state. When a node in the REQUEST state has received grants from all of its r-interfering nodes, it sends out a release message with the reservation stride value and thereby enters into the RELEASE state.
APPENDIX B

PROOF OF THEOREM \[ \text{II} \]

By the assumption that every interference-free reservation will succeed, we assume in the following that every RCS broadcast and DCS transmission will succeed. We arbitrarily choose nodes \( A \leftrightarrow D \). We don’t consider nodes that do not r-interfere with each other since they cannot make interfering/conflicting reservations. We only need to consider the propagation from a node with greater stride to another node since nodes reserve slots within any specific frame in the decreasing order of their strides. Without loss of generality, we assume that \( \lambda_A > \lambda_D \).

Suppose Figure \[ B.1 \] shows the shortest path from node \( A \) to node \( D \) via \( n = H_{A \rightarrow D} \) hops and \( n \leq \lambda_A - \lambda_D \) by the assumption. Node \( A \) (using RCS\(_A(i, u)\)) makes reservations within \( F_{m=i+\lambda_A} \). Node \( B \) overhears RCS\(_A(i, u)\) directly. The latest time for \( B \) to relay the new reservations broadcast by RCS\(_A(i, u)\) is RCS\(_B(i+1, v)\). By induction, node \( D \) will know these reservations at RCS\(_C(i + n - 1, w)\) by directly overhearing RCS\(_C(i + n - 1, w)\) in the worst-case. By the time node \( D \) using RCS\(_D(i + n, x)\) starts to reserve slots within \( F_{i+n+\lambda_D} \), it already knows reservations made by node \( A \) within \( F_m \). Moreover, it knows all preceding frames (i.e., \( F_{m-1}, \cdots \)) since reservations within these frames are relayed before those within \( F_m \).
Figure B.1. An example of reservation propagation.

We show that $F_{i+n+\lambda_D}$ belongs to the set $\{F_{m=t+\lambda_A}, F_{m-1}, \cdots \}$. Since $n \leq \lambda_A - \lambda_D$ by the assumption, we have $t+n+\lambda_D \leq t+\lambda_A - \lambda_D + \lambda_D = t+\lambda_A$. That is, node D knows those reservations within $F_{i+n+\lambda_D}$ reserved by node A before it starts to reserve slots within $F_{i+n+\lambda_D}$. Since node A will not make any reservations that conflict with those, they are interference-free. Since all parameters are arbitrarily chosen, we have therefore proved this theorem.
APPENDIX C

PROOF OF THEOREM 3

Let $T_i = \{T_1, T_2, \ldots, T_n\}$ be the set of fixed-priority reservation periodic tasks (as in Algorithm 3) corresponding to $T_i$, where each element task interferes with $T_i$. The priority $\Pi_i$ of $T_i$ is defined as in Algorithm 3. We denote $\Pi_i < \Pi_j$ if $T_j$ has an equal or higher priority than $T_i$. From the network’s perspective, the dynamic reservation policy reserves slots for reservation tasks using the fixed-priority policy when the link qualities are perfect.

We show that the worst-case response time (the time from the arrival of a packet to the time of the slots reserved for relaying it) of $T_i$ occurs when every $T_j$ and $T_k$, where $\Pi_j > \Pi_i$ and $\Pi_k > \Pi_i$, in $T_i$ interfere with each other (denoted as case $\Pi$). We only consider reservation tasks whose priorities are higher than $T_i$’s since only they can affect the response time of $T_i$. In case $\Pi$, no slots reserved for $T_j$ and $T_k$ are overlapped. Suppose that there exist $T_j$ and $T_k$, where $\Pi_j > \Pi_i$ and $\Pi_k > \Pi_i$, in $T_i$ that do not interfere with each other (denoted as case $\Pi\Pi$). Then, $T_j$ and $T_k$ may reserve overlapping slots without conflicts, which leaves more slots available for $T_i$. Therefore, the response time of $T_i$ in case $\Pi\Pi$ may be less than that in case $\Pi$.

Thus, the worst case response time of $T_i$ in case $\Pi$, which is computed by the fixed-priority response time analysis routine, is an upper bound of the response times of $T_i$. If the worst case responses of every reservation task along the stream
path is less than the corresponding local deadline, the end-to-end deadline of every packet of the stream can be met and thus the stream is end-to-end admissible.
APPENDIX D

THE DESIGN OF DRAMA ON TINYOS

D.1 Overview

The modular architecture of the design on TinyOS is illustrated in Figure D.1. The module DRAMA is responsible for the protocol processing, corresponding to Algorithm 1 in Chapter 3. All of the other modules in Figure D.1 assist the DRAMA module.

The SlottedTime module converts physical time, logic time, and frame time from one another. The physical time of a node is the time since the node’s bootstrap. Different nodes have different physical times since they are bootstrapped at different times. The logic time is a network time and synchronized among nodes. The frame time is a network time like the logic time and is used by the DRAMA module for protocol processing. The DRAMA module deals only with frame times. A frame time value is uniquely denoted by (frame index, slot index). We discuss how DRAMA synchronizes the logic times and provides frame time to the DRAMA module for protocol processing in detail in Section D.2.

The SlottedComm module provides primitive slot-based packet transmission and reception, which relies on the underlying device-dependent CC2420TDMA module for actual communication. Unlike the existing CSMA module on TinyOS, the CC2420TDMA module turns off both the transmission backoff and clear chan-
Figure D.1. The software architecture of TinyOS implementation of DRMAM. The line arrows indicate command calls or event callbacks.

...nel assessment (CCA) features. On the one hand, the SlottedComm accepts communication requests using frame times from the DRAMA module, and directs the underlying CC2420TDMA module for actual communication at specific physical times. The SlottedComm module invokes the SlottedTime module to execute the conversion between the frame times and the physical times. On the other hand, the SlottedComm module is called by the CC2420TDMA module once a packet is received and the module converts the physical time upon reception to the corresponding frame time. The received packet is classified and dispatched to the DRAMA module for protocol processing.

The Reserve module book-keeps all reserved reservations, either overheard from its neighbors or reserved by itself. Based on these reserved reservations, the node derive a table of newly reserved reservations since its last RCS broadcast. These newly reserved reservations will be relayed to their respective interference ranges in the next RCS broadcast. The Reserve module also maintains a reservation status table for each outgoing links, which marks whether a slot is reservable over the link, or has been reserved by itself or by the other end of the link. The
Reserve module also provides reservation statistics (e.g., the number of slots over a given link by a certain time) to the DRAMA module, to facilitate its reservation decision-making. The details of the Reserve module is discussed in detail in Section D.4.

The PacketQueue module stores received packets and locally generated packets (if the node is a source for a stream). Each outgoing links has a queue and packets in each queue are stored in the earliest-deadline-first manner. This module also provides packet statistics (e.g., how many packets are queued for each stream over each outgoing link) to the DRAMA module, which uses these statistics as an indicator of traffic demand for reservation decision-making. This module is discussed in detailed in Section D.3.

The Stream module keeps track of the traffic model parameters (e.g., local deadline, next hop, periods, load per periods, and latest packet release time of each stream) and predicts future traffic load for each stream.

The Interference module keeps track of an interference map. It provides the DRAMA module interference primitives (e.g., if transmissions over two links are interfering). The DRAMA uses these primitives to make non-interfering reservations and relay appropriate reservation elements. The LinkQuality module keeps track of the packet reception ratio (PPR) of each outgoing link, base on acknowledgments piggybacked by RCSs. The DRAMA module uses the PPRs to determine the appropriate number of slots to reserve. The Stride module keeps track of the reservation strides of all r-interfering nodes and is executing the stride assignment protocol as discussed in Section B.

In the following sections, we will discuss the implementation of the Slotted-Time, the PacketQueue, and the Reserve modules in details since they are at the
core of the DRAMA implementation.

D.2 The SlottedTime Module

RCSs are augmented with four fields that facilitate time synchronization among nodes and provide frame times to the DRAMA module. The four fields include (1) the logic time of the packet transmission request, (2) the delay from the moment that the logic time field is written to the moment that the packet is being transmitted, (3) the frame index of the RCS packet, and (4) the slot index of the RCS packet.

The logic time field $l_0$ is written after the packet has been assembled and right before the packet is delivered to the CC2420TDMA module for actual transmission. When the logic time is written, a corresponding physical time $p_0$ is recorded as well. When a packet is starting to transmitting, a SFD (start frame delimit) interrupt at both the transmitter and the receiver takes place simultaneously (if the negligible short-distance signal propagation latency is ignored). Upon the SFD interrupt, the transmitter and the receiver record their respective physical times as $t_0$ and $r_0$. The delay value $d_0$ is equal to $t_0 - p_0$ and will be written in the delay filed of the next RCS. Upon the reception of the RCS packet, all of the four fields are extracted by the receiver. The receiver calculates its physical time $p'_0$ corresponding to the logic time $l_0$ as $r_0 - d_0$. Then, the pair $(l_0, p'_0, f_0, s_0)$ is recorded as a synchronization point, where $f_0$ and $s_0$ are the frame index and the slot index of the RCS packet.

For each synchronization point, $l_0 - p'_0$ is taken as the offset of the physic time to the logic time at the moment. The average offset over the last ten synchronization points is used for synchronization purpose. The clock skew is ignored
since the time durations between synchronization points are short. Based on the
synchronization points, the number of slots per frame and the time duration of
each slot can be calculated as in Section 6.2.1 in Chapter 6.

D.3 The PacketQueue Module

The PacketQueue module stores packets that received from upstream or re-
leased locally using the MsgQueue as shown in Figure D.2. The MsgQueue is an
array of packets in the form of (Next, Deadline, MessageBuf), where the Next
field points to another packet, the MessageBuf field and the Deadline field are
the packet data buffer and deadline, respectively. All packets that share the same
next hop are linked into a list in the earliest-deadline-first (EDF) order. All free
packets are linked into another list which is headed by FreeHead.

When the system is initialized, the PacketQueue module provides the Slot-
tedComm module with a free packet, which is used for the reception of the next
incoming packet. When a packet is received by the SlottedComm module, the
packet is queued into the corresponding packet list in the EDF order (depending
on the next hop of the packet) and a free packet is returned to the SlottedComm
module for the reception of the next packet.

When the current slot is reserved by the node as a DCS to a neighbor, the node
picks the first packets (i.e., the one with the earliest deadline) in the packet list
corresponding to the neighbor and requests the SlottedComm module to transmit
it. When the current slot is reserved by the node as an RCS, the PacketQueue
module provides the statistics of queued packets over each outgoing links to the
DRAMA module, which determines an appropriate number of slots to reserve.
D.4 The Reserve Module

The Reserve module of each node book-keeps the reservation information that either is overheard or reserved by itself. Like the PacketQueue module, a single array (ResvElemPool) of reservation elements in the form of (EntryNext, NewlyNext, ResvElem) are used for storing overheard, newly reserved, and newly overheard (since last RCS) reservations. The EntryNext and NewlyNext fields point to the next reservation element corresponding to the same slot and the next newly overheard reservation element, respectively. The ResvElem field stores the actual reservation element in the form of (transmitter ID, receiver ID, frame index, slot index) as in Section 4.1 in Chapter 4.

The reservation elements are organized into three singly-linked lists: all known (either overheard from its neighbors or reserved by itself) reservations (linked from ResvTable), the newly reserved list (headed by NewlyResv), and the free list (headed by FreeElems). The ResvTable is organized as a cycle array of frames.
and each frame consists of entries corresponding to each slot in the frame. Each entry points to a list of known reservations (linked by the EntryNext field) in the ResvElemPool. The newly reserved list is linked by the NewlyNext field. The free list is linked by either the NewlyNext and EntryNext field.

When an RCS is overheard or the node makes new reservations, free reservation elements are extracted from the free list and their embedded fields are filled out accordingly. If the reservation element is new (i.e., it has not been linked into the corresponding slot entry in ResvTable), then it is linked into both the corresponding slot entry in ResvTable and the newly reserved list (NewlyResv). If a slot entry in ResvTable expired, the list of reservation elements corresponding to the slot is linked to the free list.

To facilitate the DRAMA module to determine which slot to reserve, the Reserve module provides a reservation status table (LinkResvX) for each outgoing link. Each such table is organized like the ResvTable and each slot entry of the
table is marked with a reservation status (e.g., reservable, non-reservable, reserved-as-transmitter, reserved-as-receiver). The default status is reservable, but when a reservation element is linked into ResvTable, the status of the corresponding slot entry in each LinkResv table is updated accordingly by the DRAMA module (with the consultant from the Interference module). A node can only reserve the reservable slots over the corresponding link. Moreover, a node must wake up to transmit or receive packets when the current slot is reserved-as-transmitter or reserved-as-receiver, respectively. When the DRAMA module starts to assemble an RCS packet, it just embeds the reservation elements in NewlyResv into the RCS packet since all overheard and self-reserved reservation elements are included in NewlyResv. When the RCS broadcast completes, the embedded reservation elements can be linked into the free list (FreeResv) (linked by the NewlyNext field).
BIBLIOGRAPHY


15. D. Ferrari and D. Verma. A scheme for real-time channel establishment in


28. IEEE 802.3 WG. IEEE standard part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications. 2008.

29. IEEE Std. 802.16-2004. IEEE standard for local and metropolitan area net-
works: Part 16: Air interface for fixed broadband wireless access systems. 


136

66. TinyOS Community. TEP 133: packet level time synchronization.


