REAL-TIME SCHEDULING ALGORITHMS IN WIRELESS SENSOR NETWORKS

YI HUI CHEN

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

AUGUST 2010

Your file Votre référence
ISBN: 978-0-494-71028-9
Our file Notre référence
ISBN: 978-0-494-71028-9

Abstract

Real-Time Scheduling Algorithms in Wireless Sensor Networks

Yi Hui Chen

Wireless sensor networks have recently attracted research and industry interest. Wireless sensor networks have great potential to be used in many applications because of their unique characteristics. In many applications, such as military battlefield surveillance and medical health care, the data packets need to be delivered to their destinations with real-time constraints. Traditional real-time algorithms cannot be directly used in wireless sensor networks. Therefore, a new challenge in wireless sensor networks arises.

In this thesis, we proposed several new real-time scheduling algorithms for wireless sensor networks. Unlike other existing real-time scheduling algorithms, we design real-time scheduling algorithms that not only consider the current situations of packets but also take the travelling history of packets into consideration. We evaluated these algorithms both in terms of the packet delivery rate and fairness to different flows. Finally we extended *IEEE* 802.11 to be able to prioritize the packets. We implemented these algorithms in NS-2 and extensively evaluated the experimental results. The results demonstrate that our new algorithms efficiently increase the system real-time performance and fairness.

Acknowledgments

I would like to express my sincere thanks to all those who supported and helped me during my studies and during the critical stages of completing this thesis. It would not have been possible for me to be here without their help.

First of all, I express my deep gratitude and respect to my supervisor Dr. L. Narayanan, whose wisdom, knowledge and experiences shaped numerous insightful conversations, without which many ideas in this research would not have been well developed. Her support, guidance and patience helped me to make this thesis possible.

I owe my sincere gratitude to my colleagues Kun Xu, Hossein Kassaei, and Mona Mehrandish for providing a stimulating and fun working environment.

Finally, I would like to give my special thanks to my beautiful wife Ning Ru whose persistent love enabled me to complete this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensor networks have recently been studied as a premier research topic. Such networks consist of spatially distributed autonomous sensors communicating with each other via wireless media channels [2]. These sensors cooperatively work together to accomplish some tasks, such as remote environmental monitoring and target tracking. With microelectronic technology improvements, sensor nodes have become smaller and smaller, at the same time as becoming efficient in terms of power consumption [55]. As the cost of sensor nodes continues to fall, wireless sensor networks will be more and more widely used in many applications such as military, intelligent agriculture and environment monitoring. Many researchers are currently engaged in wireless sensor networks from a variety of perspectives, such as hardware design, deployment mobility, coverage and connectivity, lifetime and QoS [23]. Guaranteeing Real-time QoS is an important research area in wireless sensor networks.

In this chapter, we will discuss some wireless sensor network applications. Then we

will introduce some research challenges in wireless sensor networks. Finally, we will describe the motivation and contribution of this thesis.

1.1 Applications in Wireless Sensor Networks

Wireless sensor networks can be used in various applications such as intelligent agriculture, medical health care, environment monitoring and protection, and military battlefield surveillance [49]. In addition, wireless sensor networks can be divided into two types, homogeneous and heterogeneous. In homogeneous networks, each node has an identical function and performs the same task. However, there are different types of sensors in heterogeneous networks. Therefore, this kind of network can perform multiple operations simultaneously [45].

Intelligent agriculture is one of the important wireless sensor network applications. Sensors can detect the temperature, soil moisture and humidity. In addition, wireless sensor networks can also monitor agricultural pests. The project to create regional weather and on-farm sensor networks in Washington State is one typical example of this kind of application. This wireless sensor network upgraded the local public agricultural weather system. In addition, this application provided an affordable real-time mobile system for local fruit producers to protect their fruit from frost [43].

Medical health care is another application area. In [17], researchers established a reliable medical body sensor network in a hospital. In this project, wireless sensors are used to monitor patients and transmit medical data to observation units via a wireless sensor

network. In this application many kinds of sensors are applied to do different operations.

Environment monitoring and protection is one of the primary applications of wireless sensor networks. Sensor networks can be used to monitor the weather, wild animals and environments such as ocean water, active volcanoes and glaciers. The project of bird observation on Great Duck Island is one typical example. In this project, a wireless sensor network was used to observe the behavior of a bird called Leach's Storm Petrel on Great Duck Island, Maine, the United States. This kind of bird is easily disturbed, so a wireless sensor network became an appropriate way to observe them. Within one year, over one million readings had been logged from 32 sensors [35]. Volcano monitoring is also an example in this area [33]. In 2005, researchers deployed sensors equipped with microphones and seismometers to collect data of volcano activity. There were nearly 107M bytes data collected, and 230 eruptions were successfully detected during a 19-day deployment period. In addition, glacier monitoring is another example in this field [37]. In August 2003, researchers deployed sensors in drill holes at different depths in the glacier ice to monitor a sub-glacier environment at Briksdalsbreen, Norway. This wireless sensor network collected lots of data that can enhance our understanding of the earth's climate. These examples show how important wireless sensor networks are in environment monitoring and protection. Wireless sensors can be deployed in harsh environments that are not easily accessible to human beings.

Military applications are another important application area for wireless sensor networks. In the 1960s, the U.S. army deployed thousands of sensors in Vietnam war to detect enemy troop movement [6]. The project "tracking vehicle with a UAV-delivered sensor

network" detected and tracked vehicles passing through the network [25]. The sensor is unnoticeable and reliable. All information transferred in this network has time constraints. In this project, sensors were deployed from an unmanned aerial vehicle (UAV). Each sensor node could configure itself into a multi-hop wireless network and synchronized internal clock. A magnetometer sensor was used to detect deviations in the magnetic field caused by metal contained in the vehicles. Tracking results are transmitted to the UAV. In addition, wireless sensor networks were used in the design of anti-tank landmines [39]. These landmines could communicate with each other to ensure all areas are covered.

1.2 Challenges in Wireless Sensor Networks

Unlike wired infrastructure networks such as the Internet, wireless sensor networks have some unique features. Therefore, the solutions developed for wired network cannot be simply applied to wireless sensor networks. In this section, we will present some challenges for the design and implementation of wireless sensor networks.

1. Resource Constraints: A smart sensor has multiple functions such as sensing, computing and communication. In addition, there are challenges such as how to consume extremely low power; how to efficiently use limited transmission media and how to keep the size of sensors smaller. Even through hardware technology has dramatically improved recently, these constraints are still tight [14].

2. Environment : Many sensors operate in inaccessible locations such as hostile areas, volcanoes, and oceans [14].

3. Scability and High Density: Sensors may be deployed from a moving platform. As a result, the density of the sensors could vary from hundreds to thousands in a certain area. Protocols should be able to adapt to this situation [14].

4. Sensor Network Topology: The network topology for wireless sensor networks can be frequently changed due to sensors' mobilities. Therefore, how to self-organize and collaborate to provide an acceptable performance has to be considered [48].

5. Real Time Constraints: Time constraints are very important in wireless sensor network design because many applications have implicit time requirements. However, wireless sensor networks have many nondeterministic characteristics such as sensor failure, noise and dynamic network topology. In addition, the majority of real-time solutions cannot be directly applied in distributed systems. Therefore, how to guarantee real-time requirements becomes a huge challenge in wireless sensor networks [48].

1.3 Motivation and Contribution

As we discussed, there are many challenges in wireless sensor network research. How to efficiently deliver packets to their destinations within their deadlines using scarce resources is a primary challenge in wireless sensor network applications.

The delivery of a packet before its deadline can be affected by many network protocols, for example the routing protocol, the scheduling or queue management protocol, and the

media access protocol. In this thesis, we focus primarily on scheduling or queue management: at each intermediate node, when the MAC layer is ready to send a packet, which packet out of the queue should be forwarded first?

The problem we attempt to solve in this thesis can therefore be stated as follows: Given a wireless sensor network with n nodes, and a set of k flows (source-destination pairs), such that each packet in every flow has an associated deadline, find a scheduling algorithm to be used at intermediate nodes that maximizes the total number of packets that reach their destinations before their specified deadlines. The algorithm should also try to maximize some measure of fairness between flows.

To this end, we propose several new scheduling algorithms in this thesis. Our algorithms are based on the following four main criteria to choose the next packet to schedule: the deadline of a packet, the ratio of remaining distance to remaining time, the punctuality of the packet, and the packet drop ratio of the flow. Of these, to our knowledge, using the punctuality of the packet as a criterion is a completely novel idea. We explore several combinations and variations of these criteria to obtain the best possible performance. We consider both the percentage of packets that reach their destinations before their deadlines, and the fairness between flows to evaluate performance.

Extensive simulations show that our algorithms obtain improved performance over the previously known algorithms in terms of both packet drop ratio and fairness. Finally, we demonstrate that adding a prioritizing extension to the IEEE 802.11 MAC protocol can further enhance the performance of our algorithms.

1.4 Outline of the Thesis

In Chapter 2, we provide some background of wireless sensor network scheduling algorithms and routing protocols. First we present a popular hardware platform motes and its software platform TinyOS. Then, we describe several real-time scheduling algorithms. In addition, we briefly describe *IEEE* 802.11 DCF protocol. Finally, we introduce the routing protocol AODV. In Chapter 3, we propose four baseline algorithms and six new real-time scheduling algorithms. We describe the details of each proposed real-time algorithm. In Chapter 4, we briefly describe the network simulator NS-2 that we used in this thesis. Then we present how to implement our proposed algorithms in NS-2. In addition, we describe the details of MAC-layer Prioritization Design and Implementation. In Chapter 5, we present the experimental results and evaluation. Finally, in Chapter 6, the conclusions of this thesis and future work are described.

Chapter 2

Background and Related Work

Many routing and scheduling algorithms have been developed for wireless sensor networks. Some of them are adapted from early algorithms for the Internet and traditional ad hoc networks. In this chapter, we survey the important scheduling algorithms that lead to our proposed algorithms.

In this chapter, firstly, we introduce some background on wireless sensor network platforms including hardware, software and development software. The purpose of this introduction is to provide a concept of a state-of-the-art sensor network. Secondly, we describe the traditional real-time scheduling, real-time scheduling in wired networks and in wireless networks, and present some specific real-time scheduling algorithms in each field. Finally, we describe two typical MAC layer and routing layer protocols used in wireless sensor networks.

2.1 Wireless Sensor Network Hardware

Unlike other wireless devices such as laptops and PDAs, wireless sensors have many limitations. As stated in chapter 1, size and cost are two major concerns for wireless sensors. Fortunately, over the past decade, many types of wireless sensor nodes have been designed by the University of California at Berkeley. These nodes are widely used in many wireless sensor network applications. A sensor node consists of several components including a transducer, micro processor, Flash ROM, RAM and EEPROM, a radio, an antenna and a programming interface. This sensor node is equipped with a battery, and it consumes very low power. Table 1 lists the types of sensor node available when this thesis was started:

Mote	Dot	Mica	Telos	Tmote Sky
Year	2000	2001	2004	2009
CPU	ATmega163	ATmega103	MSP430	MSP430
ROM(KB)	16	128	48	48
RAM(KB)	1	4	10	10
Nonvolatile Storage(KB)	32	512	1024	1024
Radio	TR1000	CC1000	CC2420	CC2420

Table 1: Berkeley sensor nodes [5]

Based on Table 1, the newest model of Berkeley sensor node is called Tmote Sky. According to [3], Tmote Sky is the next-generation sensor node consuming extremely low power. It also has the feature of high data-rate, fault tolerance and development ease. It provides various integrated peripherals such as, I2C, SPI and UART bus protocols. In addition, Tmote Sky offers a large external flash and a USB interface for programming, debugging and data collection. Figure 1 shows the front and back of the Tmote Sky mote.

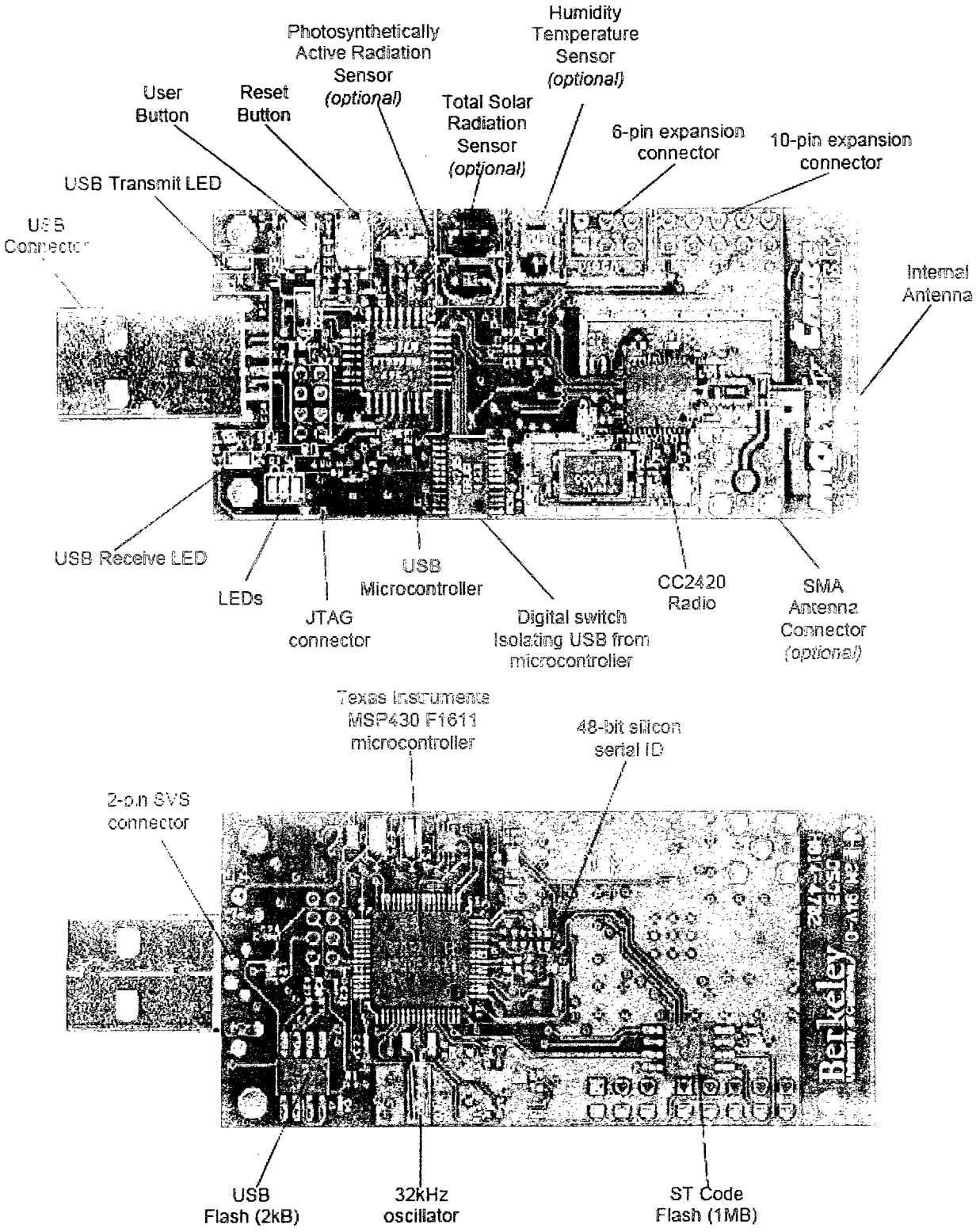

Figure 1: The front and back of Tmote Sky module [4]

Operating systems provide a famework to manage hardware and to provide programming interfaces. Traditional embedded real-time operating systems such as VxWorks, UC/OS and pSOS, require more memory than sensor nodes can provide. Therefore, sensor nodes need particular embedded real-time operating systems. Currently, there are some operating systems available for sensor nodes such as TinyOS and Contiki. TinyOS and Contiki are open source software, and they are specifically designed for running applications in the small size memory of sensor nodes. The development language for TinyOS is Networked Embedded Systems C (NesC) [1]. The development language for Contiki is the C language [9].

2.2 Scheduling Algorithms

The main purpose of a scheduling algorithm is to dispatch a task or a data packet to system resources such as processor time and communications bandwidth. In this section, we discuss traditional real-time scheduling algorithms and real-time scheduling algorithms on both wired networks and wireless networks.

2.2.1 Traditional Real-time Scheduling Algorithms

Traditional real-time scheduling algorithms are designed for centralized systems. They are used to assign a processor's time to multiple tasks to ensure that each task meets its deadline. It can either statically or dynamically prioritize tasks. Rate monotonic scheduling and deadline monotonic scheduling are typical static algorithms [27]. Both of these

algorithms statically set the priority for each task. Therefore, they require complete knowledge of the task set and its constraints in advance. In order to improve system flexibility, some schedulers were proposed that can dynamically schedule tasks such as the earliest deadline first algorithm [32], spring scheduling algorithm [44] and robust earliest deadline scheduling [19]. Next, we present rate monotonic scheduling and the earliest deadline first scheduling.

Rate Monotonic Scheduling

In many real-time control applications, periodic tasks represent the major computation demand such as control loops, system monitoring and sensory data acquisition [21].

As a classical periodic real-time scheduling, Rate Monotonic Scheduling (RMS) [32] enjoys widespread use in CPU scheduling. RMS is a periodic fixed-priority scheduler ,which assigns the priority based on the period of each task. The shortest period task will get the highest priority.

A major limitation of periodic fixed-priority scheduling is that it does not always fully utilize the CPU. Therefore, we introduce a processor utilization factor to measure the CPU workload. The definition of processor utilization factor U is the fraction of processor time spent in the execution of the task set [21]. We denote T_i as the period of task i, and C_i as the worst case execution time of task i. The expression for U is given below:

$$U = \sum_{i=0}^{n} \frac{C_i}{T_i} \tag{1}$$

According to [21], the utilization U in RMS is bounded by $n \times (2^{\frac{1}{n}} - 1)$, where n is the number of tasks in the system. From the above expression, we find that the value of U decreases as the number of tasks increases. The worst case upper bound finally converges to $ln\ 2$ ($\approx$ 0.69). Therefore, RMS guarantees that an arbitrary set of periodic tasks is schedulable if the total processor utilization does not exceed a value of 0.69.

Earliest Deadline First

Earliest Deadline First (EDF) is a classic dynamic scheduling algorithm used in real-time operating systems. It selects tasks based on their deadlines. At any given time, the task with the earliest deadline will be assigned the highest priority.

We denote p_k^i as the k_{th} packet in the traffic flow Q_i, $G(p_k^i)$ as the generating time of packet p_k^i and $D(p_k^i)$ as the deadline of packet p_k^i. At time T, a packet p_k^i is scheduled if $G(p_k^i) \leq T \leq D(p_k^i)$ and $D(p_k^i)$ is the minimum among all the packets.

Earliest Deadline First guarantees that an arbitrary set of periodic tasks is schedulable when the utilization is not more than 1, in contrast to Rate-Monotonic Scheduling which can only provide the guarantee for a utilization of 0.69.

2.2.2 Real-time Scheduling Algorithm for Wired Networks

Compared with traditional real-time scheduling algorithms, real-time scheduling algorithms in wired networks focus on communication scheduling. The two main reasons the packet could be dropped are overflow and missed deadline. Real-time scheduling algorithms in

wired networks have been well-studied owing to the rise in popularity of multimedia applications. Many of these algorithms use individual packet deadlines to schedule packet transmissions over the outgoing link in order to minimize the number of packets that miss their deadlines.

High-quality audio can tolerate the loss of 5 to 10 percent [40]. The tolerable losses for video are relatively low, up to one percent [46]. On the Internet, different kinds of traffic flows pass through the same routers. Some of them are real-time data flows, which are sensitive to delay. Others are non-real-time data flows. which are sensitive to loss. The simplest priority scheduling is a static priority scheduling [16]. In this scheme, priority is always given to the real-time traffic. As a result, high loss rates are experienced by the non-real-time data flows. We describe two dynamic scheduling methods proposed for real-time and non-real-time flows: Minimum Laxity Threshold (MLT) and Queue Length Threshold (QLT) [20].

MLT gives sending priority to real-time traffic when the minimum *laxity* among the queued real-time packets is less than or equal to a threshold L. The definition of *laxity* is the remaining time to the deadline. Otherwise, priority is given to the non-real-time data flow. The value of L is appropriately chosen to achieve a tradeoff between these two types of traffic. Packets in the same type of traffic are served in an FCFS manner.

QLT gives sending priority to non-real-time traffic when the length of the queue for non-real-time traffic exceeds threshold value T. Otherwise, priority is given to real-time traffic. The value of T is appropriately chosen to achieve a tradeoff between these two types of traffic. Packets in the same type of traffic are served in an FCFS manner.

In [15], a scheduling algorithm called LDOLL was provided. In this algorithm, the oldest packet is replaced when the buffer fills. Real-time traffic is given priority when the number of non-real-time traffic packets is below a given threshold. In [29], a scheduling algorithm was proposed in ATM networks. It discards audio traffic cell first, then non-real-time data and finally video traffic.

2.2.3 Real-time Scheduling Algorithm for Wireless Sensor Networks

Recall that wireless networks have many challenges that wired networks do not have to suffer. For example, wireless sensor networks' channel may have bursty error. In addition, the network topology of wireless sensor networks can change frequently causing the data transmission path to be changed frequently as well.

Real-time scheduling for wireless sensor networks can be classified as location-based algorithms and location-free algorithms. Location-based algorithms require the location of node, which can be obtained from a Global Positioning System (GPS). For example, in paper [13], the scheduling algorithm requires the location of the gateway nodes. In addition, the Probing Environment and collaborating Adaptive Sleeping algorithm requires location information to probe the neighborhood [24].

Location-free algorithms do not need location information. Recall that Earliest Deadline First guarantees an arbitrary set of periodic tasks is schedulable when the utilization is not more than 1. In [12], Earliest Deadline First was adapted for real-time wireless sensor networks to provide the maximum overall system throughput with an assumption that there is not too much channel error. This algorithm is a location-free algorithm. In addition,

there are other location-free real-time scheduling algorithms proposed in [53] and [50]. We present one location-free algorithm and one location-based algorithm as follows.

Lagging Flow First Algorithm

Lagging Flow First(LFF) is a location-free algorithm proposed in [12]. In this paper, the authors considered not only how to deliver the packets to arrive at their destinations within their deadlines but also how to minimize the maximum degradation. The degradation value is defined as ϵ

$$\epsilon = 1 - \frac{M_i^a}{M_i} - e_i \tag{2}$$

M_i^a is the number of packets that flow i actually successfully delivered; M_i is the number of packets that flow i was supposed to deliver; e_i is acceptable packet loss rate [12].

This algorithm maintains two queues. One is a reservation list to hold the packets that will be delivered. The other is a buffer to hold packets that do not have spots in the reservation list. This algorithm is composed of two phases. To begin with, whenever a packet is available to be scheduled, a time slot could be reserved based on its deadline. Then this packet could be inserted into a reservation list if it is possible. Otherwise, this packet will be inserted into the second queue. In the second phase, this algorithm schedules a packet in the reservation list. Therefore, we can observe that the packets in the reservation list have higher priorities.

In the reservation list, a time slot denoted as TS is reserved for a new packet P_j^i such that no time slot between TS and the deadline of P_j^i is reserved for another packet p_n^m such that the traffic flow Q_m has a lower degradation than the traffic flow Q_i.

In order to maintain this reservation list. this algorithm first performs a search process from the deadline of packet p_j^i to the current time to look for an empty time slot or a time slot which is occupied by a packet p_n^m. If there is an empty time slot available, it will be reserved for packet P_j^i. If there is a time slot associated with a low degradation traffic flow Q_m, this time slot will be reserved for new packet p_j^i. If no time slot is available for packet p_j^i, packet p_j^i will not be in the reservation list. If any packet p_n^m yield its time slot to other packets, a new searching process will be performed for this packet p_n^m.

We present an example to illustrate the difference in scheduling performed by EDF and LFF algorithms. Assume there are six packets. Their deadlines and the degradation values are listed in Table 2. The schedule generated by the algorithms are shown in Table 3.

Packet	Deadline	Degradation Value
p_1^1	t+2	0.02
p_1^2	t+2	0.06
p_1^3	t+2	0.04
p_1^4	t+3	0.02
p_1^5	t+3	0.04
p_1^6	t+3	0.06

Table 2: Packet information

Algorithm	t+1	t+2	t+3
LFF	p_1^3	p_1^2	p_1^6
EDF	p_1^1 *or* p_1^2 *or* p_1^3	p_1^1 *or* p_1^2 *qr* p_1^3	p_1^4 *or* p_1^5 *or* p_1^6

Table 3: Packet schedules

According to EDF, at time slot $t+1$, p_1^1, p_1^2 and p_1^3 have the earliest deadline $t+2$. Therefore, any of them will be scheduled at time slot $t+1$. Similarly, at time slot $t+2$ one of remaining two packets will be scheduled. Finally, at time slot $t+3$, p_1^1, p_1^2 and p_1^3

are already past their deadlines and therefore one of p_1^4, p_1^5 and p_1^6 will be scheduled. However, if using LFF, at time slot $t+1$, p_1^1, p_1^2 and p_1^3 have the earliest deadline and there are two time slots available. Therefore, $t+1$ is reserved in the reservation list for p_1^1 and $t+2$ is reserved in the reservation list for p_1^2. p_1^3 will compare its degradation value with the packets that occupy the time slot $t+1$ and $t+2$. Then p_1^1 has to yield its time slot to p_1^3. Next, we look at the packets with the next earliest deadline. They have only one slot in the reservation list available to be scheduled, and p_1^6 has the highest degradation value, so it is scheduled.

From the above discussion, we may find that this algorithm takes into consideration both the deadline and the degradation of the traffic flow.

Velocity Monotonic Scheduling Algorithm

Velocity Monotonic Scheduling (VMS) is a location-based algorithm proposed by Chenyang Lu [34]. The basic idea of this algorithm is to prioritize packets based on their *velocities*. The definition of packet's velocity is the ratio of the distance that it needs to travel to the time before its deadline. There are two versions of velocity monotonic scheduling: static VMS (SVM) and dynamic VMS (DVM).

Static VMS fixes the priority of packet when the packet is generated. The definition of velocity is given below:

$$Velocity(p_j^i) = \frac{Distance\ between\ source\ and\ destination(p_j^i)}{Available\ Time(p_j^i)} \tag{3}$$

The distance measure used in this algorithm is Euclidean distance. The routing algorithm used is GPSR, which uses geographic forwarding to the node progressively closer to the destination [28]. Packets' priorities are determined based on the velocities of these packets. A higher velocity packet will get a higher priority, and it will be placed in the appropriate queue. All intermediate nodes have a separate queue for each priority level and packets of higher priority are always forwarded before packets of lower priority. Since the priority is fixed at the beginning, an intermediate node cannot change it even if any unpredictable delay occurs.

On the other hand, dynamic VMS recalculates the velocity and resulting priority at each intermediate node. The definition of velocity used in dynamic VMS is:

$$Velocity(p_j^i) = \frac{Distance\ between\ current\ node\ and\ destination(p_j^i)}{Remaining\ time(p_j^i)} \tag{4}$$

The advantage of dynamic VMS is that an intermediate node has an ability to adjust the priority based on the specific situation.

Detailed information on the assignment of priority levels based on velocities for both static and dynamic VMS is shown in Table 4.

Priority	*SVM*	*DVM*
1	(10, ∞)	(40, ∞)
2	(5, 10]	(10, 40]
3	(0, 5]	(0, 10]

Table 4: Priority ranges (m/s) of SVM and DVM [34]

2.3 Media Access Control Protocol in Wireless Sensor Networks

Media Access Control protocols have been extensively studied in traditional wireless networks. Time division multiple access (TDMA), frequency division multiple access (FDMA) and code division multiple access (CDMA) are widely used in the wireless cellular communication systems. The commonality in these protocols is to avoid collision by assigning wireless nodes onto different sub channels, which are divided by time, frequency or orthogonal codes [54].

IEEE 802.11 is often used in wireless sensor networks. It is known that *IEEE* 802.11 supports both infrastructure networks and ad hoc networks [41]. Therefore, *IEEE* 802.11 MAC layer specifies two kinds of access methodologies as follows.

1. Distributed Coordination Function(DCF): This is a contention-based protocol, which is used in ad hoc networks.

2. Point Coordination Function(PCF): This is a contention-free protocol, which is used in infrastructure networks.

In this thesis, we used the *IEEE* 802.11 DCF in our implementation. Therefore, we describe this function in detail below.

Prior to sending a data packet, the sender will send a small control packet called Request to Send (RTS). The function of RTS is to inform the neighbors of the sender of a coming transmission. Before sending RTS, the node will sense the channel for a time interval called

DCF Inter Frame Space (DIFS). Upon receiving RTS, the receiver will send a Clear to Send (CTS) packet back to the sender if the channel is idle. The purpose of CTS is to prevent the neighbor around the destination from transmitting at the same time. After the sender receives CTS, it will sense the channel a time interval called Short Inter Frame Space (SIFS) before sending DATA packets. Whenever the receiver gets a DATA packet from the sender, it will respond with an Acknowledgment (ACK) packet indicating a correct reception of the DATA packet.

In addition, each wireless node maintains a table called Network Allocation Vector (NAV). RTS and CTS contain a duration field in their frame headers, which indicates the duration of the RTS-CTS-DATA-ACK procedure. Nodes that hear either RTS or CTS will update their NAV tables, and they will be silent during this period.

When the node is about to send packets and the channel is occupied, it will keep silent for a random number of time slots chosen between 0 to $ContentionWindow(CW)$. This period of time is called *backofftime*. If the node hears any transmission within this *backofftime*, this timer will be frozen until the channel becomes idle again. If any collision happens, *backofftime* will be doubled until it reaches the maximum value (cw_max). When a node fails to get an acknowledgement, it will retransmit this packet unless a maximum number of retries has been reached.

2.4 Routing Algorithm for Wireless Sensor Networks

Dynamic routing algorithms are prevalent in wired networks. Distance Vector Routing and Link State Routing are two of the most popular dynamic routing algorithms [31]. However, these routing algorithms cannot be directly adapted to ad hoc or sensor networks because of the frequently changing network topology in ad hoc networks. Fortunately, many routing algorithms have been designed for ad hoc networks. They can be divided into proactive routing, reactive routing and hybrid routing based on the manner in which routing information is acquired.

A proactive routing algorithm builds routing paths before any packet is about to be sent, and it periodically exchanges topology information to maintain them. Destination-Sequenced Distance-Vector (DSDV) is a typical example of a proactive routing algorithm. In contrast, a reactive routing algorithm finds a routing path only if it is necessary. Therefore, it will perform a process to find a routing path to the destination just before sending packets. Ad Hoc On-Demand Distance Vector (AODV) and Dynamic Source Routing (DSR) are typical examples of reactive routing algorithms. Based on these algorithms' features, we can conclude that a proactive routing algorithm has a low packet delivery latency compared with a reactive routing algorithm. However, routing path maintenance overhead in proactive routing algorithms will be much higher than the overhead in reactive routing algorithms [38]. Therefore, reactive routing protocols are more suitable for the networks in which network topology changes frequently, or where resources are scarce.

Being a general purpose algorithm, AODV can be used in all ad Hoc networks. According to paper [51], AODV has the best performance among DSDV, DSR and AODV. It can handle mobility and scalability pretty well with an acceptable latency. Therefore, in this thesis, we decided to use AODV as the routing protocol.

2.4.1 AODV Protocol

AODV was proposed by C. Perkins [42]. Being a reactive routing algorithm, AODV only maintains the routing paths that are used in packet transit. Compared with DSR, AODV maintains the routing table in each node rather than in each data packet header. There are three kinds of control packets used in AODV.

1. Route Request (RREQ): This packet is used in a route discovery operation when the destination is not available. It includes the addresses of the source and the destination. In addition, it has request ID, hop count and the sequence number of the source and the destination. Figure 2 shows the fields contained in a RREQ packet.

source address	request ID	source sequence No.	destination address	destination sequence No.	hop count

Figure 2: The RREQ packet [47]

2. Route Replies (RREP): If the intermediate node knows the route to the destination or the destination receives a RREQ packets, they will send a RREP packet back to the source. Figure 3 shows the fields contained in a RREP packet.

source address	destination address	destination sequence No.	hop count	life-time

Figure 3: The RREP packet [47]

3. Route Error (RERR): This packet is to report a broken routing path.

One difference between DSR and AODV is that each node in the AODV algorithm has a routing table. Each routing entry in the routing table contains the following information: destination, next hop, number of hops, sequence number of the destination, active neighbours for this route, and expiration time for this route entry.

When the source node initiates a route discovery process to the destination, a RREQ packet will be broadcast to the network. When its neighbours receive this packet, they will check the source address and request id, which can uniquely identify this packet. If it is the first time they receive this packet and they do not have the routing information to the destination, they will rebroadcast it and the hop count of RREQ will be increased by one. Otherwise, they will discard it. In addition, they will set up a backward entry that points to the source in their routing table. Eventually, the destination or any intermediate node that has routing information to the destination receives this RREQs packet. Then a RREP packet will be sent back to the source. Figure 4 illustrates the process of route discovery.

The RREP packet will travel back to the source in the reverse direction. In addition, all intermediate nodes will set up a forward entry in their routing table to the destination. When the RREPs packet arrives at the source, the route discovery process is completed.

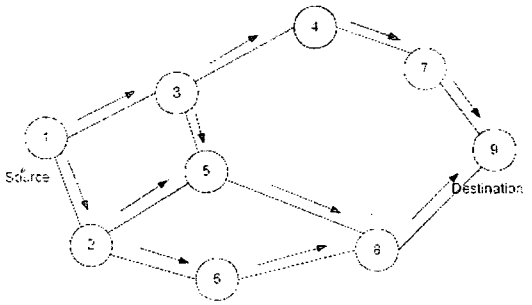

Figure 4: The RREQ broadcast in AODV [31]

Figure 5 illustrates the process of RREPs forwarding.

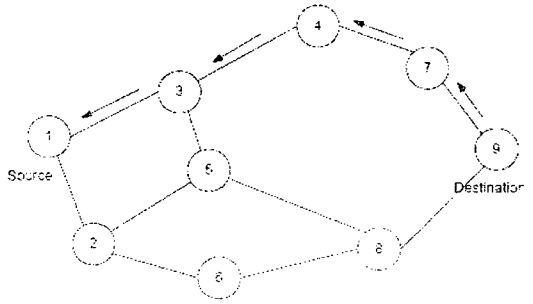

Figure 5: The RREP forward in AODV [31]

When the route is set up successfully, the source can start sending data packets to the destination. Each routing entry has an expiration period. If this routing entry is idle for a while, this route will be considered broken and a RERR packet will be generated and sent back to the source indicating that the destination is unreachable.

Each node can get to know its one-hop neighborhood by using HELLO packets. The purpose of HELLO packets is to inform its neighborhood that is is still alive. Hello packets

will not be forwarded. When a node receives a HELLO packet, it updates the corresponding lifetime of the neighbor information in its routing table. This local connectivity management should be distinguished from general topology management to optimize response time to local changes in the network [47].

In addition, AODV uses sequence numbers to solve the loop problem. Each node maintain its own sequence number. When it sends a RREQ, its sequence number will be incremented. In addition, when it sends a RREP, its own sequence number will be the maximum of the current sequence number and the sequence number in the RREQ.

In summary, AODV is quite efficient as an ad hoc routing algorithm. It has the minimum control traffic overhead regarding the cost of increased latency to find new routes. It can also react rapidly to network topology changes.

2.5 Summary

In this chapter, first we provide the state-of-the-art background of wireless sensor network hardware. Then, we surveyed several algorithms that are related to our research in particular scheduling algorithms, including algorithms specifically designed for wireless sensor networks, MAC-layer contention resolution algorithms and routing algorithms. In each case, we gave a detailed description of the algorithms either used in this thesis or closely related to the algorithms proposed in this thesis.

Chapter 3

Proposed Real-Time Scheduling Algorithms

In this chapter, we define the multiple-hop real-time scheduling problem in wireless networks and propose various solutions to address this problem. The purpose of all real-time scheduling algorithms is to fulfill timeliness constraints. Traditional real-time scheduling algorithms assign processors' time to tasks in a centralized manner. However, the scheduling algorithm in a wireless sensor network has to work in a distributed manner. The scheduling algorithm in each node must make scheduling decisions without any global knowledge. In addition, the problem of how to independently schedule packets at each node to achieve the minimum overall packet missing ratio is proved as an NP-hard problem [30]. Therefore, heuristic algorithms are needed.

We assume that all nodes can work as a wireless data source, a router or a destination. All the traffic flows are assumed to be unidirectional. In addition, there are no mobile

nodes in the wireless sensor network. The clocks on all nodes are synchronized. In the first section we give some baseline algorithms. In Section 3.2 we propose the Most Punctual First algorithm. In Section 3.3 two variation of earliest deadline first algorithms. In the last section we propose Highest Velocity First and two variation algorithms.

3.1 Baseline Algorithms

The main purpose of baseline algorithms is to introduce some basic concepts and define some basic standards to compare with the subsequently proposed algorithms. The following lists four candidates.

3.1.1 Single-Queue Algorithms

In this section, we will introduce two single-queue algorithms: the Single-Queue Deadline-Unaware algorithm and Single-Queue Deadline-Aware algorithm. Figure 6 shows the scheduling queue architecture of a node where F1, F2 and F3 represent the different traffic flows that share a common queue in a node. A packet in the scheduling queue is scheduled based on a First-In-First-Out (FIFO) policy.

The Single-Queue Deadline-Unaware algorithm does not check the deadlines of packets until packets arrive at their destinations. This algorithm is described in Algorithm 1.

The main purpose of the Single-Queue Deadline-Unaware algorithm is to measure the difference between a real-time system and a non real-time system. The major concern of a

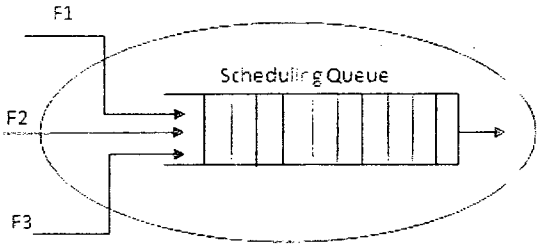

Figure 6: The single-queue scheduling model

Algorithm 1 The Single-Queue Deadline-Unaware Algorithm

1: *Upon receiving a packet* P_j
2: **if** *not full*(*scheduling queue*) **then**
3: *enqueue*(P_j, *scheduling queue*)
4: **else**
5: *drop*(P_j)
6: **end if**
7: *Upon receiving a call back from MAC layer*
8: **if** *not empty*(*scheduling queue*) **then**
9: *pkt* $\leftarrow$ *dequeue*(*scheduling queue*)
10: *Send pkt to MAC layer*
11: **end if**

non real-time system is to successfully deliver a packet to its destination. However, a real-time algorithm must consider both the delivery rate and the time constraint of each packet. Therefore, we propose the second baseline algorithm: Single-queue with Deadline-aware algorithm.

The main purpose of the Single-Queue Deadline-Aware algorithm is to take into consideration the time constraint of packets in a real-time system. The Single-Queue Deadline-Aware algorithm at a node checks the deadline of packets when it receives or forwards packets to the MAC layer and drops them if the deadline has been missed. The pseudocode of this algorithm is listed in Algorithm 2.

In the previous algorithm, packets whose deadlines have been missed in transit are still forwarded along to the destination and are only dropped at the destination. Since the scheduling algorithm is FIFO, these packets would potentially hold up other packets that have not yet missed their deadlines.

3.1.2 Multiple-Queue Algorithms

The single queue algorithms in the previous section place packets belonging to different flows in the same queue. In order to assure fairness between the different flows, we introduce two algorithms that maintain a different queue for each flow. The first algorithm is the Multiple-Queue Deadline-Unaware algorithm, and the second is the Multiple-Queue Deadline-Aware algorithm. The scheduling model of these two algorithms in each node is shown in Figure 7. The solution is to use different queues for each flow and to serve

Algorithm 2 The Single-Queue Deadline-Aware Algorithm

```
1: Upon receiving a packet P_j
2: if not full(scheduling queue) then
3:    if deadline(P_j) > Current Time then
4:       enqueue(P_j, scheduling queue)
5:    else
6:       drop(P_j)
7:    end if
8: else
9:    drop(P_j)
10: end if
11: Upon receiving a call back from MAC layer
12: found ← false
13: while not empty(scheduling queue) and not(found) do
14:    pkt ← dequeue(scheduling queue)
15:    if deadline(pkt) > Current Time then
16:       found ← true
17:    else
18:       drop(pkt)
19:    end if
20: end while
21: if found is true then
22:    Send pkt to MAC layer
23: end if
```

these flows in a Round-robin fashion. Therefore, the packets will be selected from different queues and be sent to MAC layer. The packet in each scheduling queue is scheduled based on a First in First out (FIFO) policy.

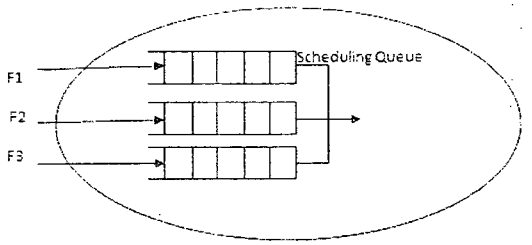

Figure 7: The multiple-queue scheduling model

In the Multiple-Queue Deadline-Unaware algorithm, intermediate nodes do not check deadlines of packets. The pseudocode of this algorithm is given in Algorithm 3.

Algorithm 3 Multiple-Queue Deadline-Unaware Algorithm

```
1: Assume that there is a queue corresponding to every flow
2: Upon receiving a packet P_j with flow id i
3: if not full(Q_i) then
4:    enqueue(P_j, queue(i))
5:    ▷ insert P_j into a queue with flow id i
6: else
7:    drop(P_j)
8: end if
9: Upon receiving a call back from MAC layer
10: while not found do
11:    Q_i ← next queue using Round-robin policy
12:    while not empty(Q_i) and not(found) do
13:       pkt ← dequeue(Q_i)
14:       Send pkt to MAC layer
15:       found ← true
16:    end while
17: end while
```

In the Multiple-Queue Deadline-Aware algorithm, intermediate nodes check the deadlines of packets when they receive and send packets. Compared with the previous one, this algorithm promptly removes a packet whose deadline has already been missed to prevent the packets that have not missed their deadlines from being held up. The pseudocode of this algorithm is listed in Algorithm 4.

Algorithm 4 The Multiple-Queue Deadline-Aware Algorithm

```
1: Assume that there is a queue corresponding to every flow
2: Upon receiving a packet p_j with flow id i
3: if not full(Q_i) then
4:    if deadline(p_j) > Current Time then
5:       enqueue(p_j, queue(i))
6:       ▷ insert p_j into a queue with flow id i
7:    else
8:       drop(p_j)
9:    end if
10: else
11:    drop(p_j)
12: end if
13: Upon receiving a call back from MAC layer
14: found ← false
15: while not found do
16:    Q_i ← next queue using Round-robin policy
17:    while not empty(Q_i) and not(found) do
18:       pkt ← dequeue(Q_i)
19:       if deadline(pkt) > Current Time then
20:          found ← true
21:       else
22:          drop(pkt)
23:       end if
24:    end while
25:    Send pkt to MAC layer
26: end while
```

3.2 Most Punctual First

In this section, we introduce a new criterion to schedule packets, which takes both the spent time and the covered distance of the packet into consideration. First, we will introduce a new parameter called the *Measure of Punctuality* (MP).

The *Measure of Punctuality* of a packet is defined as the difference between the ratio of the time already spent in transit to the total available time for transit and the ratio of the number of hops the packet has already traveled to the number of hops from the source to the destination. We denote p_k^i as the k_{th} packet in the traffic flow Q_i, and available time for transit is denoted by Δ. For packet p_k^i, the ratio of the time already spent in transit to the total available time for transit is denoted by T_k^i, and the ratio of the number of hops the packet has already traveled to the number of hops between the source and the destination as H_k^i. The expressions of T_k^i and H_k^i are denoted as follows:

$$T_k^i = \frac{Time\ Spent(p_k^i)}{\Delta} \tag{5}$$

$$H_k^i = \frac{Number\ of\ hops\ covered(p_k^i)}{Total\ number\ of\ hops(p_k^i)} \tag{6}$$

Then the *Measure of Punctuality* of packet p_k^i is given by

$$MP(p_k^i) = T_k^i - H_k^i \tag{7}$$

It is not hard to observe that $T_k^i \in [0, 1]$ and $H_k^i \in [0, 1]$. Referring to the above equation,

we can easily find that $MP(p_k^i) \in [-1, 1]$. When $-1 \leq MP(p_k^i) < 0$, the packet p_k^i can be considered ahead of schedule. When $MP(p_k^i) = 0$, packet p_k^i can be considered on time. Otherwise, $MP(p_k^i)$ indicates that packet p_k^i is behind schedule.

We denote $SP(p_k^i)$ as the packet sending priority, which is defined as

$$SP(p_k^i) = 1 - \mid MP(p_k^i) \mid \tag{8}$$

The relationship between $SP(p_k^i)$ and $MP(p_k^i)$ is shown in Figure 8.

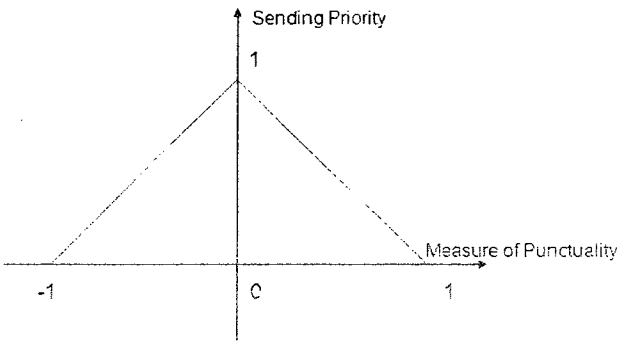

Figure 8: The relationship between packet sending priority and measure of punctuality

$SP(p_k^i)$ increases in direct proportion to $MP(p_k^i)$ when $-1 \leq MP(p_k^i) < 0$. In contrast, the $SP(p_k^i)$ decreases in inverse proportion to $MP(p_k^i)$ when $0 < MP(p_k^i) \leq 1$.

At the decision time T, the head packet within each queue is iteratively inspected. If a packet p_k^i has already missed its deadline, it is dropped and $MP(p_k^i)$ will not be calculated either. The next packet is checked until a packet that has not missed its deadline is found. After all queues are inspected, the packet with the maximum sending priority among all queues will be dequeued and sent to the MAC layer. The intuition behind the heuristic is

as follows, If a packet is very much ahead of schedule, it can afford to wait and need not be prioritized. On the other hand, if a packet is very much behind schedule, it will likely be dropped later, and therefore, should not be prioritized now.

We present a scheduling example of two algorithms, Earliest Deadline First and Most Punctual First, to compare them. Table 5 illustrates the deadline, hops number, MP, and

Packet	G_j^i	D_j^i	$Hops$	T_j^i	H_j^i	$MP(p_j^i)$	$SP(p_1^1)$
p_1^1	0	3	2	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{5}{6}$
p_1^2	0.25	3.25	4	$\frac{1}{4}$	$\frac{1}{4}$	0	1

Table 5: Two packets information

SP of two packets p_1^1 and p_1^2 from two separate traffic flows Q_1 and Q_2. We assume at time 1 second, both of them have finished one hop and arrived at the same node. The deadline of p_1^1 is at 3 second, and the deadline of p_1^2 is at 3.25 second. Obviously based on the Earliest Deadline First algorithm, p_1^1 has a higher priority. However, according to the Most Punctual First algorithm, $SP(p_1^1)$ is $\frac{5}{6}$ and $SP(p_1^2)$ is 1. Therefore, p_1^2 has a higher priority. The pseudocode of Most Punctual First Algorithm is listed in Algorithm 5.

3.3 Earliest Deadline First and Variations

In this section, first we briefly introduce Earliest Deadline First. The scheduling model of Earliest Deadline First in each node is the same as the Multiple-Queue Deadline-Aware Algorithm shown in Figure 7. Each traffic flow has a separate queue. We assume each packet has exactly the same time Δ available for transit. That is, for every packet p_k^i, its deadline $D(p_k^i) = G(p_k^i) + \Delta$. Therefore, a packet generated earlier also has an earlier

Algorithm 5 The Most Punctual First Algorithm

Assume that there are k flows , and there is a queue corresponding to every flow
Upon receiving a packet p_j with flow id i
if *not full*(Q_i) **then**
  if *deadline*(p_j^i) > Current Time **then**
    enqueue$(p_j^i, queue(i))$
    ▷ insert p_j^i into a queue with the same flow id
  else
    drop(p_j^i)
  end if
else
  drop(p_j^i)
end if
Upon receiving a call back from MAC layer
max ← 0
for $i = Q_1$ to Q_k **do**
  found ← *false*
  while *not empty*(Q_i) and *not*(*found*) **do**
    look up the front packet pkt in Q_i
    if *deadline*(*pkt*) < Current Time **then**
      drop(*pkt*)
    else
      if max < *SP*(*pkt*) **then**
        max ← *SP*(*pkt*)
        ▷ SP(pkt) is the sending priority of pkt
        flowId ← *i*
      end if
      found ← *true*
    end if
  end while
end for
if *found* **then**
  pkt ← *dequeue*(*flowid*)
  Send pkt to MAC layer
end if

deadline. The Earliest Deadline First policy can be implemented by using a FIFO queue.

At the decision time T, the head packet within each queue is iteratively inspected. If the head packet has missed its deadline, it is dropped. The next packet is inspected until a packet that has not missed its deadline is found. After all queues are inspected, the packet with the earliest deadline among all queues will be dequeued and sent to the MAC layer. The pseudocode for Earliest Deadline First is listed in Algorithm 6.

In addition, we can replace multiple FIFO queues with a single priority queue that is operated based on the deadlines of packets. Table 6 shows the running time performance of these two data structures. We can observe that it may be more efficient by using multiple FIFO queues structure.

Data Structure	*enqueue*	*dequeue*
Single Priority Queue	$O(\log n)$	$O(\log n)$
multiple FIFO Queues	$O(1)$	$O(k)$

Table 6: Performance of two data structures, n is the number of packets in the queue, k is the number of queues

In [12], Earliest Deadline First was used in a one-hop wireless network. In this thesis, we apply Earliest Deadline First for a multiple hop wireless sensor network. In the Earliest Deadline First algorithm, a packet could be dropped or not depending on its deadline. However, in a multiple-hop network traveling, a packet could be very close to its deadline, but still has a long journey to go. Therefore, the overall system could waste its valuable resources to transit such kind of packets that would be dropped in future. We propose an algorithm, Earliest Deadline among Punctual Packets, to try to improve real-time performance.

Algorithm 6 The Earliest Deadline First Algorithm

```
1: Assume that there are k flows , and there is a queue corresponding to every flow
2: Upon receiving a packet p_j with flow id i
3: if not full(Q_i) then
4:    if deadline(p_j^i) > Current Time then
5:       enqueue(p_j^i, queue(i))
6:    else
7:       drop(p_j^i)
8:    end if
9: else
10:   drop(p_j^i)
11: end if
12: Upon receiving a call back from MAC layer
13: min ← ∞
14: for i = Q_1 to Q_k do
15:    found ← false
16:    while not empty(Q_i) and not(found) do
17:       look up the front packet pkt in Q_i
18:       if deadline(pkt) < Current Time then
19:          drop(pkt)
20:       else
21:          if deadline(pkt) <min then
22:             min ← deadline(pkt)
23:             flowId ← i
24:          end if
25:          found ← true
26:       end if
27:    end while
28: end for
29: if found then
30:    pkt ← dequeue(flowid)
31:    Send pkt to MAC layer
32: end if
```

3.3.1 Earliest Deadline among Punctual Packets

The main function of the scheduling algorithm is to forward incoming packets. Earliest Deadline among Punctual Packets is based on the ideas of Earliest Deadline First and Most Punctual First. In order to avoid wasting limited resources to send a packet that would be dropped later, Earliest Deadline among Punctual Packets will study the packets' traveling history by calculating *Measure of Punctuality* of packet. We define a packet p_j^i to be punctual if $\mid MP(p_j^i) \mid < \alpha$ where α represents a punctuality threshold and will be determined later. Therefore, this algorithm considers both the packets' deadlines and packets' traveling history.

The punctuality threshold α may be either a fixed value between 0 and 1 or can be a random value between 0 and 1. It remains to decide what to do with the packets not deemed punctual. Observe that packets with $\alpha < MP < 1$ are behind schedule and are perhaps unlikely to make it to the destination on time. We simply drop such packets. On the other hand, packets with $-1 < MP < -\alpha$ are ahead of schedule. This is why we did not choose them to be forwarded in the current step, but there is no need to drop them. Such packets simply remain in the queue.

From the definition of MP, we can find that the head packet in each traffic queue has the highest value of MP in its queue. The reason is the value of T_j^i of head packet p_j^i is maximum, and the values of H_j^i are the same as the subsequent packets in the queue. As a result, if $MP(p_j^i) > Threshold$, packet p_j^i will be dropped until a packet with $MP(p_j^i) < Threshold$ is found. The pseudocode of Earliest Deadline among Punctual Packets is listed in Algorithm 7.

Algorithm 7 The Earliest Deadline among Punctual Packets Algorithm

```
1: Assume that there are k flows , and there is a queue corresponding to every flow
2: Upon receiving a packet p_j with flow id i
3: if notfull(Q_i) then
4:   if deadline(p_j^i) > Current Time then
5:     enqueue(p_j^i, queue(i))
6:   else
7:     drop(p_j^i)
8:   end if
9: else
10:  drop(p_j^i)
11: end if
12: Upon receiving a call back from MAC layer
13: min ← ∞
14: for i = Q_1 to Q_k do
15:   found ← false
16:   while not empty(Q_i) and not(found) do
17:     look up the front packet pkt in Q_i
18:     if deadline(pkt) < Current Time then
19:       drop(pkt)
20:     else
21:       calculate MP(pkt)
22:       if | MP(pkt) |> α) then
23:         if MP(pkt) > 0 then
24:           drop(pkt)
25:         else
26:           hold(pkt)
27:         end if
28:       else
29:         if deadline(pkt) <min then
30:           min ← deadline(pkt)
31:           flowId ← i
32:         end if
33:         found ← true
34:       end if
35:     end if
36:   end while
37: end for
38: if found then
39:   pkt ← dequeue(flowid)
40:   Send pkt to MAC layer
41: end if
```

3.3.2 Earliest Deadline among Degraded Flows

From the viewpoint of the system, it would be desirable to successfully transmit as many packets as possible over all traffic flows. In Earliest Deadline First, the traffic flow with the most urgent deadline will obtain the highest priority. As a result, some traffic flows will have high delivery rate. On the other hand, some traffic flows with lower priorities could have a large proportion of packets that miss their deadlines or even experience starvation. This phenomenon will impact on the quality of service of the whole wireless sensor network. Therefore, it is very important to improve the fairness of the whole wireless sensor network.

For example, in Figure 9 there are three flows going through the node A. We assume traffic flow F_1 has the shortest deadline among these three flows; flow F_2 has the intermediate deadline, and flow F_3 has the longest deadline. Based on the Earliest Deadline First criteria, flow F_1 will obtain the highest priority to be forwarded. The burst of flow F_1 will cause flow F_1 to significantly occupy the communication channel so that the packets in flow F_2 and flow F_3 have less chance to be transferred.

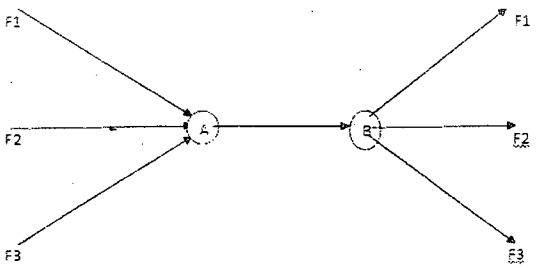

Figure 9: The fairness in multiple-flow wireless sensor network

First of all, we introduce a parameter called *drop ratio* to measure the fairness of the whole network. We denote the *drop ratio* of traffic flow Q_i as $MR(Q_i)$. The expression of $MR(Q_i)$ is defined below:

$$DR(Q_i) = \frac{Number\ of\ dropped\ packets(Q_i)}{Total\ number\ of\ packets(Q_i)} \tag{9}$$

Then, we introduce a threshold value called *DropRatioThreshold* whose purpose is to adjust priorities among traffic flows. That is, when the drop ratio of Q_i exceeds the *dropRatioThreshold*, the node will prioritize traffic flows Q_i based on the drop ratio of Q_i. Under this circumstance, some traffic flows could temporarily obtain higher priorities even though they are not urgent. In this algorithm, we create two queues to classify the drop ratio of different queues. If the $MR(Q_i)$ is more than *DropRatioThreshold*, we will put its earliest deadline and its flow id into a queue called *OverThresholdQueue* which contains all queues whose drop ratios are over threshold. On the other hand, its earliest deadline and its flow id will be put into a queue called *NormalQueue*, which contains all queues whose drop ratios are less than threshold. The queue in *OverThresholdQueue* has a higher priority than the queue in *NormalQueue*. The pseudocode of Earliest Deadline among Degraded Flows is listed in Algorithm 8.

3.4 Highest Velocity First and Variations

In this section, we discuss our three proposed algorithms: Highest Velocity First, Highest Velocity among Punctual Packets and Highest Velocity among Degraded Flows.

Algorithm 8 Earliest Deadline among Degraded Flows

Assume that there are k flows , and there is a queue corresponding to every flow
Upon receiving a packet p_j with flow id i
if *not full*(Q_i) **then**
  if *deadline*$(p_j^i) >$ Current Time **then**
    enqueue$(p_j^i, queue(i))$, *numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
  else
    drop(p_j^i), *numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
    numOfDropped$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
  end if
else
  drop(p_j^i), *numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
  numOfDropped$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
end if
Upon receiving a call back from MAC layer
$min1 \leftarrow \infty, min2 \leftarrow \infty$
for $i = Q_1$ to Q_k **do**
  found $\leftarrow$ *false*
  while *not empty*(Q_i) and *not*(*found*) **do**
    look up the front packet pkt in Q_i
    if *deadline*$(pkt) <$ Current Time **then**
      drop(pkt), *numOfDropped*$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
    else
      calculate $MR(Q_i)$
      if $MR(Q_i) > missingRatioThreshold$ **then**
        if $min1 >$ *deadline*(pkt) **then**
          $min1 \leftarrow$ *deadline*(pkt), *flowId*$1 \leftarrow i$
        end if
      else
        if $min2 >$ *deadline*(pkt) **then**
          $min2 \leftarrow$ *deadline*(pkt), *flowId*$2 \leftarrow i$
        end if
      end if
      found $\leftarrow$ *true*
    end if
  end while
end for
if *found* **then**
  if $min1! = \infty$ **then**
    pkt $\leftarrow$ *dequeue*(*flowId*1), *Send pkt to MAC layer*
  else
    pkt $\leftarrow$ *dequeue*(*flowId*2), *Send pkt to MAC layer*
  end if
end if

3.4.1 Highest Velocity First

Recall that Velocity Monotonic Scheduling prioritizes these packets into three levels [34]. Each level has a specific velocity range. We observe that since a queue for a given priority level is FIFO, packets with a higher velocity in the queue may get stuck behind packets with lower velocities in the same level. Secondly, the definition of velocity is based on distance, which is not an accurate measure even if geographic routing is used. Finally, the definition of velocity range for a priority assumes a certain network area and it is therefore not flexible.

Based on the above discussion, we propose our new algorithm called Highest Velocity First. In our algorithm, each traffic flow has a FIFO queue at each intermediate node. Similar to dynamic VMS, we define the velocity of packet to be the number of hops remaining over the remaining time. It is easy to see that the head packet within each queue has the largest value of velocity. The algorithm iteratively inspects the head packet for each queue. If the head packet in the queue has missed its deadline, it will be dropped, and the next one is checked until a packet whose deadline has not been missed is found. Finally, the packet with the highest velocity among all the flows will be sent to the MAC layer. The pseudocode of Highest Velocity First is listed in Algorithm 9.

3.4.2 Highest Velocity among Punctual Packets

This algorithm combines the idea of Highest Velocity First and Most Punctual First. It is known that Highest Velocity First considers both the remaining time of the packet and the remaining distance to the destination. Most Punctual First traces back the packet traveling

Algorithm 9 Highest Velocity First

Assume that there are k flows , and there is a queue corresponding to every flow
Upon receiving a packet p_j with flow id i
if *not full*(Q_i) **then**
  if *deadline*(p_j^i) > Current Time **then**
    enqueue$(p_j^i, queue(i))$
    ▷ insert p_j^i into Q_i
  else
    drop(p_j^i)
  end if
else
  drop(p_j^i)
end if
Upon receiving a call back from MAC layer
max ← 0
for $i = Q_1$ to Q_k **do**
  found ← *false*
  while *not empty*(Q_i) and *not*(*found*) **do**
    look up the front packet pkt in Q_i
    if *deadline*(*pkt*) < Current Time **then**
      drop(*pkt*)
    else
      if *velocity*(*pkt*) > max **then**
        max ← *velocity*(*pkt*)
        flowId ← *i*
      end if
      found ← *true*
    end if
  end while
end for
if *found* **then**
  pkt ← *dequeue*(*flowid*)
  Send pkt to MAC layer
end if

history. Therefore, Highest Velocity among Punctual Packets considers both the past and the future of packet when it is scheduling packets.

This algorithm is similar with Earliest Deadline among Punctual Packets. To begin with, the algorithm will iteratively inspect each queue. If the packet has missed its deadline, it will be dropped and the next packet will be checked until a packet whose deadline has not been missed is found. Then, the *MP* value of the packet will be calculated. If $\mid MP(p_j^i) \mid <$ *punctuality threshold*, packet p_j^i will be scheduled later. The pseudocode of Highest Velocity among Punctual Packets is listed in Algorithm 10.

3.4.3 Highest Velocity among Degraded Flows

As with Earliest Deadline First, Highest Velocity First could have a fairness problem as well. The problem is that some traffic flows with high priorities have higher successful delivery rate compared with other traffic flows with low priorities. Therefore, we propose Highest Velocity among Degraded Flows to improve the fairness. The pseudocode of Highest Velocity among Degraded Flows is listed in Algorithm 11.

3.5 Summary

In this chapter, we proposed several algorithms for scheduling in multiple-hop wireless networks. First, we proposed two baseline algorithms, two of which do not take deadlines of packets into consideration at all, and the other two simply drop packets that have already missed their deadlines. We introduced the notion of punctuality of a packet and proposed

Algorithm 10 Highest Velocity among Punctual Packets

Assume that there are k flows . and there is a queue corresponding to every flow
Upon receiving a packet p_j with flow id i
if *not full*(Q_i) **then**
  if *deadline*(p_j^i) > Current Time **then**
    enqueue$(p_j^i, queue(i))$
    ▷ insert p_j^i into a queue with the same flow id
  else
    drop(p_j^i)
  end if
else
  drop(p_j^i)
end if
Upon receiving a call back from MAC layer
max ← 0
for $i = Q_1$ to Q_k **do**
  found ← *false*
  while *not empty*(Q_i) and *not*(*found*) **do**
    look up the front packet pkt in Q_i
    if *deadline*(*pkt*) < Current Time **then**
      drop(*pkt*)
    else
      calculate MP(*pkt*)
      if $\mid MP(pkt) \mid > \alpha$ **then**
        if $MP(pkt) > 0$ **then**
          drop(*pkt*)
        else
          hold(*pkt*)
        end if
      else
        if *velocity*(*pkt*) >max **then**
          max ← *velocity*(*pkt*)
          flowId ← *i*
        end if
        found ← *true*
      end if
    end if
  end while
end for
if *found* **then**
  pkt ← *dequeue*(*flowid*)
  Send pkt to MAC layer
end if

Algorithm 11 Highest Velocity among Degraded Flows

Assume that there are k flows , and there is a queue corresponding to every flow
Upon receiving a packet p_j with flow id i
if *not full*(Q_i) **then**
  if *deadline*(p_j^i) > Current Time **then**
    enqueue$(p_j^i, queue(i))$, *numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
  else
    drop(p_j^i),*numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
    numOfDropped$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
  end if
else
  drop(p_j^i), *numOfPkt*$(Q_i) \leftarrow$ *numOfPkt*$(Q_i) + 1$
  numOfDropped$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
end if
Upon receiving a call back from MAC layer
$max1 \leftarrow 0, max2 \leftarrow 0$
for $i = Q_1$ to Q_k **do**
  found $\leftarrow$ *false*
  while *not empty*(Q_i) and *not*(*found*) **do**
    look up the front packet pkt in Q_i
    if *deadline*(*pkt*) < Current Time **then**
      drop(*pkt*), *numOfDropped*$(Q_i) \leftarrow$ *numOfDropped*$(Q_i) + 1$
    else
      calculate $MR(Q_i)$
      if $MR(Q_i) > missingRatioThreshold$ **then**
        if $max1 <$ *velocity*(*pkt*) **then**
          $max1 \leftarrow$ *velocity*(*pkt*), *flowId*$1 \leftarrow i$
        end if
      else
        if $max2 <$ *velocity*(*pkt*) **then**
          $max2 \leftarrow$ *velocity*(*pkt*), *flowId*$2 \leftarrow i$
        end if
      end if
      found $\leftarrow$ *true*
    end if
  end while
end for
if *found* **then**
  if $max1! = 0$ **then**
    pkt $\leftarrow$ *dequeue*(*flowId*1),*Send pkt to MAC layer*
  else
    pkt $\leftarrow$ *dequeue*(*flowId*2),*Send pkt to MAC layer*
  end if
end if

an algorithm that schedules the most punctual packet first. We proposed two variations of the well-known Earliest Deadline First, one that considers the punctuality of the packet in addition to its deadline, and another that considers the drop ratio of the packet's flow in addition to its deadline. We proposed a variation of Velocity Monotonic Scheduling called Highest Velocity First, and two further variations, one that considers the punctuality of the packet in addition to its velocity, and another that considers the drop ratio of the packet's flow in addition to its velocity. We implemented all these eleven algorithms in the NS-2 simulator and compared their performance experimentally. The results of the experiments are described in Chapter 5.

Chapter 4

Design and Implementation

In this chapter, we describe our implementation of the algorithms proposed in Chapter 3. First we will introduce a popular generic network simulator NS-2 (Network Simulator Version2) [8]. Then, we describe our implementation of algorithms in NS-2. Finally, we will propose a prioritized *IEEE* 802.11 MAC layer design.

4.1 Simulators for Wireless Sensor Networks

It is well known that simulators are very useful tools to support research in networking. First, they provide a practical environment that enables a researcher to verify algorithms or protocols before they are deployed and run on real hardware. In addition, since simulators provide an ideal experiment environment and many useful analyzing tools, implementing and analyzing an algorithm or a protocol in simulators are more efficient than in the real hardware. Another advantage of simulators is that researchers are able to verify algorithms

or protocols in a very large number of units, which may not be available in real deployment.

4.1.1 NS-2 Simulator

NS-2 is a popular generic open-source simulator that runs on Linux. It is a discrete event simulator aimed at networking research [8]. Currently, it is widely used in all kinds of network research areas such as, LAN, Internet and wireless sensor network. It provides many useful tools such as NAM, trace file and TCL. In addition, many protocols have already been implemented in this simulator for example, TCP, UDP, many routing algorithms, multicast protocols and IEEE 802.11 DCF. Therefore, we can concentrate on implementing and studying our scheduling algorithms.

NS-2 is an object-oriented simulator, which is written in C++. In addition, it uses an object TCL (OTCL) interpreter as a front end [52]. Therefore, there are two languages used in NS-2. C++ is used to build all kinds of algorithms and protocols. The TCL script language is used to build network scenarios, configure all kinds of parameters of the simulator and interact with the C++ implemented modules. Implemented protocols by researchers are built into NS-2 and become one of NS-2's modules.

The resulting simulator in a network scenario is an OTCL interpreter which is used to execute a user-provided TCL script. The simulation results are recorded into a trace file so that the researchers can use this file to analyze their protocol's performance. Also users can use NAM which is a Tcl/TK based animation tool to observe network simulation. It provides network topology layout, packet transmission animation, and various data inspection tools [7].

4.2 Real-time Algorithm Implementation

Before discuss the algorithm implementation, we show Figure 10 that illustrates a mobile node model in NS-2.

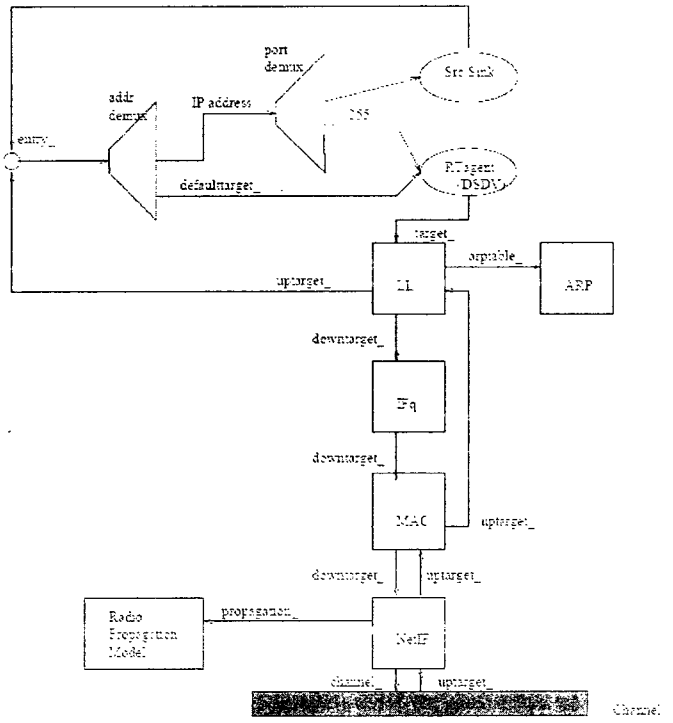

Figure 10: The schema of mobile node [52]

As illustrated in Figure 10, when a mobile node receives a packet from a physical communication channel, the packet will be sent from MAC module to LL module. In the LL module, the ARP protocol checks the MAC address of this packet to find its corresponding IP address. Then this packet is transferred to the upper layer.

When a forwarded packet or a generated packet comes down to LL module from the upper module, firstly the LL module transfers the packet to the IFq module. The main

function of IFq is to schedule packets. In addition, it also provides a buffer for packets. This buffer in NS-2 is a priority queue, which gives a higher priority to routing packets to find routing path. After IFq module, a selected packet enters the MAC module, which was implemented based on 802.11 DCF protocol.

In this thesis, we focus on the implementation of IFq module and prioritized MAC module. In this section, we mainly discuss the implementation of IFq module. The prioritized MAC implementation will be discussed in the next section. It is known that IFq module provides a buffer and a scheduling algorithm. Therefore, one of our main tasks is to implement our proposed algorithms to replace the original IFq algorithm. The architecture of the NS-2 MAC layer has been changed as shown in Figure 11. In this chart, the real-time algorithm module is between the LL module and the MAC module. It accepts packets from the LL module and selects a packet in terms of its scheduling algorithm. Then it transmits packets to the MAC module. The real-time scheduling module contains the implementation.

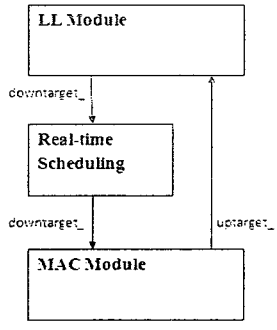

Figure 11: The architecture of NS-2 MAC layer

4.3 MAC-layer Prioritization Design and Implementation

Prioritizing each packet only in the real-time scheduling module is not sufficient in a wireless sensor network. The reason is that packets with different deadlines from different nodes will share a contention physical radio communication channel. Priorities determined in the scheduling module can be completely reordered due to contention for the wireless channel. In order to enhance real-time performance, it is therefore important to retain the priority in the MAC module. That is, a packet with higher priority should have a higher probability to grasp the communication channel. A new scheme *IEEE* 802.11e which defines a new MAC protocol for quality of service in wireless network had been proposed [36]. Another scheme AMPA (Adaptive MAC Parameters) dynamically changed *IEEE* 802.11 MAC's parameters to improve a contention based channel access performance [10].

First of all, we will introduce a priority for each packet in the MAC module. In [34], a packet is prioritized in three pre-defined ranges. However, one shortcoming of this prioritizing method is that its scalability is so limited that it only can be used in certain circumstances. Therefore, we propose a new packet prioritizing method. In our proposal, first we will calculate each packet's sending priority $MP(p_k^i)$, which we mentioned in chapter 3. Then we will prioritize packets based on their sending priorities. In Figure 8 the packet will get higher priority if its sending priority is higher. Plus we observe that the sending priority $SP(p_k^i)$ is between 0 and 1. Therefore, it is easier to dynamically adjust the packet's priority than in RAP [34].

Secondly, we implement two extensions which were proposed by Ada and Castelluccia [11]. One is the initial wait time after the channel becomes idle. The other is the back-off window increase function. Due to both extension, a packet with higher priority will get a higher chance to access the communication channel. The mapping between packet sending priority and MAC priority is listed in table 7:

MAC Priority	Sending Priority
1	$0.5 - 1$
2	$0.25 - 0.5$
3	$0 - 0.25$

Table 7: The mapping between packet sending priority and MAC priority

The initial wait time (DIFS) is a time interval between the time channel becomes idle and the time to send a RTS. Therefore, we may adjust packets' priorities by appropriately setting the DIFS value. The new equation of DIFS is as follows:

$$DIFS = DIFS \times MAC\ Priority \tag{10}$$

The packet with higher priority will have a low value in this equation. Therefore, it will be sent after a smaller waiting time.

The original back-off window increase function in NS-2 is that

$$CW = 2 \times CW + 1 \tag{11}$$

CW: contention window size.

From the above equation, the contention window will be doubled when a transmission

collision occurs. Our new equation will be changed as:

$$CW = MAC\ Priority \times CW + 1 \tag{12}$$

In this equation, the contention window size of a packet with higher priority (corresponding to a lower priority in the equation) increases slower than a node with a lower priority.

In summary, in order to improve the real-time performance, we design some modifications to the original *IEEE* 802.11 DCF. These implementations make a higher priority packet to have more opportunity to catch a communication channel in a contention based channel access environment.

4.4 Summary

In this chapter, first we indicate why we choose NS-2 as our experiment simulator. Then, we explain how to implement our proposed algorithms using NS-2. Finally, we develop a prioritized *IEEE* 802.11 DCF protocol to enhance the real-time performance.

Chapter 5

Experiments and Evaluation

In this chapter, we describe our experiments with the eleven algorithms described in Chapter 3. We implemented these algorithms using Network Simulator (NS2 Version 2.33 [8]).

We use two main performance metrics to evaluate the algorithms: the *packet drop ratio* and the *fairness* of the scheduling. A brief discussion of these two metrics follows.

The *overall packet drop ratio* is the ratio between the total number of packets dropped and the total number of packets sent. Packets can be dropped due to a variety of reasons. One main reason for dropping a packet is that it missed its deadline either when it reaches the destination or at an intermediate node. As shown in Algorithms 6, 5, 8, 9, 11, packets can be dropped by the scheduling algorithm if they are past their deadline either when they are enqueued, or when they are dequeued to be sent to the MAC layer. In Earliest Deadline among Punctual Packets and Highest Velocity among Punctual Packets algorithms, packets can also be dropped because they are not considered punctual enough, even though they may not yet have missed their deadline. Secondly, packets may be dropped because of

buffer overflow. Finally, packets can be dropped for a variety of reasons at other modules of NS-2 not implemented by us, such as at the MAC layer and at the Routing layer. Table 8 shows the percentage of drops due to various causes as experienced by the Single-Queue Deadline-Unaware Algorithm.

	Scheduling Algorithm		*NS − 2*		
Drops	*Overflow*	*Missed Deadline*	*MAC Drops*	*Routing Drops*	*Unknown*
Percentage	9.95%	32.6%	7.3%	0.4%	0.6%

Table 8: The percentage of all drops for the Single-Queue Deadline-Unaware algorithm

Table 9 shows the percentage of drops due to various causes as experienced by Earliest Deadline First.

	Scheduling Algorithm		*NS − 2*		
Drops	*Overflow*	*Missed Deadline*	*MAC Drops*	*Routing Drops*	*Unknown*
Percentage	0.17%	34.8%	7.2%	0.4%	0.42%

Table 9: The percentage of all drops for Earliest Deadline First

Since we are concerned with the scheduling algorithm in this thesis, apart from the overall drop ratio, we also evaluate algorithms based on the ratio between packets dropped by the scheduling algorithms given in Chapter 3 due to their missed deadlines and the total number of packets sent. We call this ratio *the Missed Deadline Drop ratio*. For Earliest_Deadline among Punctual Packets and Highest Velocity among Punctual Packets we separately count the packets dropped due to their not being punctual; we call this ratio *the Unpunctual Drop ratio*.

In addition, we are also interested in the fairness of the algorithms to different flows.

We assume all flows have the same priority and produce packets at the same rate. Therefore, they should have similar drop ratios. We use the well-known Jain's Fairness Index to evaluate the fairness of each algorithm [26]. We denote the ratio of received packets of each flow as x_i, and the number of flows as n. The expression of Jain's Fairness Index is given below:

$$Fairness = \frac{(\sum_{i=0}^{n} x_i)^2}{n \times \sum_{i=0}^{n} x_i^2} \tag{13}$$

It is easy to see that the value of the index was between 0 and 1, and a higher value corresponds to a fairness algorithm. To summarize, we measure the real-time performance in terms of the following criteria:

1. Overall Packet Drop Ratio: the ratio between the total number of packets that do not reach their destinations before their deadlines and the total number of packets sent

2. Missed Deadline Drop Ratio: the ratio between the total number of packets dropped due to a missed deadline at intermediate nodes or at the destinations and the total number of packets sent

3. Unpunctual Drop Ratio: the ratio between the total number of packets dropped due to their not being punctual enough either at intermediate nodes or at the destinations and the total number of packets sent (if relevant)

4. Jain's Fairness Index

In Section 5.1, we focus on the simulation results of the four baseline algorithms to find the one with the best performance. Then, in Section 5.2, Section 5.3 and Section 5.4, we

compare the best baseline algorithm with our real-time algorithms proposed in Chapter 3, and study these algorithms' real-time performance. All these experiments use the original *IEEE* 802.11 DCF. Finally, in Section 5.6 we evaluate real-time scheduling algorithms with the prioritizing *IEEE* 802.11 protocol.

Table 10 shows the simulation parameters we used in this thesis. We randomly deploy a number of sensor nodes in a rectangle area in different experimental scenarios to simulate our algorithms. The source and destination of traffic flows (CBR) are randomly selected as well. We calculate the performance criteria for each scenario. The results described for overall packet drop ratio are averaged over one hundred scenarios, and the results for fairness are averaged over ten scenarios.

Transmission Radio Range	250m
Packet Size	128B
Data Rate	11 packet/sec
Simulation Area	$1000 \times 1000\ m^2$
Number of Sensor Nodes	60,80,100,120,140
Number of Flows	15
Simulation Time	500s
Mac Layer Protocol	*IEEE* 802.11 DCF
Routing Layer Protocol	AODV

Table 10: Simulation parameters

5.1 Baseline Algorithms

In this section, we describe our experiments on our baseline algorithms to see which algorithm has the best real-time performance. The simulation results are presented in Figure 12

and Figure 13.

Figure 12 shows the overall packet drop ratio of the four baseline algorithms for different node densities. In this figure, we can find that the Single-Queue Deadline-Aware algorithm has the lowest overall packet drop ratio among baseline algorithms, and the Multiple-Queue Deadline-Unaware algorithm obtains the highest overall packet drop ratio among baseline algorithms.

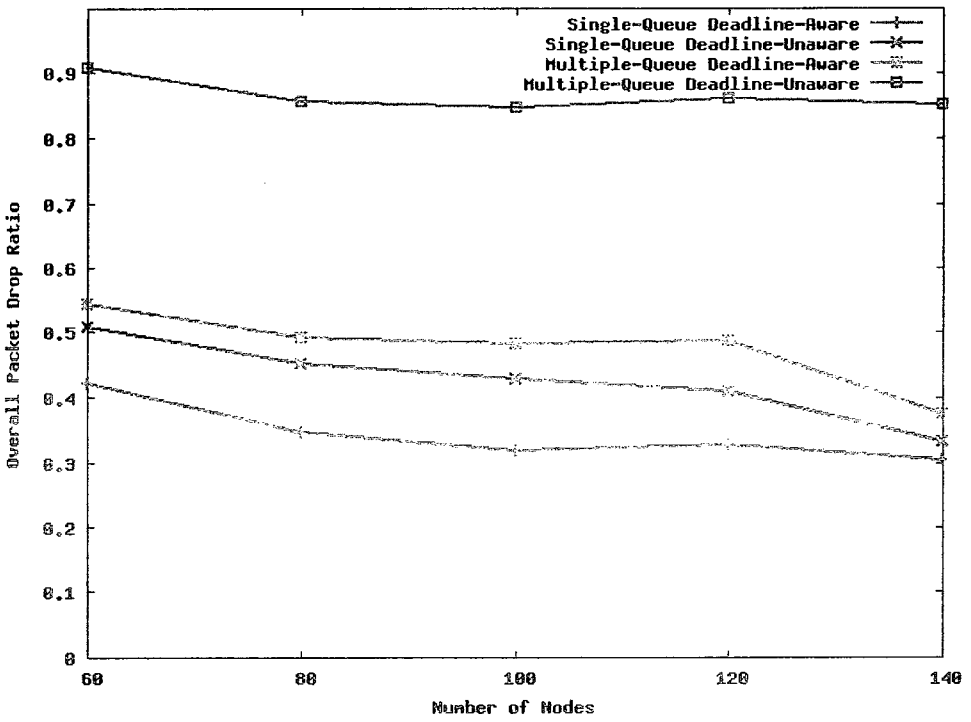

Figure 12: The overall packet drop ratio for baseline algorithms

Figure 13 shows the missed deadline drop ratio of four baseline algorithms for different node densities. In this figure, we can see that the Single-Queue Deadline-Aware algorithm has the lowest missed deadline drop ratio among these four baseline algorithms, and the

Multiple-Queue Deadline-Unaware algorithm obtains the highest missed deadline drop ratio. In the Multiple-Queue Deadline-Unaware algorithm and the Single-Queue Deadline-Unaware algorithm, all incoming packets will be forwarded regardless of whether their deadlines have already been missed at intermediate nodes. As expected, their Deadline-Aware counterparts have a lower overall packet drop ratio as well as missed deadline drop ratio. In addition, we may observe that Single-Queue algorithms have better performance than Multiple-Queue algorithms. Recall that the Round-robin policy is implemented in Multiple-Queue algorithms. With the Round-Robin scheduling, a node forwards packets based on the order instead of the deadline of the packet. Therefore, many urgent packets can't be sent on time resulting in a high drop ratio. The initial motivation to use Round-Robin policy is to improve the fairness. However, we can discover that the real-time performance impacts the fairness. Table 11 illustrates the fairness of four baseline algorithms. Clearly, the Single-Queue Deadline-Aware algorithm improves the system performance compared with other baseline algorithms in terms of both drop ratio and fairness.

Algorithms	Single-Queue Deadline Aware	Single-Queue Deadline Unaware	Multiple-Queue Deadline-Aware	Multiple-Queue Deadline-Unaware
Fairness	0.765	0.625	0.732	0.53

Table 11: Fairness for four baseline algorithms

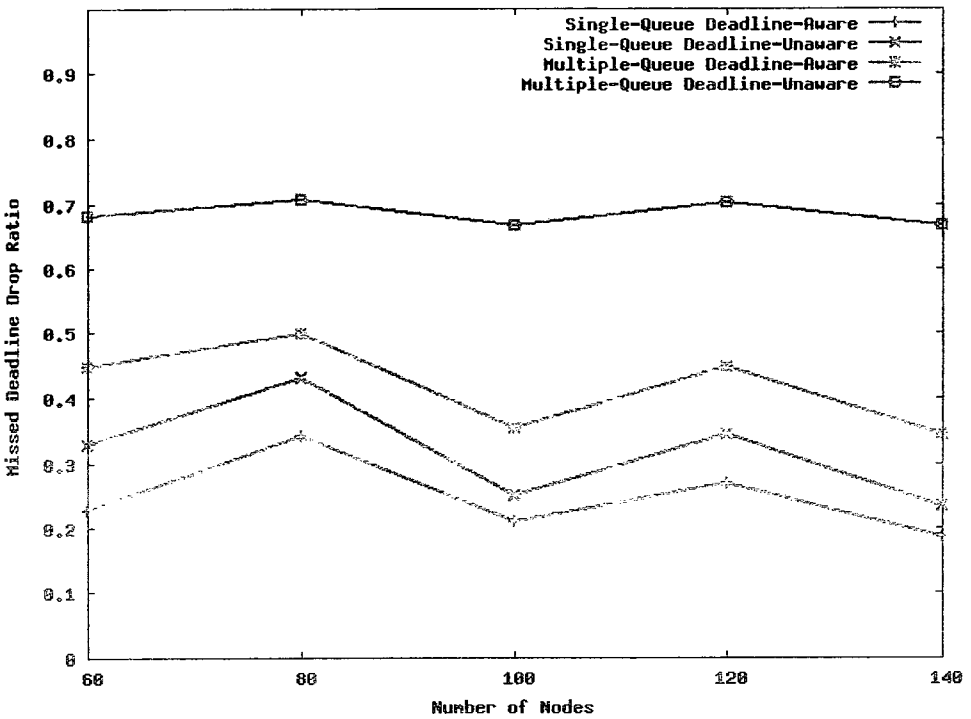

Figure 13: The missed deadline drop ratio for baseline algorithms

5.2 Pure Heuristics

In this section, we compare the overall packet drop ratio of the Single-Queue Deadline-Aware algorithm with Earliest Deadline First, Most Punctual First and Highest Velocity First. As shown in Figure 14, we can find that Highest Velocity First has the lowest overall packet drop ratio among these four algorithms. Earliest Deadline First algorithm has a drop ratio similar to the baseline algorithm. Our two proposed algorithms have lower overall packet drop ratio than the other two algorithms.

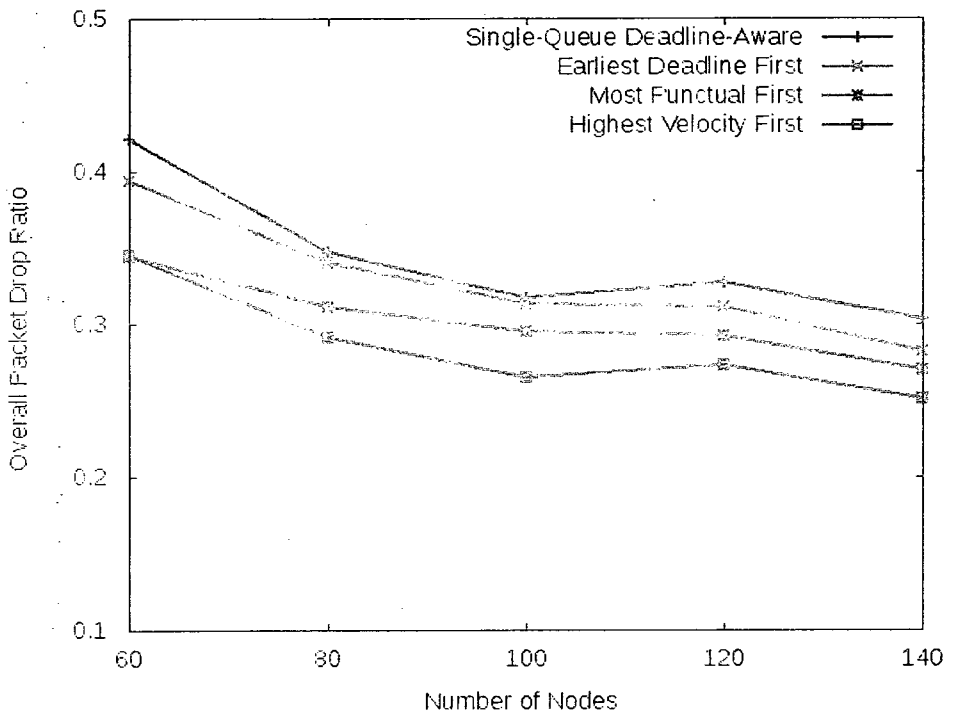

Figure 14: The overall packet drop ratio for Single-Queue Deadline-Aware, Earliest Deadline First, Most Punctual First and Highest Velocity First

In Figure 14, we find that Earliest Deadline First has a drop ratio similar to Single-Queue Deadline-Aware algorithm. We may observe that as the node density increases, the number of traffic flows going through one node could decrease. In addition. the packets in each traffic flow is scheduled in the Single-Queue Deadline-Aware algorithm following the earliest-deadline-first policy. Therefore, we investigate the performance of these algorithms for a higher traffic flow density. Figure 15 shows the overall packet drop ratio of Earliest Deadline First, the Single-Queue Deadline-Aware algorithm, Most Punctuality First and Highest Velocity First, for scenarios with 50 traffic flows.

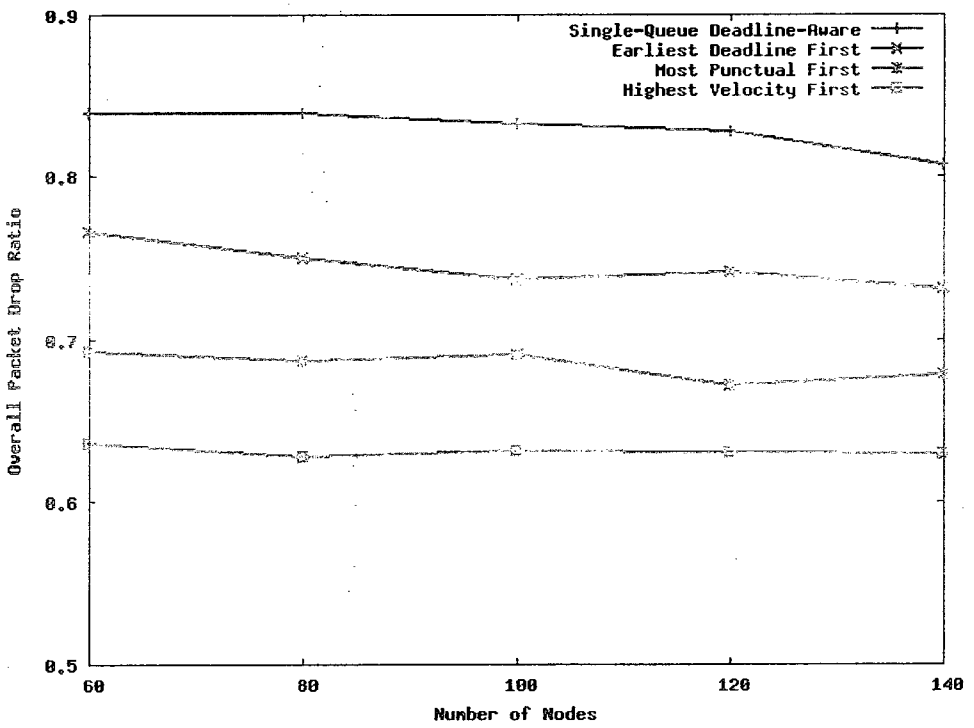

Figure 15: The overall packet drop ratio for Single-Queue Deadline-Aware, Earliest Deadline First, Most Punctuality First and Highest Velocity First with 50 traffic flows

5.3 Variations of Earliest Deadline First

In this section, we present the real-time performance of Earliest Deadline First, Most Punctual First, Earliest Deadline among Punctual Packets and Earliest Deadline among degraded Flows. The last two algorithms use certain thresholds as parameters. We experimentally determine the best value of these thresholds.

First, we compare the overall packet drop ratio and the unpunctual drop ratio of Earliest Deadline among Punctual Packets with different punctuality thresholds for different node density scenarios. Figure 16 shows the experimental results of the overall packet drop ratio

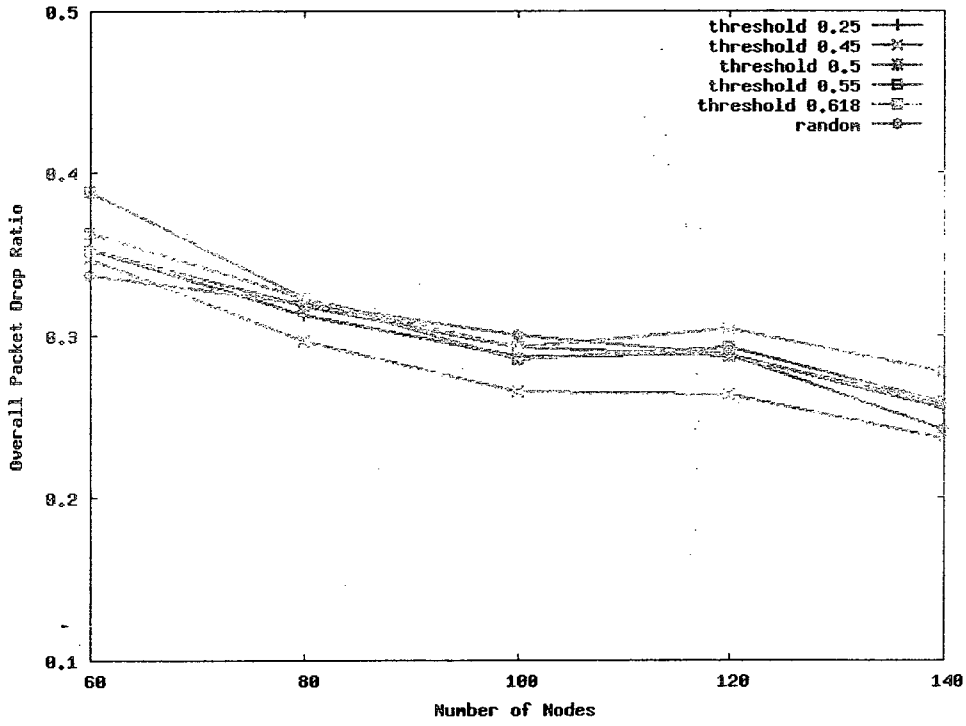

Figure 16: The overall drop ratio for Earliest Deadline among Punctual Packets with different punctuality thresholds

of this algorithm with different punctuality thresholds. We can observe that the overall packet drop ratio becomes the lowest when the threshold is 0.45.

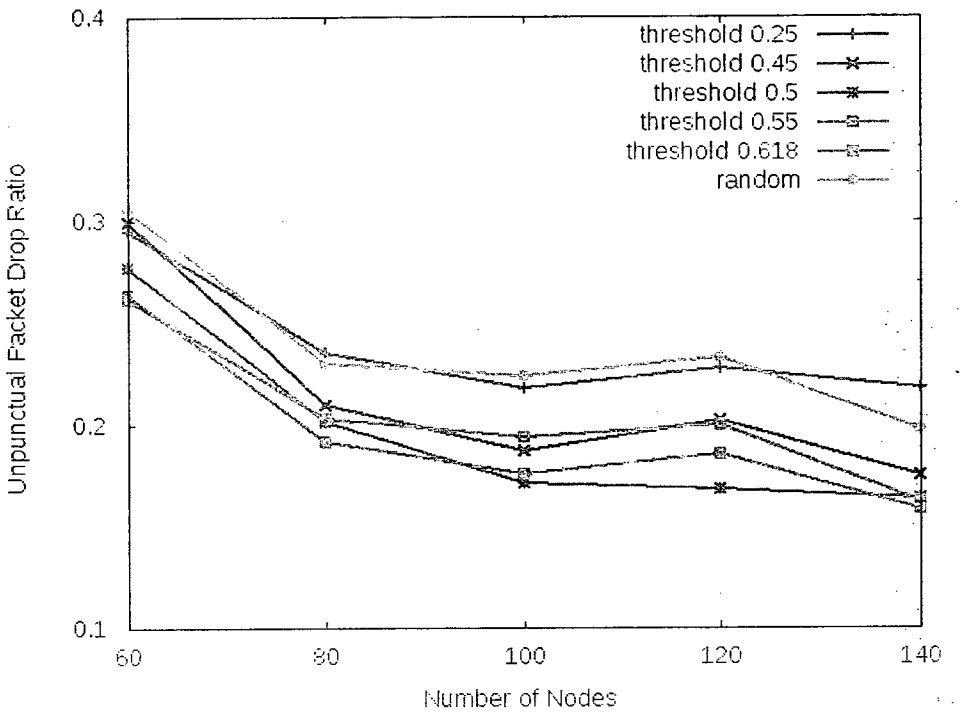

Figure 17: The unpunctual drop ratio for Earliest Deadline among Punctual Packets with different punctuality thresholds

Figure 17 indicates the unpunctual drop ratio of Earliest Deadline among Punctual Packets with different punctuality thresholds in different node density scenarios. Combining Figure 16 and Figure 17, we can find that dropping appropriate amount of unpunctual packets could decrease the overall packet drop ratio. Then, we study the fairness of this algorithm with different punctuality thresholds. The fairness of each value of threshold is given in Table 12. We see that using a threshold value of 0.45 also gives a high value of fairness.

Threshold	0.25	0.45	0.5	0.55	0.618	*random*
Fairness	0.752	0.773	0.781	0.734	0.7338	0.767

Table 12: Fairness for Earliest Deadline among Punctual Packet with different punctuality threshold

Next, we find the best value of threshold for Earliest Deadline among Degraded Flows. Recall that we introduce a value called *drop ratio Threshold* to dynamically adjust the priority of each traffic flow within intermediate node to prevent the low priority traffic flow from starvation. Table 13 presents *Jain's Fairness Index* with different *drop ratio Thresholds* of Earliest Deadline among Degraded Flows. In Table 13, when *drop ratio Threshold* is set as 0.25 in the algorithm, the system has the best fairness. In addition, we study the overall packet drop ratio for this algorithm with different *drop ratio Thresholds* which is illustrated in Figure 18. It can be observed that the value of threshold does not significantly affect the overall drop ratio.

Threshold	0.2	0.25	0.375	0.5	*no threshold*
Fairness	0.869	0.876	0.849	0.778	0.765

Table 13: Fairness for Earliest Deadline among Degraded Flows

Finally, we compare the performance of Earliest Deadline First, Most Punctual First, Earliest Deadline among Punctual Packets with threshold 0.45 and Earliest Deadline among Degraded Flows with threshold 0.25. The last two algorithms are using the best values for their thresholds for overall packet drop ratio. Figure 19 shows the results for different node densities. We can find that Earliest Deadline among Punctual Packets has the best overall drop ratio among these four algorithms.

Then, we present the fairness of these four algorithms in Table 14. We can find out that

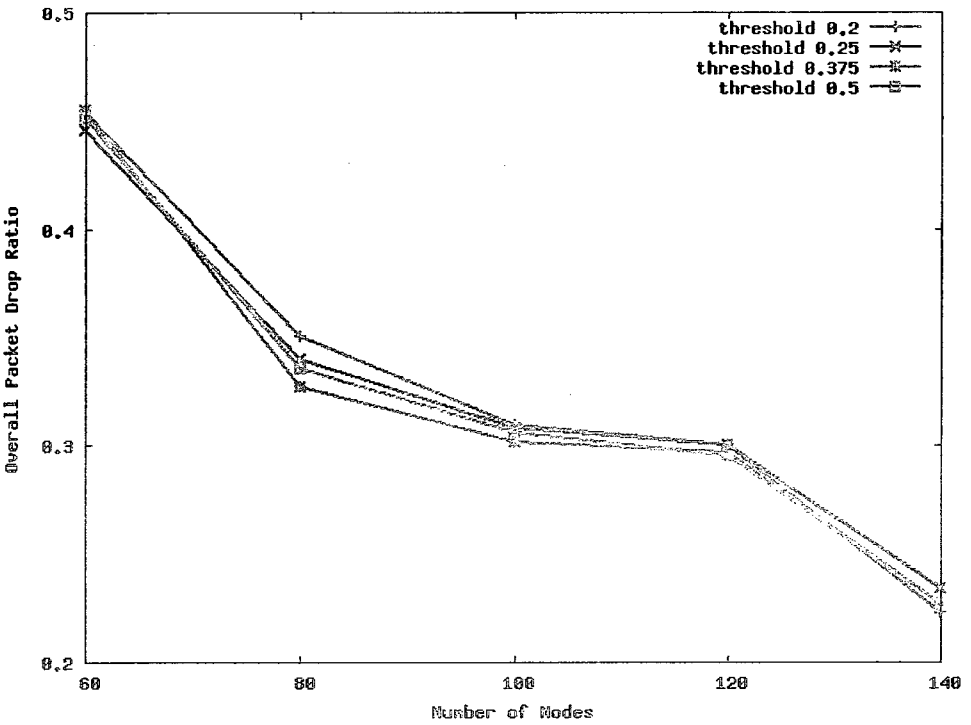

Figure 18: The overall packet drop ratio for Earliest Deadline among Degraded Flows with different thresholds

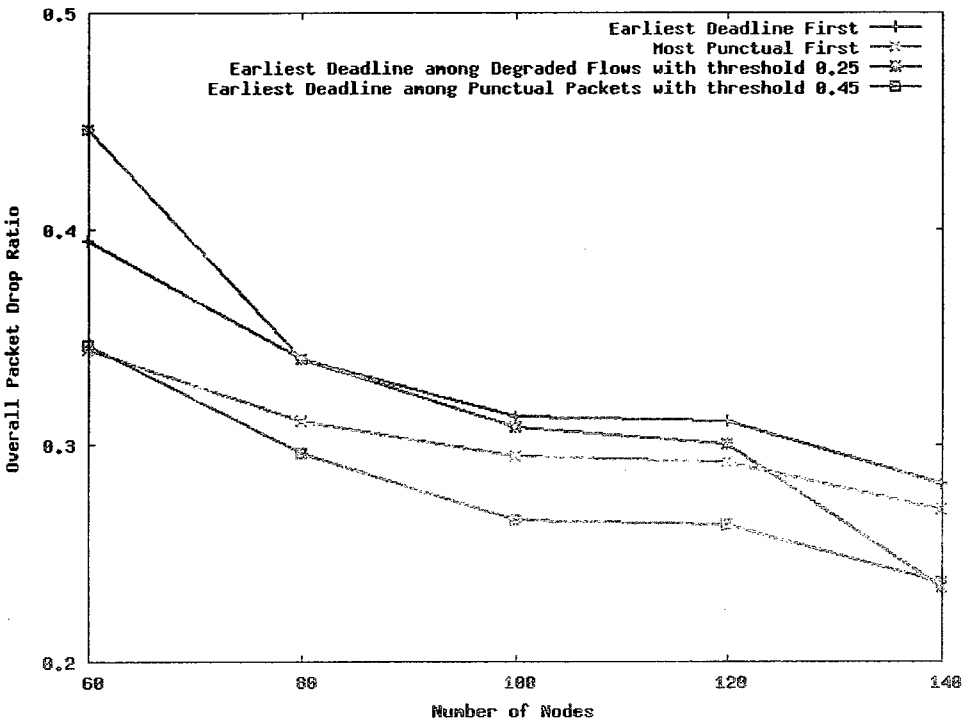

Figure 19: The overall packet drop ratio for Earliest Deadline First, Most Punctual First, Earliest Deadline among Punctual Packets with threshold 0.45, and Earliest Deadline among Degraded Flows with threshold 0.25

Earliest Deadline among Degraded Flows has the best fairness. On the contrary, Earliest Deadline First has the worst fairness.

Algorithms	*Earliest Deadline First*	*Earliest Deadline among Degraded Flows with threshold 0.25*	*Most Punctual First*	*Earliest Deadline among Punctual Packets with threshold 0.45*
Fairness	0.765	0.876	0.772	0.773

Table 14: Fairness for four algorithms

In summary, from the above experimental results, we find that our proposed algorithms did improve the system real-time performance compared with the original Earliest Deadline First algorithm. In addition, we find that there is a tradeoff between fairness and overall packet drop ratio.

5.4 Variations of Highest Velocity First

In this section, we study the real-time performance of Highest Velocity First, Highest Velocity among Punctual Packets and Highest Velocity among Degraded Flows. The last two algorithms use certain thresholds as parameters. We experimentally determine the best values of these thresholds.

First, we compare the overall packet drop ratio and the unpunctual drop ratio of Highest Velocity among Punctual Packets with different punctuality thresholds for different node density scenarios. Compared with Highest Velocity First, which only considers the remaining time and remaining distance of packets, Highest Velocity among Punctual Packets considers not only the remaining time and remaining distance of packets but also the packet's

travelling history.

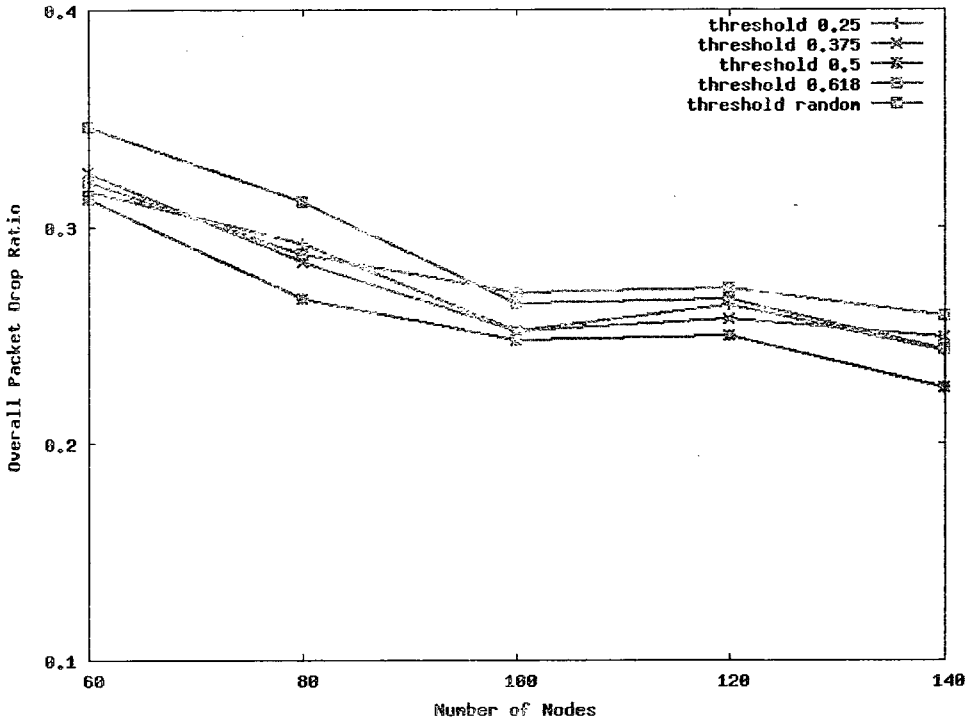

Figure 20: The overall packet drop ratio for Highest Velocity among Punctual Packets with different thresholds

Figure 20 illustrates the overall packet drop ratio of this algorithm with different punctuality thresholds. From this figure, we can observe that overall packet drop ratio becomes the lowest when *punctuality threshold* is 0.5. Figure 21 indicates the unpunctual drop ratio of Highest Velocity among Punctual Packets with different punctuality thresholds for different node density scenarios. Then, we study the fairness of this algorithm with different punctuality thresholds. In Table 15, we find out all values of threshold result in similar value of fairness. Although a threshold value of 0.5 does not have the best fairness in the observed set, it is still not too far from the best in terms of fairness.

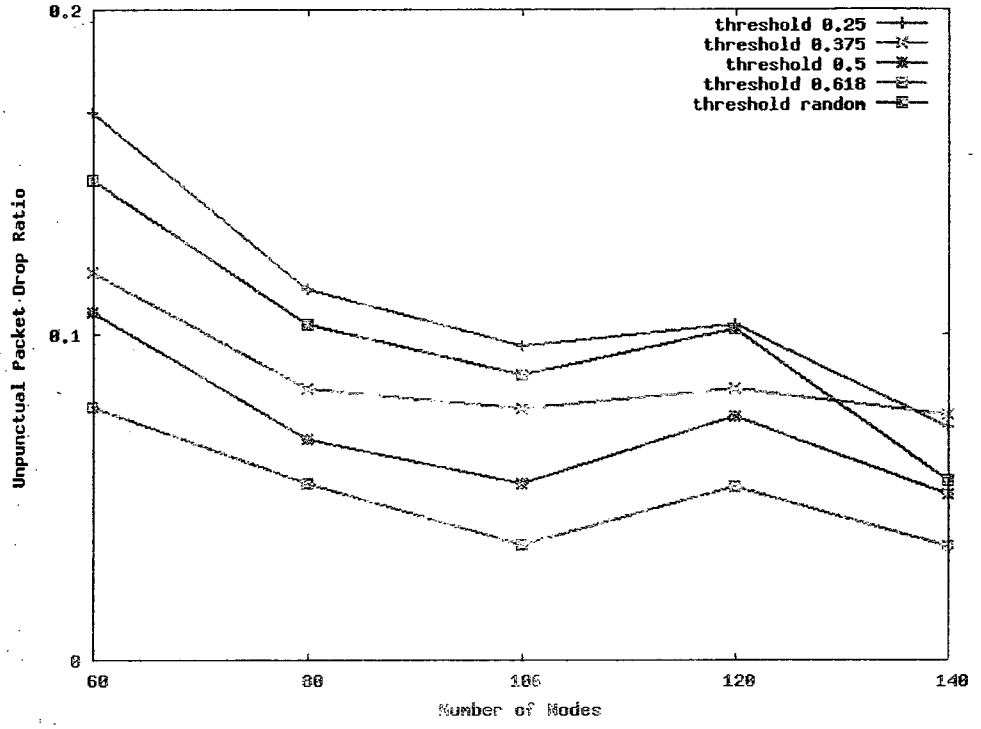

Figure 21: The unpunctual drop ratio for Highest Velocity among Punctual Packets with different thresholds

Threshold	0.25	0.375	0.5	0.618	*random*
Fairness	0.776	0.788	0.768	0.75	0.783

Table 15: Fairness for Most Velocity among Punctual Packet with different punctuality threshold

Next, we find the best value of threshold for the Highest Velocity among Degraded Flows. Table 16 presents *Jain's Fairness Index* with different *drop ratio Thresholds* of Highest Velocity among Degraded Flows. In Table 16, we observe that when the threshold

Threshold	0.05	0.1	0.15	0.2	0.5	*no threshold*
Fairness	0.836	0.844	0.839	0.821	0.77	0.759

Table 16: Fairness for Highest Velocity among Degraded Flows with difference thresholds

is set as 0.1, this algorithm shows the best fairness. In addition, we study the overall packet drop ratio for this algorithm with different *drop ratio Thresholds*, which is illustrated in Figure 22. We observe that while the overall packet drop ratios are not too different for different values of threshold, there may be a tradeoff between fairness and overall packet drop ratio, as the threshold value of 0.1 has the worst overall packet drop ratio. We chose a threshold value of 0.15 since it has almost the best performance in terms of both overall packet drop ratio and fairness and therefore provides the best balance.

Next, we compare the overall packet drop ratio of Highest Velocity First, Highest Velocity among Punctual Packets with threshold 0.5 and Highest Velocity among Degraded Flows with threshold 0.15. We use the best thresholds for the last two algorithms. Figure 23 shows the results for different node densities. We can see that Highest Velocity among Punctual Packets has the best overall packet drop ratio among these three algorithms. We also find that Highest Velocity First has the highest overall packet drop ratio.

Then, we present the fairness of these three algorithms in Table 17. We can find that Highest Velocity among Degraded Flows has the best fairness. On the contrary, Highest

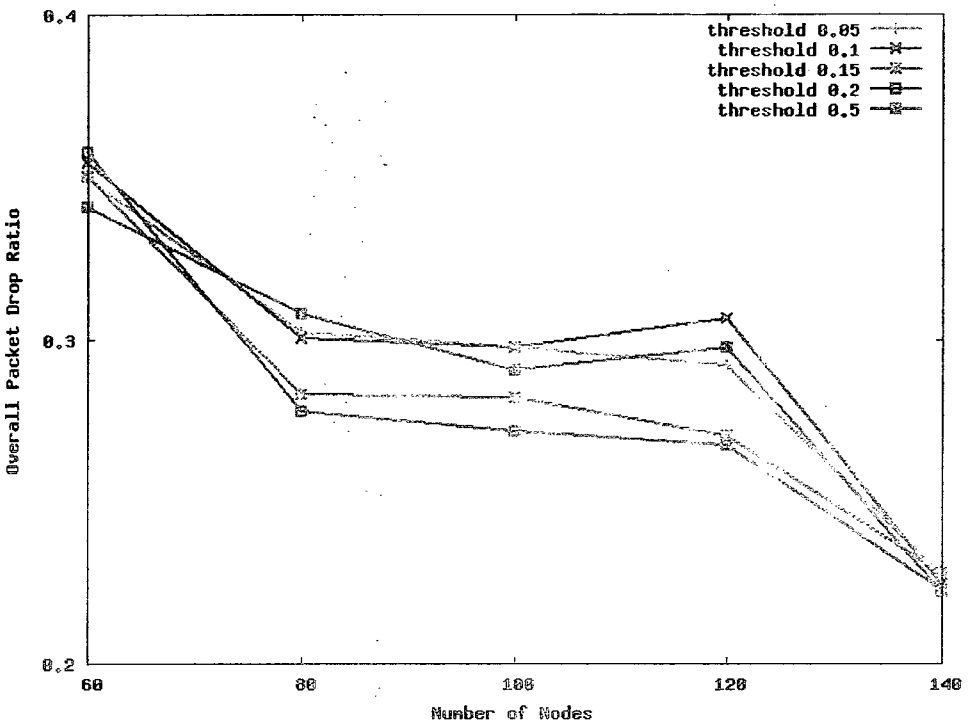

Figure 22: The overall packet drop ratio for Highest Velocity among Degraded Flows with different thresholds

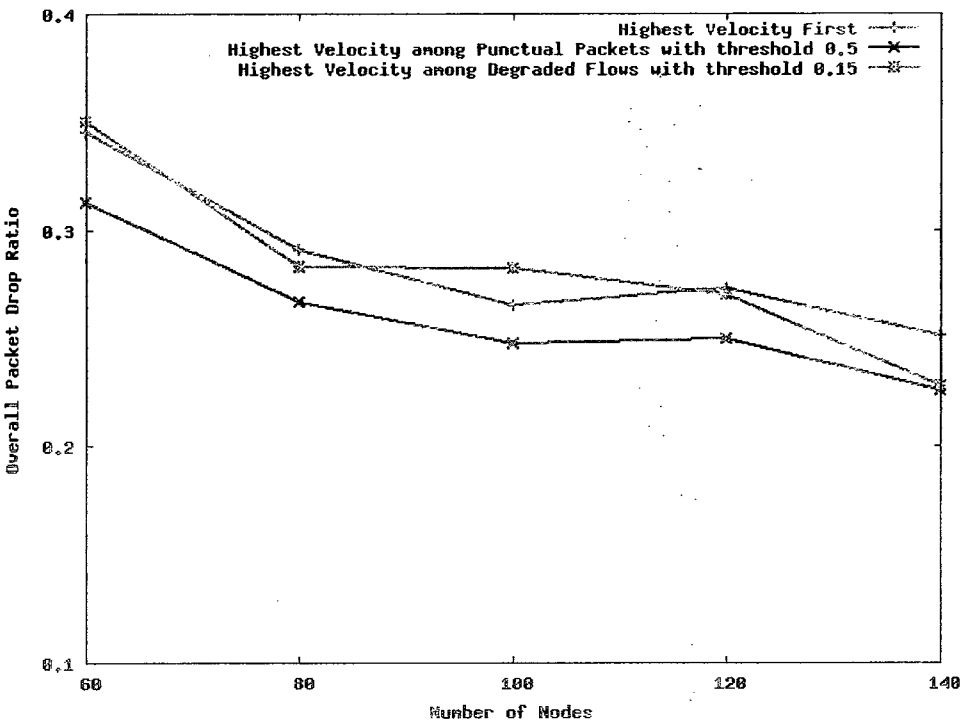

Figure 23: The overall packet drop ratio for Highest Velocity First, Highest Velocity among Punctual Packets, and Highest Velocity among Degraded Flows

Velocity First has the worst fairness.

Algorithms	*Highest Velocity First*	*Highest Velocity among Punctual Packets with threshold 0.5*	*Highest Velocity among Degraded Flows with threshold 0.15*
Fairness	0.759	0.768	0.839

Table 17: Fairness for three algorithms

5.5 Comparison of Best Algorithms

In the previous section, we demonstrated the performance of the variations of Earliest Deadline First and variations of Highest Velocity First. In this section, we compare the overall packet drop ratio of these two sets of algorithms.

Figure 24 presents the overall packet drop ratio of these two sets of algorithms. We may observe that Highest Velocity among Punctual Packets First has the lowest overall packet drop ratio.

Next, we study the fairness of these algorithms. In Table 18, we may observe that Earliest Deadline among Degraded Flows with threshold 0.25 has the best fairness.

From the above figures and table, we can find that the algorithms based on Highest Velocity First have lower overall packet drop ratios than these based on Earliest Deadline First. The reason is that the latter only consider one dimension, the time. However, Highest Velocity First consider two dimensions, the time and the distance. We conclude than Highest Velocity among Punctual Packets achieve the lowest overall packet drop ratio, while Earliest Deadline among Degraded Flows achieve the best fairness. The best balance

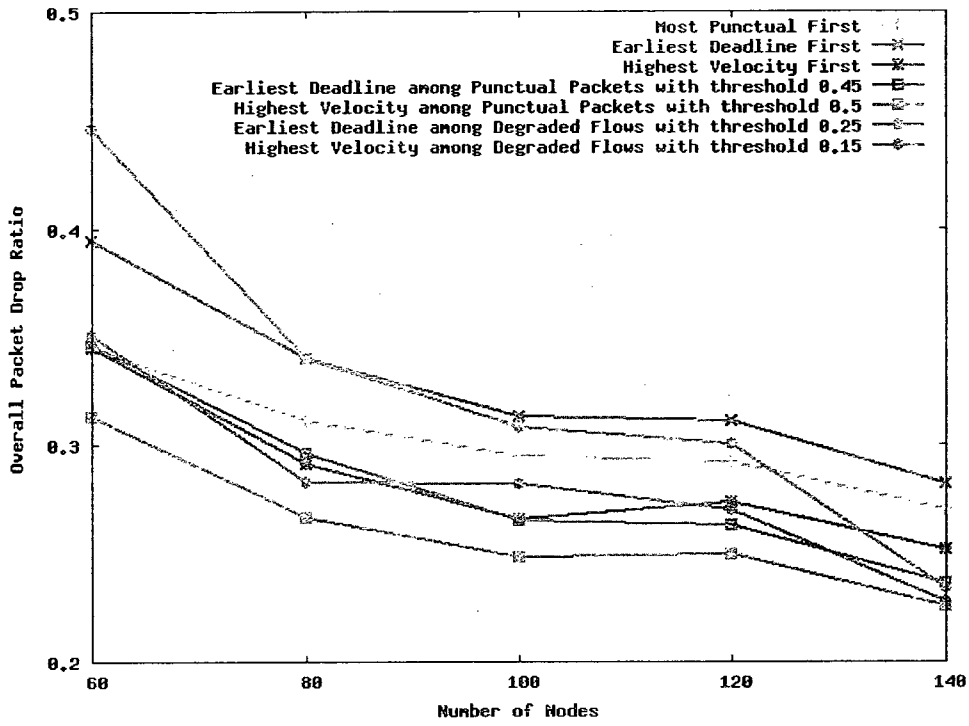

Figure 24: The overall packet drop ratio for variations of Earliest Deadline First and variations of Highest Velocity First

Algorithms	*Highest Velocity First*	*Earliest Deadline First*	*Most Punctual First*	*Highest Velocity among Punctual Packets with threshold 0.5*	*Highest Velocity among Degraded Flows with threshold 0.15*	*Earliest Deadline among Degraded Flows with threshold 0.25*	*Earliest Deadline among Punctual Packets with threshold 0.45*
Fairness	0.759	0.765	0.772	0.768	0.839	0.876	0.773

Table 18: Fairness for seven algorithms

between overall packet drop ratio and fairness is achieved by Highest Velocity among Degraded Flows.

5.6 Using *IEEE* 802.11 with prioritizing extension

In order to efficiently improve the real-time performance, it may be helpful to prioritize the packet in *IEEE* 802.11 protocol. In this section, we will demonstrate the simulation results of our algorithms with *IEEE* 802.11 with prioritizing extension.

In Figure 25, we demonstrate the overall packet drop ratio of Earliest Deadline First, Earliest Deadline among Punctual Packets and Earliest Deadline among Punctual Packets with MAC Prioritizing Extension.

From Figure 25 we can observe that the overall packet drop ratio has been efficiently decreased after MAC Prioritizing Extension is implemented. Therefore, we may realize that the MAC layer plays a very important role in real-time system design.

In Figure 26, we demonstrate the overall packet drop ratio of Highest Velocity First, Highest Velocity among Punctual Packets and Highest Velocity among Punctual Packets with MAC Prioritizing Extension. We can observe that Highest Velocity among Punctual Packets with MAC Extension algorithm has the best performance. It is clear that prioritizing packets at the MAC layer in addition to the scheduling algorithm can further improve performance.

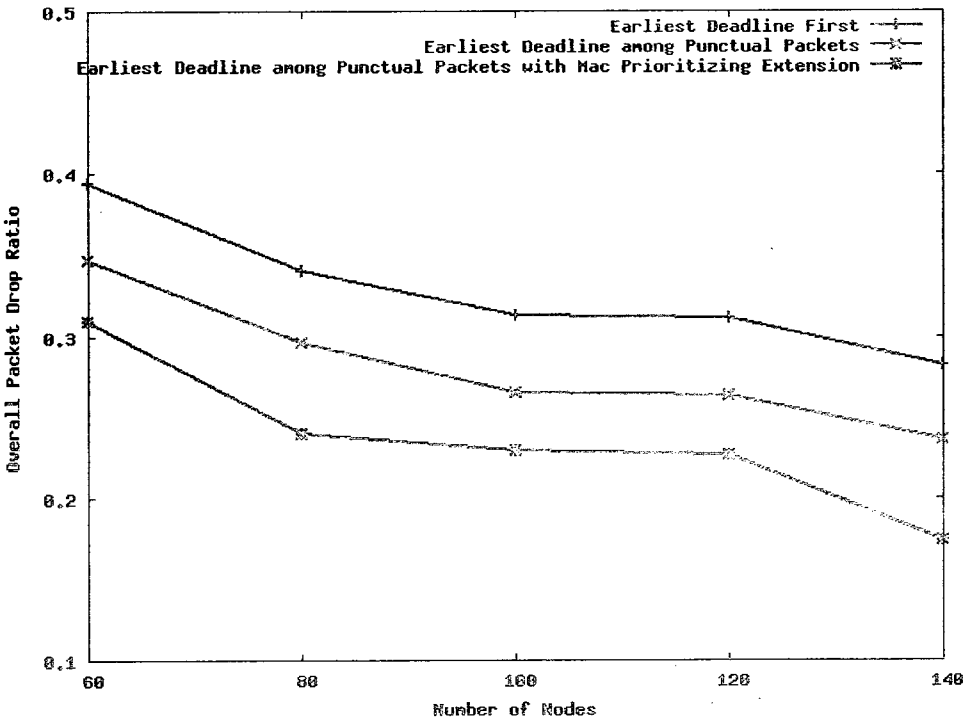

Figure 25: The overall packet drop ratio for EDF, Earliest Deadline among Punctual Packets and Earliest Deadline among Punctual Packets with MAC prioritizing extension

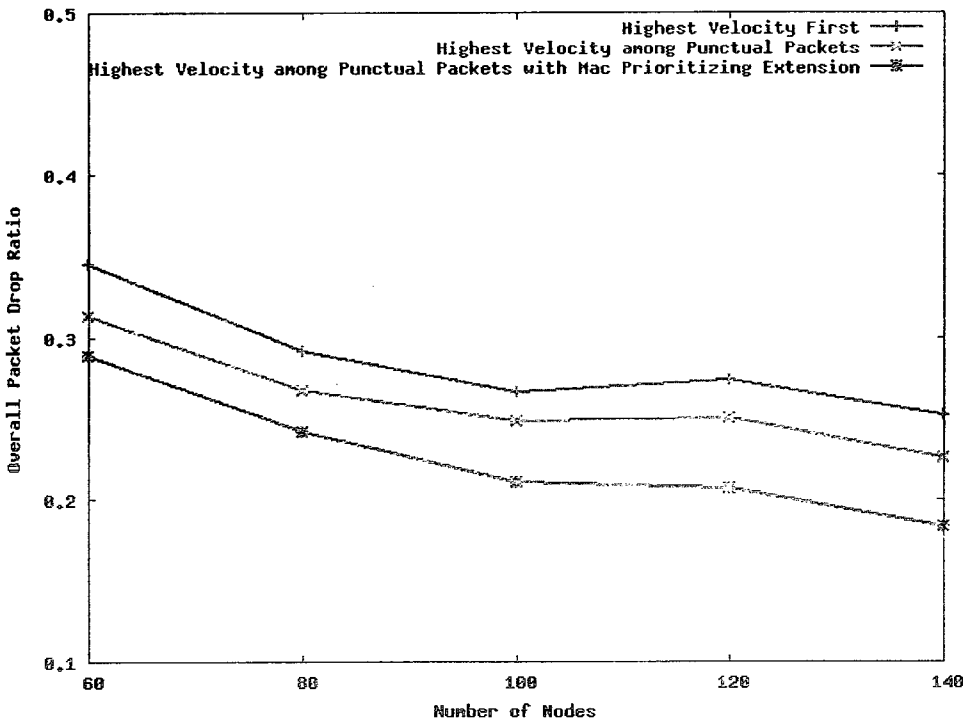

Figure 26: The overall packet drop ratio for Highest Velocity First, Highest Velocity among Punctual Packets and Highest Velocity among Punctual Packets with MAC prioritizing extension

5.7 Summary

In this chapter we describe a series of experimental results on eleven scheduling algorithms by using the NS-2 simulator. Our results show that Highest Velocity among Punctual Packets achieve the lowest overall packet drop ratio, while Earliest Deadline among Degraded Flows achieve the best fairness. The best balance between overall packet drop ratio and fairness is achieved by Highest Velocity among Degraded Flows.

We may observe that the overall packet drop ratio is decreased when the algorithm considers the punctuality of the packet. In addition, dynamically adjusting the priority of traffic in terms of their drop ratio can improve the fairness, and does not affect their overall packet drop ratio too much. We also showed that using a prioritizing extension in the 802.11 MAC protocol can further decrease the overall packet drop ratio.

Chapter 6

Conclusions and Future Work

In this thesis, we proposed and implemented many real-time scheduling algorithms for wireless sensor networks. Our algorithms take into account the deadline of packets, then required velocities, then punctuality, the per-flow drop ratio, or some combinations thereof. The performance of these algorithms have been extensively evaluated. The experimental results indicate that our proposed real-time scheduling algorithms can efficiently decrease the overall packet drop ratio and improve fairness. The improvements in these two performance metrics are very desirable in wireless sensor networks. We implemented our algorithms in NS-2, which is a widely used open source software. Our results show that Highest Velocity among Punctual Packets achieve the lowest overall packet drop ratio, while Earliest Deadline among Degraded Flows achieve the best fairness. The best balance between overall packet drop ratio and fairness is achieved by Highest Velocity among Degraded Flows.

Many challenges still remain in wireless sensor networks. Intensive research is ongoing in the field of hardware, MAC protocols, real-time scheduling and routing algorithms, among many others. Adapting our algorithms to work with a mixture of real-time and non-real-time traffic would be the next step in the research described in this thesis. It would also be interesting to incorporate position information, such as that used by geometric routing algorithms [22] [18] into our work. In addition, we did not measure our proposed algorithms on real hardware environment because of limitations on the available time. Conducting this measurement and comparison is a worthwhile future work.

Bibliography

[1] http://en.wikipedia.org/wiki/tinyos.

[2] http://en.wikipedia.org/wiki/wireless_sensor_network.

[3] http://sentilla.com/files/pdf/eol/tmote-sky-brochure.pdf.

[4] http://sentilla.com/files/pdf/eol/tmote-sky-datasheet.pdf.

[5] http://www.cse.iitk.ac.in/users/braman/.

[6] http://www.globalsecurity.org/military/systems/ground/rembass.htm.

[7] http://www.isi.edu/nsnam/nam/.

[8] http://www.isi.edu/nsnam/ns/.

[9] http://www.sics.se/contiki/about-contiki.html.

[10] Topic 03: Tinyos and telos. http://www.cse.iitk.ac.in/users/braman/.

[11] I. Aad and C. Castelluccia. Differentiation mechanisms for ieee 802.11, 2001.

[12] M. Adamou, S. Khanna, I. Lee, I. Shin, and S. Zhou. Fair real-time traffic scheduling over a wireless lan. In *In RTSS 01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS01*, page 279. IEEE Computer Society, 2001.

[13] K. Akkaya, M. Younis, and M. Youssef. Efficient aggregation of delay-constrained data in wireless sensor networks. In *AICCSA '05: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications*, pages 904–909, Washington, DC, USA, 2005. IEEE Computer Society.

[14] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[15] G. A. Awater and F. C. Schoute. Performance improvement of fast packet switching by ldoll queueing. In *IEEE INFOCOM '92: Proceedings of the eleventh annual joint conference of the IEEE computer and communications societies on One world through communications (Vol. 2)*, pages 562–568, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[16] J. J. Bae and T. Suda. Survey of traffic control schemes and protocols in atm networks. In *Proceedings of the em IEEE*, volume 79, pages 170–189, 1991.

[17] H. Baldus, K. Klabunde, and G. Müsch. Reliable set-up of medical body-sensor networks. In *EWSN*, pages 353–363, 2004.

[18] P. Bose, P. Morin, I. Stojmenovi, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Wireless Networks*, pages 48–55, 2001.

[19] G. Buttazzo and J. A. Stankovic. Adding robustness in dynamic preemptive scheduling. In *Responsive Computer Systems: Steps Toward Fault-Tolerant Real-Time Systems*. Kluwer Academic Publishers, 1995.

[20] R. Chipalkatti, J. F. Kurose, and D. Towsley. Scheduling policies for real-time and nonreal-time traffic in a statistical multiplexer. In *In Proceeding. IEEE INFOCOM '89*, pages 774–783, 1989.

[21] E. Corrige and G. Buttazzo. Hard real-time computing systems: Predictable scheduling algorithms and applications, 2001.

[22] G. G. Finn. Routing and addressing problems in large metropolitan-scale internetworks. In *Technical Report ISI/RR-87-180*. University of Southern California, 1987.

[23] C. F. Garcła-hernndez, P. H. Ibargengoytia-gonzlez, J. Garcła-hernndez, and J. A. Prez-dłaz. Wireless sensor networks and applications: a survey, 2007.

[24] C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 129–143, New York, NY, USA, 2004. ACM.

[25] J. L. Hill. *System Architecture for Wireless Sensor Networks*. PhD thesis, University of California, Berkley, Spring 2003.

[26] R. Jain, A. Durresi, and G. Babic. Throughput fairness index: An explanation. In *ATM Forum/99-0045*, 1999.

[27] J.Y.T.Leung and J. Whitehead. Scheduling algorithms for multiprogramming in a hard-real-time environment, 1973.

[28] B. Karp. Challenges in geographic routing: Sparse networks, obstacles, and traffic provisioning. In *the DIMACS Workshop on Pervasive Networking*, 2001.

[29] K. Kubota, M.Murata, H.Miyahara, and Y. Oie. Congestion control for burstry video traffic in atm networks. In *Electronics and Communications in Japan*, volume 75, pages 13–19, 1992.

[30] H. Li, P. Shenoy, and K. Ramamritham. Scheduling communication in real-time sensor applications. *Real-Time and Embedded Technology and Applications Symposium, IEEE*, 0:10, 2004.

[31] C. Liu and J. Kaiser. A survey and mobile ad and hoc routing protocols, 2003.

[32] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment, 1973.

[33] K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. In *IEEE Internet Computing*, pages 18–25, 2006.

[34] C. Lu, B. M. Blum, T. F. Abdelzaher, J. A. Stankovic, and T. He. Rap: A real-time communication architecture for large-scale wireless sensor networks. 2002.

[35] A. Mainwaring, J. Polastre, R. Szewczyk, and D. Culler. Wireless sensor networks for habitat monitoring. pages 88–97, 2002.

[36] S. Mangold, S. Choi, P. May, O. Klein, G. Hiertz, L. Stibor, C. poll Contention, and F. Poll. Ieee 802.11e wireless lan for quality of service, 2002.

[37] K. Martinez and R. Ong. Glacsweb: a sensor network for hostile environments. In *IEEE SECON*, pages 81–87, 2004.

[38] C. Mbarushimana and A. Shahrabi. Comparative study of reactive and proactive routing protocols performance in mobile ad hoc networks. *Advanced Information Networking and Applications Workshops, International Conference on*, 2:679–684, 2007.

[39] W. M. Meriall, F. Newberg, K. Sohrabi, W. Kaiser, and G. Pottie. Collaborative networking requirements for unattended ground sensor systems. In *In Proc. IEEE Aerospace Conference*, 2003.

[40] H. Nagabuchi, A.Takahashi, and N. Kitawaki. Speech quality degraded by cell loss in atm networks. In *NTT Review*, 1992.

[41] B. O'Hara and A. Petrick. *802.11 Handbook - a Designer's Companion.* IEEE press, 1999.

[42] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *In Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1997.

[43] F. J. Pierce and T. V. Elliott. Regional and on-farm wireless sensor networks for agricultural systems in eastern washington. *Comput. Electron. Agric.*, 61(1):32–43, 2008.

[44] K. Ramamritham and J. A. Stankovic. Dynamic task scheduling in hard real-time distributed systems. *IEEE Softw.*, 1(3):65–75, 1984.

[45] P. Samundiswary, P. Priyadarshini, and P. Dananjayan. Performance evaluation of heterogeneous sensor networks. *Future Computer and Communication, International Conference on*, 0:264–267, 2009.

[46] S.Kawaguchi, M.Tsujikado, Y.Ueda, and K. Hosoda. Quality controlled variable rate coding based on constant error criterion. In *In Proceedings of Visicom '90: Third International Workshop on Packet Video*, 1990.

[47] G. Sklyarenko. Aodv routing protocol. Technical report, Institute of Informatics, Freie Universy.

[48] J. A. Stankovic, C. Lu, L. Sha, T. Abdelzaher, and J. Hou. Real-time communication and coordination in embedded sensor networks. In *Proceedings of the IEEE*, pages 1002–1022, 2003.

[49] J. Tavares, F. J. Velez, and J. M. Ferro. Application of wireless sensor networks to automobiles. *Measurement Science Review*, 2008.

[50] D. Tian and N. D. Georganas. Location and calculation-free node-scheduling schemes in large wireless sensor networks. In *Ad Hoc Networks*, pages 65–85, 2004.

[51] N. S. M. Usop, A. Abdullah, and A. F. A. Abidin. Performance evaluation of aodv. dsdv & dsr routing protocol in grid environment. In *IJCSNS International Journal of Computer Science and Network Security*, 2009.

[52] K. Varadhan and K. Fall. The ns manual, 2009.

[53] C. Wang, B. Li, K. Sohraby, M. Daneshmand, and Y. Hu. Upstream congestion control in wireless sensor networks through cross-layer optimization. *IEEE Journal on Selected Areas in Communications*, 25(4):786–795, 2007.

[54] W. Ye and J. Heidemann. Medium access control in wireless sensor networks. In *USC/ISI Technical Report ISI-TR-580*, 2003.

[55] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Comput. Netw.*, 52(12):2292–2330, 2008.