8-2007

Topology-aware transmission scheduling for distributed highway traffic monitoring wireless sensor networks

Devang Bagaria
Clemson University, dbagari@clemson.edu

TOPOLOGY-AWARE TRANSMISSION SCHEDULING FOR DISTRIBUTED HIGHWAY TRAFFIC MONITORING WIRELESS SENSOR NETWORKS

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science
Computer Engineering

by
Devang Bagaria
August 2007

Accepted by:
Dr. Kuang-Ching Wang
Dr. Harlan Russell
Dr. Ronnie Chowdhury

ABSTRACT

Wireless sensor networks have been deployed along highways for traffic monitoring. The thesis studies a set of transmission scheduling methods for optimizing network throughput, message transfer delay, and energy efficiency. Today's traffic monitoring systems are centrally managed. Several studies have envisioned the advantages of distributed traffic management techniques. The thesis is based on previously proposed hierarchical sensor network architecture, for which the routing and transmission scheduling methods are derived.

Wireless sensor networks have a lifetime limited by battery energy of the sensors. The thesis proposes to assign schedules for nodes to transmit and receive packets and turning off their radios during other times to save energy. The schedules are assigned to minimize the end-to-end packet delivery latency and maximize the network throughput. Conflict-free transmission slots are assigned to sensors along road segments leading to a common intersection based on locally discovered topology. The slot assignment adopts a heuristic that rotates among segments, assigns closest possible slots to neighboring nodes in a pipelined fashion, and exploits radio capture effects when possible. Based on the single-intersection approach, centralized and distributed multi-intersection scheduling methods are proposed to resolve conflicts among nodes belonging to different intersections. The centralized approach designates a controller as the *leader* to collect topology information of a set of contiguous intersections and assign schedules using the same single-intersection algorithm. The distributed approach has each intersection determine its own schedule independently and then exchange the topology information and schedules with its adjacent intersections to resolve conflicts locally. Based on simulation studies in ns-2, the centralized approach achieves better performance, while the distributed approach tries to approach the centralized performance at much lower communication costs. A communication cost analysis is performed to assess the trade-off between the centralized and distributed approaches.

DEDICATION

This thesis is dedicated to my beloved family – my mother, my father, and my friends. Their support has helped me climb the steps of success in every phase of my life.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Kuang-Ching Wang, for all his guidance and motivation. During all my time of study and research at Clemson University, he gave me valuable support and advices and always helped direct my thoughts in the right direction. I would like to thank him for all the valuable knowledge and experience he has shared with me. I appreciate the time we have spent together working on the research issues and driving it towards the direction.

I would also like to thank Dr. Ronnie A. Chowdhury for all the knowledge that he has shared with us during our ITS meetings and for all the support and motivation he gave during us during the project. I would also like to thank all the people in the ITS group.

I would like to thank all the colleagues in the wireless group for their friendship and help. I have had a great time conducting my research within such an active group. I have enjoyed all the time spent with my colleagues in the lab and office.

TABLE OF CONTENTS

Page

CHAPTER

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Figure Page

CHAPTER 1

INTRODUCTION

Countries around the globe are adopting Intelligent Transportation Systems (ITSs) for continuous monitoring and real-time control of vehicle traffic on highways and major roads [1], [2], [3]. Sensors are laid down along the roads to collect the data and route the data to the traffic management centers (TMCs) that take the appropriate actions. The data is sent to the TMC via various types of communication links for analysis by traffic engineers for highway traffic management. Communication links between sensors and TMCs are hence the lifeline of such systems, determining the sustainable scope and response efficacy of the system. State-of-the-art ITSs largely depend on wired communication technologies, while advancements in wireless sensor network technologies have brought opportunities in building wireless traffic sensor networks permitting easier deployment, more variety in operation, and more extensive coverage. In this thesis a distributed wireless sensor network solution for highway traffic monitoring is studied.

A few ITS systems have already started to utilize wireless data links recently [4], [5]. One fundamental limitation to an ITS system's scale is due to its *centralized* data collection model, where sensors must feed data to TMCs via wired, wireless, or a combination of both types of links. Leveraging the processing capability of advanced wireless sensor devices, it is possible to construct a *distributed* system, where sensors locally collaborate in processing traffic data and initiating appropriate control responses, while reporting to TMCs only data of known significance. The vision has been posed in

[6] where the importance of a distributed wireless sensor network is explained, while in [7] a distributed sensor system based on hierarchical network architecture was proposed, for implementing traffic control operations in a distributed manner. The hierarchical architecture considers sensors as lowest level devices organized in clusters, each of which has a designated cluster head called controller. A set of controllers along with the clusters controlled by it form a higher level cluster. In this system, sensors and controllers coordinate to carry out distributed traffic sensing and control operations. This thesis builds on this concept to study the transmission scheduling problem for such systems. A practical routing method pertaining to the architecture is also proposed in this thesis to facilitate the study.

Wireless sensor devices are envisioned by many to be battery-powered. To conserve energy, numerous studies have been made by means of energy-efficient medium access control methods [8-16] that assign sleep-awake cycles to sensor nodes. Turning off the sensor radios during sleep periods provides energy savings but leads to degradation of transmission delay and throughput. For traffic control operations requiring high throughput and stringent delay requirements, the thesis proposes a transmission scheduling solution that simultaneously improves energy efficiency, network throughput and packet delivery latency.

Existing energy-efficient MAC layer protocols address certain specific issues. SMAC [8] was the first to study energy savings for sensors by scheduling them to receive and transmit in pre-decided slots. TMAC [9] discusses turning off the radio for scheduled slots that do not have any data to be transmitted and thus provides for additional savings. DMAC [10] proposes to reduce the end-to-end delay for wireless sensor networks. It proposed to avoid sleep latency by assigning schedules such that the receiver in slot *n* becomes the sender in slot *n+1*. In [17], hierarchical PEGASIS allows spatially separated nodes to simultaneously transmit in the same slot and increase the network throughput. In this thesis, the aforementioned concepts were incorporated to derive a topology-aware scheduling solution tailored for a large scale highway traffic sensor network.

The proposed scheduling method is based on the following concepts:

- **Intersection based topology discovery and slot assignment.** A scalable distributed system must discover and adapt to local changes in network topology. The algorithm considers a network to be composed of numerous road segments connected by controllers at intersections. A controller collects the topology information of all sensors belonging to its cluster and executes the scheduling algorithm to assign them non-conflicting transmission schedules.

- **Closest-possible, directional, and pipelined slot assignment for adjacent nodes.** To minimize delay of forwarding packets along the road in a particular direction, the adjacent node towards that direction is assigned the closest non-conflicting time slot according to the current schedule and the conflict graph. To maximize throughput, time slots are assigned in a pipelined fashion such that, along each road segment, subsequent messages are scheduled at the earliest time following the last message's departure. The pipelining allows spatially separated nodes along the same segment to transmit in the same slot thereby providing higher throughput.

- **Rotated assignment among road segments of a common intersection.** Assuming messages arriving from any of the connected road segments are of equal probability and priority, one node on every segment is scheduled in a round robin manner until the specified schedule length is met. This assures fair message throughput along each segment. However, weighted assignment can be used to divide the throughput between the segments based on their communication requirements.

- **Topology propagation and schedule negotiation among multiple intersections.** Each intersection controller determines the schedule for a set of nodes without considering potential conflicts with nodes managed by other intersection controllers. Multiple intersections need to have conflict free schedules which can be achieved in a centralized or distributed manner. In centralized assignment a *leader* collects the topology of all intersections and constructs a complete conflict graph that is used to assign conflict free schedules. In the distributed assignment the intersections negotiate their schedules to achieve conflict free schedules.

The remaining of the thesis is organized as follows. In chapter 2 the related work and achievements are described. In chapter 3 the hierarchical traffic sensor network architecture and the hierarchical packet routing are described. In chapter 4 the single intersection transmission scheduling algorithm and its variations are described and the chapter ends with the simulation studies showing the effectiveness of the algorithm. In chapter 5 the multiple intersection transmission scheduling algorithms are described and the simulation studies show the effectiveness of the transmission scheduling algorithms. In chapter 6 the conclusion and future work are presented.

CHAPTER 2

BACKGROUND AND RELATED WORK

Wireless sensor networks are composed of large number of sensors that are battery powered and the lifetime of the network depends on the life of these sensors. The sensors follow a sleep-awake schedule to help them save energy in the no activity period. The thesis studies a scheduling solution for highway traffic monitoring wireless sensor networks that can improve different data transmission metrics. In this chapter, the current traffic control practice, the distributed sensor networking architecture, and several energy-efficient sensor MAC protocols are reviewed.

2.1 Current Traffic Control Practices

The 2004 U.S. ITS survey conducted by the U.S. Department of Transportation reveals the nation's latest traffic control operations and deployed infrastructure [18] portraying similar practices in other parts of the world. These systems adopt a centralized architecture with TMCs at the core and sensors and remote control devices on the roads. The functions of the different elements of this system are as follows:

- **Sensors** acquire real-time traffic data of monitored road segments. Widely adopted are loop detectors, closed circuit television (CCTV) cameras, video detection systems (VDS), infra-red laser, acoustic, microwave radar, and piezo-electric strain gauge sensors. Typical metrics acquired are traffic speed, volume, road occupancy, and road conditions (pavement integrity, ice on pavements, fog, inclement weather conditions such as snow, rain, and tornadoes).

- **TMCs** collect, archive, and analyze the sensor data. TMC operators detect traffic incidents, dispatch incident removal teams, and publish current traffic information to the public via Internet, highway advisory radio, TV, operator hotlines (511), in-car navigation systems, and call-outs to subscribed travelers.
- **Offsite control devices** are controlled by TMCs to regulate traffic flow. Mostly adopted are Dynamic Message Signs (DMS), ramp meters, traffic signals, etc.

Traffic management systems have to perform several traffic management operations for proper functioning of the highway traffic:

- **Traffic regulation** – Controlling or reprogramming offsite control devices based on current and historical traffic data to regulate traffic flows against congestion.
- **Traveler information** – Providing travelers with current traffic condition, such as travel time estimates, driving advices, suggesting alternative routes, etc.
- **Incident prevention** – Disseminating real time alerts, such as dangerous road conditions, misbehaving vehicles, and interfering entities, to prevent incidents.
- **Incident management** – Expanding surveillance coverage for quicker incident detection and removal.

Traffic control strategies differ in urban and rural areas. They also differ for freeways and local roads [18]. In urban areas, traffic features high volumes with larger and faster variations. Typically more resourceful, urban systems can afford more sensors, wider data bandwidth, and more control devices for fine-grained traffic control and guidance [19]. In contrast, rural roads stretch long distances with few cross roads or alternative roads. Traffic is sparser, faster, and seldom interrupted, while incidents are often severe and mostly due to unexpected interruption (e.g., animal crossing). Rural roads typically cannot afford centrally controlled sensing and control devices. Distributed system can involve local processing of data and execution of traffic management operations thereby reducing the communication overhead.

Freeway Performance Measurement System (PeMS) in California [4], is a large *centralized* sensor systems in U.S. that links more than 25,000 loop detectors to the TMC. The system collects data from each detector every 30 seconds, accumulating more than 2 GB data per day in a central database. Sensors can be connected with different mediums like fiber cables, GPRS wireless cellular links [5], CDPD cellular wireless links [20]. Recently, the PEDAMACS project [21] unties the sensors from TMCs; instead, sensors in each cluster send data to a wireless gateway, which is then directly wired with the TMC. Multi-hop forwarding is used by sensors to send data to the gateway but not beyond the cluster. Data collection remains centralized and rooted at the TMC. An industrial vision of a fully distributed system is portrayed in [22] without further technical insights. A distributed system can scale over large area and based on the requirements the operations can be carried out locally. In [7] a hierarchical sensor network architecture is proposed that can support the formation of a distributed system. The sensors form clusters with controllers and the operations that need communication between sensors of a cluster can be carried out locally. Many clusters combine to form higher level clusters that can carry out operations needing communication limited within the scope of higher level cluster. Thus the hierarchical network architecture helps form a distributed system that can carry out operations more effectively without needing the communication overhead to communicate with the TMC.

2.2 Distributed Operations and Hierarchical Architecture

The distributed operations can be supported by the hierarchical architecture using hierarchical addressing and routing. The following explains the choice of hierarchical addressing and illustrates examples of distributed operations.

2.2.1 Architectures and Addressing schemes

It is commonly agreed that sensor addressing and routing is application specific and the key is to intimately reflect the sensor data contexts. Sensor addressing can be considered in four contexts: geographic, ad hoc, hierarchical, and data addressing/routing. Geographic addresses are suitable for sensors deployed in unstructured terrains with geographic locations being their only identity [23][24][25]. Ad hoc addresses are context-less identifiers (ID), useful when individual sensors are insignificant and communications are predominantly one way towards the data sinks [26]. Data addresses are not device specific identifiers but data type descriptors, useful when only data types are of interest, e.g., [27]. Hierarchical addresses are useful when the application features an inherent hierarchy. It is noticed that if the network is structured according to the application and has a large scope then hierarchical addressing is a good choice.

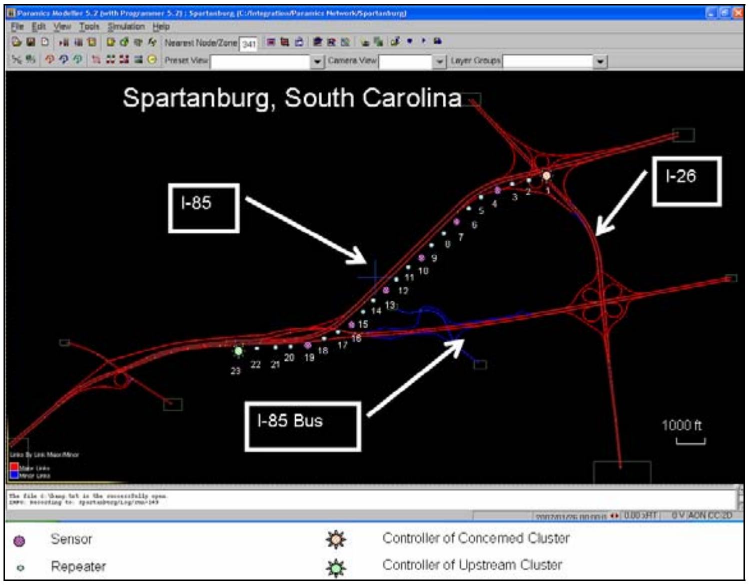
The traffic sensor network being studied certainly operates in a hierarchy, where sensors interact with local controllers, who then interact with regional controllers and can extend several levels until reaching the TMC. The communications, however, are not rooted at a single node but require complex interactions across different levels of the hierarchy. The scope of the network is very large and such a network should be able to scale to long highways spread over large geographical areas. Because of the nature of the network being structured and the large scope of the network it is suitable to use hierarchical addressing. Hierarchical architecture and addressing thereby have advantages for highway traffic monitoring wireless sensor networks.

In [7] the authors have defined a hierarchical architecture composed of sensors, controllers, and other higher level controllers for highway traffic monitoring wireless sensor networks. The idea of cluster formation for sensor networks was first proposed by LEACH [28] as it helps for efficient energy management whereby nodes with high energy become the cluster heads. In the proposed architecture [7] the controllers (cluster heads) are assumed to have more energy as compared to sensors. Initially it was proposed by LEACH to have direct transmissions from every sensor to the cluster head. This involved higher energy consumption as the transmission power required is directly proportional to the square (or higher exponent) of the distance and square of the sum is always higher than sum of the squares. It was thereby proposed by PEGASIS [29] to transmit the data from a sensor to the cluster head using multi-hop transmissions to save more energy. In this way the proposed hierarchical network has sensors and controllers forming clusters where data is forwarded over multiple hops to reach the controller.

2.2.2 Example of Distributed Operation

A simple traffic control operation for incident detection is explained in the following section to illustrate the distributed communication patterns in such a system. An example of a wireless sensor network for highway traffic monitoring on a segment of I-85 near Spartanburg, South Carolina is shown in Fig. 2.1. The sensors monitor the data at a time interval, delta [Δ]. There are traffic algorithms running at the application layer of the sensors that can help the sensor determine the event of an incident on the monitored segment. If the sensor collects data that determines the occurrence of an incident, then it sends a message to the neighboring sensors where the accuracy of the incident detection is checked. If the neighboring sensor confirms the incident then it reports the data to the controller. The controller can send a message to the upstream controller asking it to divert the cars on another segment thereby avoiding the cars queuing up on the segment where the incident had occurred. It is seen from the above example that the sensors and controllers could communicate locally and make the decision that can help divert the traffic without needing to communicate to the TMC.

Another application is travel time updates. The sensors along the segment periodically report the vehicular speed recorded by them to the controller. The controllers periodically report the average speeds along the segments to higher level controllers that are controlling the high level clusters. The higher level controllers know the average


Paramics Modeller 5.2 (with Programmer 5.2) : Spartanburg (C:/Integration/Paramics Network/Spartanburg)
File Edit View Tools Simulation Help
Nearest Node/Zone 341
Preset View
Camera View
Layer Groups
Spartanburg, South Carolina
I-85
I-26
I-85 Bus
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
1000 ft
Links By Link Major/Minor
Major Links
Minor Links
0.00 xRT
0 V AON CC:2D

The file c:\temp.txt is the successfully open.
INFO: recording to: spartanburg/Log/run-149

● Sensor
○ Repeater
✹ Controller of Concerned Cluster
✹ Controller of Upstream Cluster

Figure 2.1: Traffic monitoring wireless sensor network

speed on all the segments in its cluster and thereby can calculate the travel time for traveling from one segment to any other segment in its cluster. Thus the average speed from one point to another can be calculated at the high level controller and then distributed to any other controller without any communication with the TMC. In this way a hierarchical architecture facilitates distributed control operations.

2.3 Sensor MAC Concepts

The distributed wireless sensor network system is made up of battery powered sensors and thus the lifetime of the network depends on the life of the sensors. The four major sources of energy wastage are collision, overhearing, control packet overhead and idle listening. The energy wasted in idle listening is 50-100% of that spent in receiving [30] [31]. There have been MAC layer protocols that propose to increase the lifetime of the network by turning off the radio in period of inactivity but this increases the end-to-end delay and decreases the end-to-end available throughput. The delay incurred at every node is the sum of back-off delay, processing delay, propagation delay, transmission delay, queuing delay and *sleep latency* which is the defined as the delay incurred when the sender has to wait till the receiver wakes up before transmitting the packet. The end-to-end delay consists of the sum of the delay incurred at every forwarding node.

SMAC (sensor-Mac) [8] was the first to propose the sleeping mechanism for sensors to save energy. A node remains awake for sometime and sleeps for the remaining time instead of idle listening thereby saving energy. The amount of energy saved will be inversely proportional to the *duty cycle* which is defined as the ratio of listen interval (active time) to the frame length (active time + sleep time). It is seen that with 10% duty cycle they achieve very high energy efficiency at the cost of high sleep latency. SMAC proposed an adaptive listening mechanism to reduce the sleep latency whereby a node that overhears its neighbor's transmission wakes up for a short period of time at the end of the transmission, so that if it is the next hop of its neighbor, it can receive the message without waiting for its scheduled active time.

TMAC (Timeout MAC) [9] is another algorithm proposed for energy efficient MAC layer protocol that provides higher energy efficiency than SMAC. Like SMAC, TMAC also decides schedules for the different nodes and thereby allows the nodes to sleep and save energy. TMAC addresses that the traffic rate can vary from time to time and hence it is difficult to predict the optimal duty cycle. TMAC provides a mechanism of having dynamic active periods sufficient for relaying the instantaneous traffic load. With TMAC, a node in its active period monitors the medium for a time interval; if there is no activity seen in a timeout interval *TA*, the node assumes that there is no data to be received and it goes back to sleep. Thus TMAC gives additional energy savings as compared to SMAC.

The DMAC [10] protocol provides an energy efficient solution to reduce the end-to-end delay. For DMAC it is assumed that there is a fixed topology (a tree formation) and the data always flows from far away nodes to the sink. Because of this assumption, the slots should be assigned to nodes in such a way that packets flow continuously without incurring any sleep latency. It achieves this by assigning the receiver in slot *n* to be the transmitter in slot *n+1* thereby removing any sleep latency. DSMAC (Dynamic SMAC) [11] is a modified version of SMAC where a node doubles its duty cycle when the latency of packets received crosses a certain threshold. The neighboring nodes get this information in a sync packet and they double their duty cycle. Thus DSMAC tries changing the duty cycle based on the observed delay.

For PMAC (Pattern MAC) [12] every node calculates the load that is present in its queue and depending on the load it sends out a pattern which is exchanged with the nearby nodes to achieve a common schedule. This schedule is followed by every node for transmitting data. The main advantage of PMAC is that the slots are assigned dynamically based on the traffic load at the nodes. For WiseMAC [13] every node transmits a preamble before sending out a data packet. The nodes send out a preamble during the sampling time of the receiver. By doing this the receiver will read the information from the preamble and will stay awake till the data packet transmission is completed. The main advantage of WiseMAC is that the preamble contains all the information about which nodes have to be awake and for what time and the receiver has to be awake to only sample the medium for a small interval of time. The traffic-adaptive MAC protocol (TRAMA) [14] divides time into slots that have random access periods and then scheduled access periods. One node is elected as transmitter in the random access period and it tells the surrounding nodes the duration of its transmission and the intended receivers, thereby allowing the other nodes to sleep.

The distributed energy aware MAC protocol (DE-MAC) [15] increases the network lifetime by trying to save energy for the node that has the least energy among all the nodes in the network. When the energy at a node falls below a certain threshold then the node starts an election process and after the election process the node with the least energy is elected to sleep for a longer time thereby saving more energy and increasing network lifetime. ZMAC [16] is a hybrid MAC protocol that incorporates the advantages of both CSMA and TDMA mechanisms by making the nodes function in CSMA mode during low traffic periods and switching them to TDMA mode during high traffic periods.

It can be seen from the previous discussion that there has been a lot of research for providing energy efficiency for sensor networks. Some protocols, like SMAC and TMAC address energy efficiency but not packet delivery latency. Protocols like PMAC, WiseMAC and TRAMA address procedures to adjust to dynamic loads but not sleep latency. DMAC and DSMAC address sleep latency but have not considered the overall network throughput. Thus there has not been a complete study that considers all the different metrics for performance. To increase throughput, reduce packet delivery latency, and increase energy efficiency the proposed solution integrates concepts from DMAC and TMAC in its topology aware scheduling solution.

CHAPTER 3

NETWORK ARCHITECTURE AND ROUTING

The proposed wireless sensor network system is based on a hierarchical architecture and hierarchical addressing scheme proposed in [7] and a description of the same is given below.

3.1 Network Architecture and Addressing

In [7] the authors have defined a hierarchical architecture composed of sensors, controllers, and other higher level controllers. A set of sensors together with one controller (cluster head) form a *cluster* that is controlled and managed by the controller. As shown in Fig. 3.1, sensors on the same side of a road form a *segment* and all segments in the direction leading to the controller form a cluster controlled by that controller. A controller is connected with multiple segments and the segments that form clusters controlled by that controller are called the *associated segments* of that controller. The connected segments that form clusters controlled by the controller on the other end of the segment are known as the *neighbor-associated segments* of the controller. Fig. 3.1 shows an example of associated and neighbor-associated segments for controller C2. Each sensor or controller device can serve one or multiple logical functions according to the one or multiple logical identities assigned to the device. In the hierarchical architecture, multiple sensors form a cluster with a controller and multiple clusters form a higher level cluster with a higher level controller controlling it.

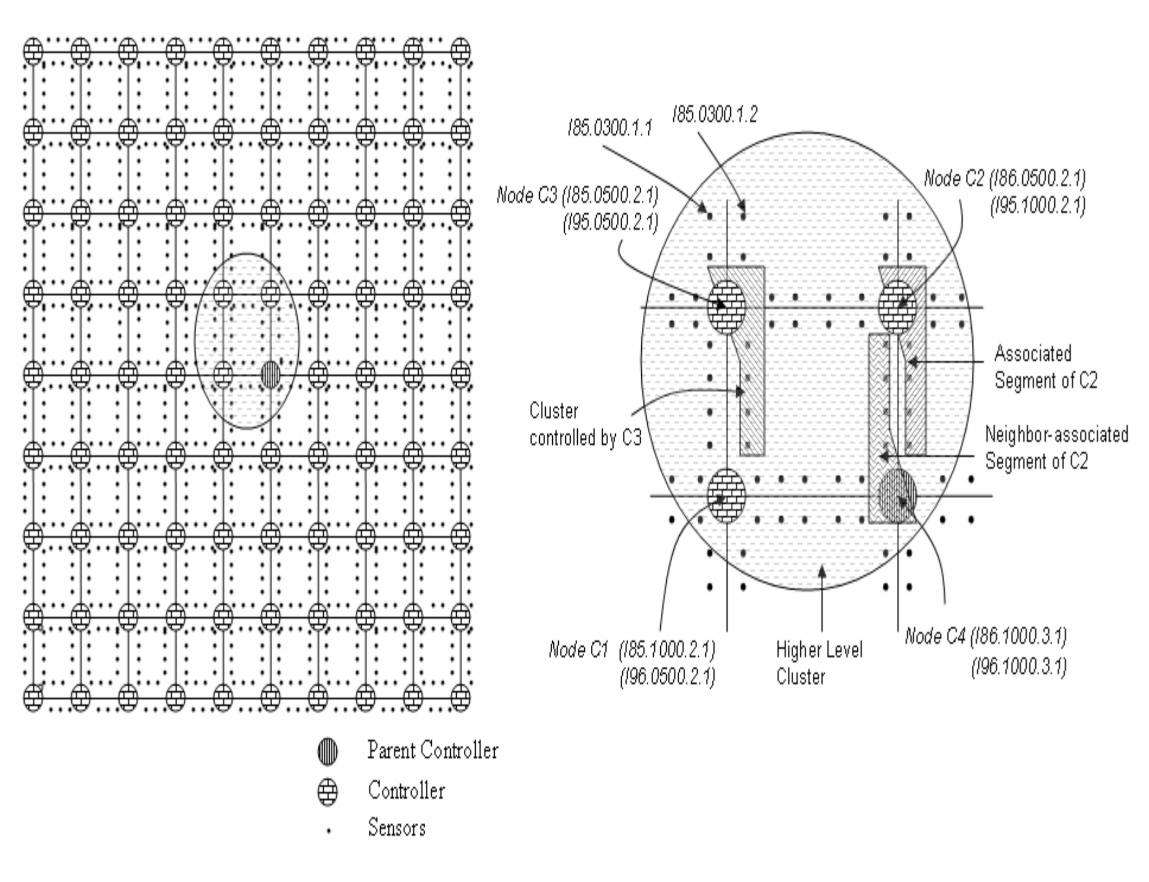


Figure 3.1: Network architecture

The following address format is adopted to identify any sensor or controller [7]:

[RID; Milepost; Level; Direction]

A brief description of each filed in the address is given.

- RID – A 3 character field representing the highway ID on which the sensor is located. *For example:* I85
- Milepost – A 4 character field indicating the mileage on the highway where the sensor is located. *For example:* 0127
- Level – A 1 character field indicating the level at which the node is functioning. *For example:* 1-Sensor, 2-Level 1 Controller, 3-Level 2 Controller

- Direction – A 1 character field indicating the side of the road the node is located.

  For example: 1- North bound (North-South Highway) and East bound (East-West Highway), 2- South bound (North-South Highway) and West bound (East-West Highway).

While simple and intuitive, the address fully exposes the contextual information necessary for message routing and handling of distributed control operations. Note that each unit can have multiple addresses if it serves multiple roles.

3.2 Routing

For routing data between sensors the authors of [7] proposed to have a hierarchical routing protocol. A detailed routing procedure, however, was not available. In this thesis a complete routing procedure is explained that explains cluster formation and data routing in such a hierarchical network. The routing procedure can be divided into three phases – local topology discovery, hierarchical cluster formation and data routing. The detailed procedures are explained in the Appendix.

The local topology discovery is performed to establish local connectivity and form local clusters in the network. During the local topology discovery phase, every sensor identifies the adjacent sensor on its road segment towards its cluster head as its *parent.* A sensor *X* becomes a *child* of another sensor *Y,* if Y is the parent of X. Hierarchical cluster formation is performed to form higher level clusters controlled by higher level controllers. During this phase every controller registers to a cluster controlled by higher level controllers. The higher level controller thereby becomes the parent of the low level controller that has registered to its cluster. After the local topology discovery and hierarchical cluster formation phases have completed, packets can be routed from one part of the network to another using the hierarchical routing protocol. For routing messages the controllers maintain routing tables storing the next controller information to reach any other part node in the network. It is important to minimize the packet delivery latency for sending messages from controller to its adjacent controllers. A scheduling solution is thereby provided in the following chapter that tries to minimize delay for sending messages from controller to controller forwarded by sensors along the connecting road segment.

3.3 Medium Access Control

The proposed scheduling solution operates over a carrier-sense multiple-access link-layer protocol. Nodes are assumed capable of coarse grain synchronization, with which the scheduling algorithm assigns collision-free time slots for nodes to transmit data packets. Specifically, IEEE 802.11 is considered without its Request-To-Send/Clear-To-Send (RTS/CTS) option. Each data packet is transmitted directly (after an initial idle period) from a sender to the receiver, and if the packet is received correctly, the receiver transmits an acknowledgement to the sender.

The carrier-sense multiple-access with collision-avoidance (CSMA/CA) scheme of IEEE 802.11 requires a sender to sense the channel to be idle before sending a data packet. The duration a sender must sense the channel to be idle is one distributed inter frame spacing (DIFS) duration plus initial random backoff duration. The sensing timer is paused whenever the channel is not idle. A receiver on receiving the packet correctly waits for one short inter frame spacing (SIFS) duration before sending its acknowledgement. Upon receiving the acknowledgement the sender has to wait for the DIFS and random backoff duration again before transmitting the next packet. IEEE 802.11 adopts an exponential random backoff algorithm [33]. The random backoff time is incurred to resolve contention when two stations want to access the medium at the same time. Each station randomly selects the number of slots to wait between 0 and its contention window variable. The contention window starts with a minimum default value, doubles if a transmission fails, and resets to minimum after each successful transmission.

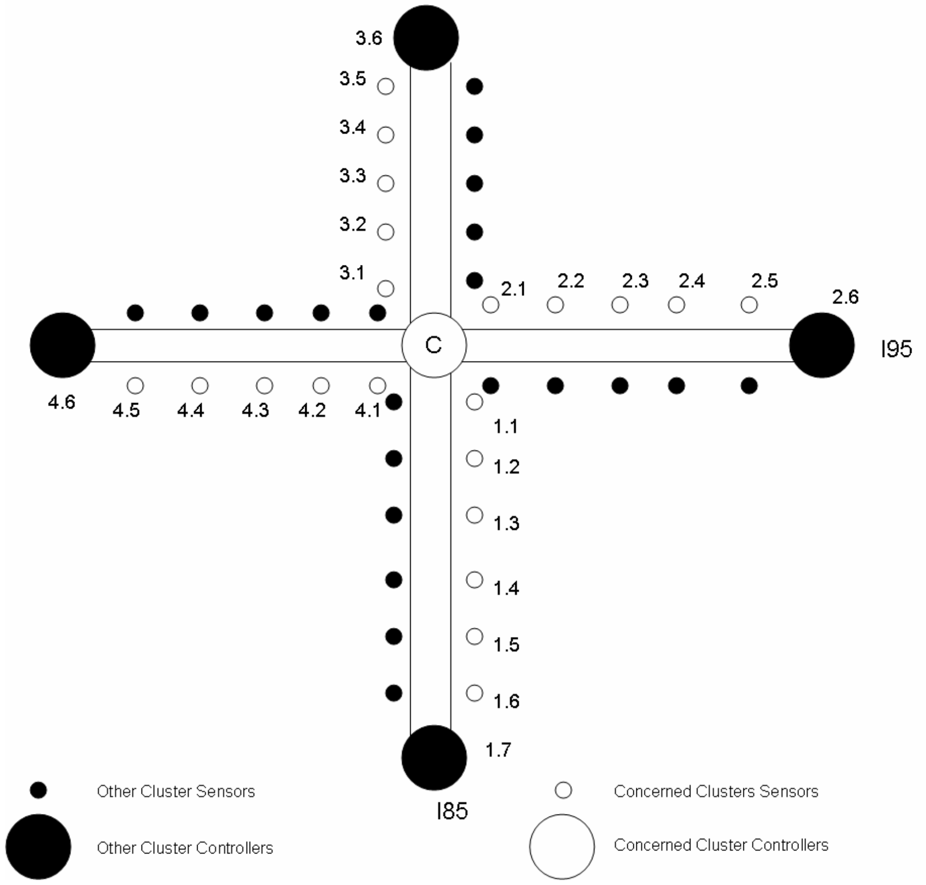
We adopt a model for IEEE 802.11 that accounts for the capture effect [34]. The *capture effect* states that, if a conflicting signal arriving during an ongoing transmission is perceived at a power less than $1/\kappa$ of the ongoing transmission power then the ongoing data reception will proceed without error; if the conflicting transmission power is larger than $1/\kappa$ the ongoing transmission's power, both packets will be corrupted. Here, κ is denoted as the capture ratio and the typical values are $0 < \kappa < 10$. In our model we consider the capture ratio κ is 10db [35]. Routing protocol assumes the nodes to be capable of adapting the transmission power to reach nodes that are multiple hops away. However, we consider the base case of tuning the power to reach the adjacent neighbors.

CHAPTER 4

SINGLE INTERSECTION TRANSMISSION SCHEDULING

The scheduling algorithms proposed aims at increasing throughput, decreasing end-to-end delay and increasing energy efficiency for the proposed wireless sensor network. The scheduling solution incurs a topology discovery phase, followed by the slot assignment for individual intersections and across multiple intersections. The following describes the single-intersection scheduling algorithm.

A *single intersection* denotes the collection of an intersection controller and all sensors along its connected road segments controlled by it, as shown in Fig. 4.1. The intersection controller shown in Fig. 4.1 is associated with four segments. The nodes in the figure are denoted as *segment.member,* where *segment* is the segment number and *member* is the member number for that particular segment. After the topology discovery phase, the controller knows the hierarchical and geographical addresses and the transmit power of the nodes on its associated segments and constructs a conflict graph of these nodes. In the conflict graph, a pair of nodes are connected with an edge if they can interfere with each other when both are active (transmitting or receiving) and at least one of them is transmitting. Fig. 4.2 shows the conflict graph and the associated matrix representation for the intersection is shown in Fig. 4.3. Given the conflict graph, the controller can calculate non-conflicting transmission schedules for all nodes associated with the intersection.



Node	Address
C	I85.0650.2.1 I95.0550.2.1
1.1	I85.0600.1.2
1.2	I85.0500.1.2
1.3	I85.0400.1.2
1.4	I85.0300.1.2
1.5	I85.0200.1.2
1.6	I85.0100.1.2
1.7	I85.0000.1.2
2.1	I95.0600.1.1
2.2	I95.0700.1.1
2.3	I95.0800.1.1
2.4	I95.0900.1.1
2.5	I95.1000.1.1
2.6	I95.1000.1.1
3.1	I85.0700.1.1
3.2	I85.0800.1.1
3.3	I85.0900.1.1
3.4	I85.1000.1.1
3.5	I85.1100.1.1
3.6	I85.1200.1.1
4.1	I95.0500.1.2
4.2	I95.0400.1.2
4.3	I95.0300.1.2
4.4	I95.0200.1.2
4.5	I95.0100.1.2
4.6	I95.0000.1.2

Figure 4.1: Single intersection topology

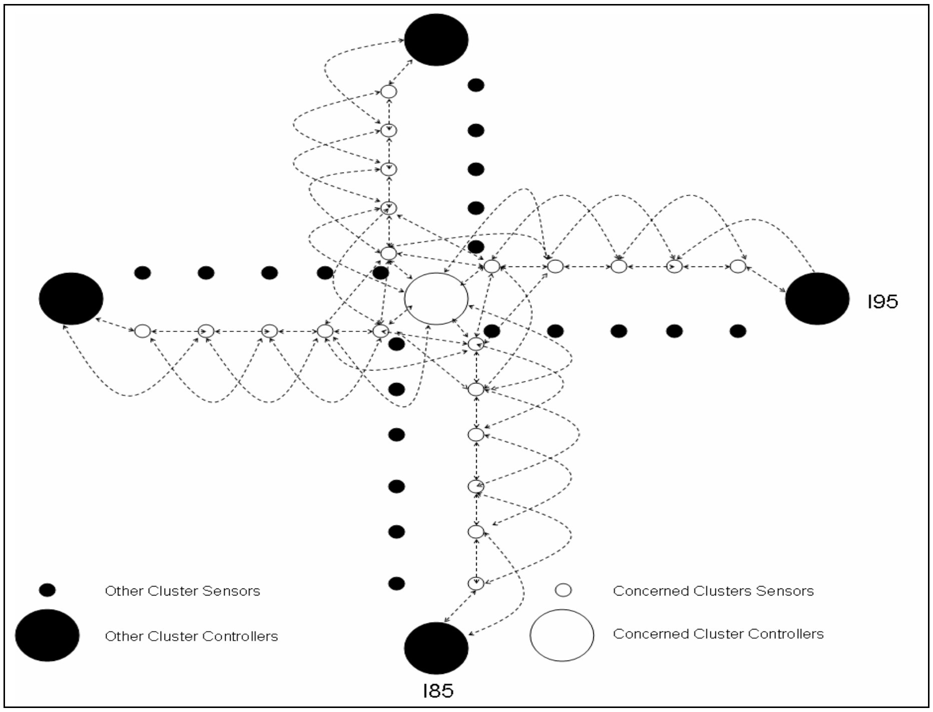
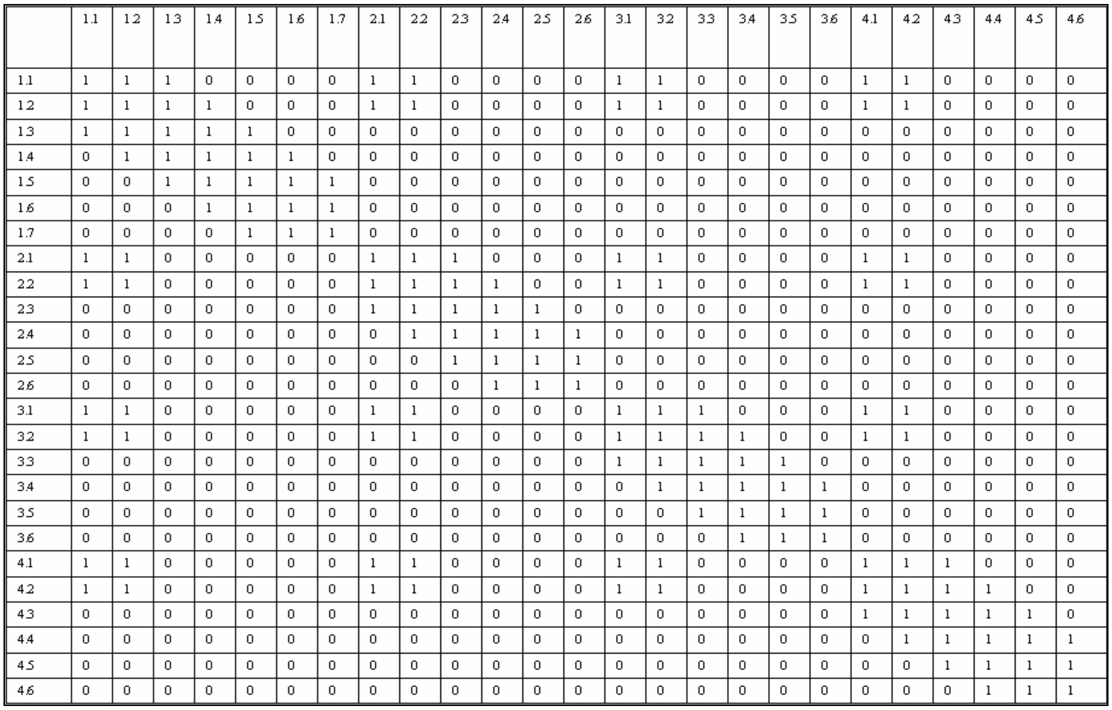

I95
Other Cluster Sensors
Concerned Clusters Sensors
Other Cluster Controllers
Concerned Cluster Controllers
I85

Figure 4.2: Single intersection topology conflict graph



	1.1	1.2	1.3	1.4	1.5	1.6	1.7	2.1	2.2	2.3	2.4	2.5	2.6	3.1	3.2	3.3	3.4	3.5	3.6	4.1	4.2	4.3	4.4	4.5	4.6
1.1	1	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
1.2	1	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
1.3	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.4	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.5	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.6	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1.7	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2.1	1	1	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0
2.2	1	1	0	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0	1	1	0	0	0	0
2.3	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
2.4	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2.5	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2.6	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
3.1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0
3.2	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0
3.3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
3.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0
3.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
3.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
4.1	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	0	0	0
4.2	1	1	0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	1	1	1	1	0	0
4.3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
4.4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
4.5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
4.6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Figure 4.3: Single intersection topology conflict graph matrix representation

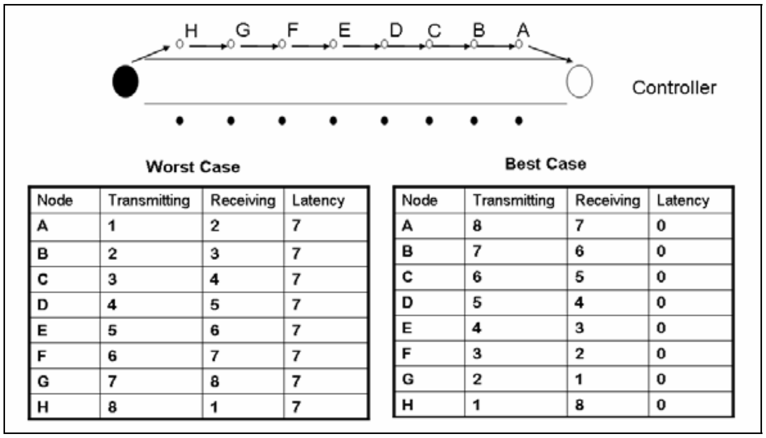
4.1 Fundamental Concepts

The scheduling algorithm is based on the following three key concepts.

4.1.1 Reducing sleep latency

The sensor radios are set to sleep mode to save energy when no communication is needed. For neighboring sensors to communicate, a sleep-awake schedule is agreed on. Packets arriving at a node during sleep durations must wait for the next active period for the receiver and this duration is referred to as the sleep latency [8]. The sleep latency varies for each packet according to their arrival time at every node and the node's sleep-awake schedule. Fig. 4.4 illustrates scenarios for the best and worst case latencies.

As shown in the Fig. 4.4, the child to parent relationship is indicated by arrows pointing from child to parent. Every node is allotted a slot for transmission and the parent of that node will be active in that slot for reception of the packet. With no pipelining the controller would decide a cycle length of 8 as every slot for transmission can be used by only one node on the segment.



Worst Case

Node	Transmitting	Receiving	Latency
A	1	2	7
B	2	3	7
C	3	4	7
D	4	5	7
E	5	6	7
F	6	7	7
G	7	8	7
H	8	1	7

Best Case

Node	Transmitting	Receiving	Latency
A	8	7	0
B	7	6	0
C	6	5	0
D	5	4	0
E	4	3	0
F	3	2	0
G	2	1	0
H	1	8	0

Figure 4.4: Sleep latency

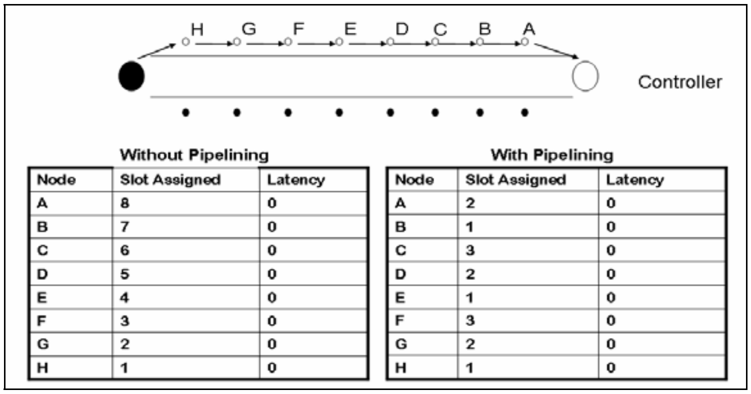
The worst case scenario is when the slot assignment is such that the node that is next to transmit is allotted the slot which is farthest away in time from the current slot. Thus the worst case is when a child is allotted a transmission slot of *m* and parent is allotted a transmission slot in the previous slot. With such assignment the parent gets the packet in slot *m* and it has to wait for (*cycle length - m)* slots till the end of cycle and then for *m - 1* slots in the new cycle to forward the packet. Thus the packet will be buffered at every node for *(cycle_length–1)* slots. The worst case end-to-end sleep latency can therefore be represented as:

*end-to-end sleep latency = (cycle_length-1)*number of nodes*

The best case scenario is if the child node is allotted a transmission in slot *m* and parent is allotted a transmission in slot *m+1*. In such an assignment the packet will not be buffered and the sleep latency is reduced.

4.1.2 Increasing End-to-end Throughput

In order to achieve high throughput the scheduling algorithm assigns slots in a pipelined fashion, whereby, multiple nodes on the same segment in spatially separated regions can transmit in the same slot. Thus spatial reuse helps create a pipeline of transmissions for forwarding end-to-end data along a segment. It is shown in Fig. 4.5 that if the transmission range is affecting the nodes in two hop neighborhoods then with pipelining every node gets a slot every 3 slots and without pipelining it gets a slot every 8 slots. Thus pipelined assignment provides the nodes with more transmission opportunities thereby providing a high end-to-end throughput.



Without Pipelining

Node	Slot Assigned	Latency
A	8	0
B	7	0
C	6	0
D	5	0
E	4	0
F	3	0
G	2	0
H	1	0

With Pipelining

Node	Slot Assigned	Latency
A	2	0
B	1	0
C	3	0
D	2	0
E	1	0
F	3	0
G	2	0
H	1	0

Figure 4.5: Pipelined assignment

4.1.3 Increasing Energy Efficiency

The pipelined assignment provides for higher throughput by assigning more slots to every node but the nodes might require fewer slots based on the input traffic load experienced. A node should save energy by switching to sleep mode in the assigned slot for which it does not have any data to be exchanged. The transmitter in that slot should monitor its input queue and the receiver should monitor the medium for any data for a small interval of time (TA) and if there is no data to be exchanged then the transmitter's queue is empty and receiver will not detect any activity in the medium. Under such a scenario they should switch to the sleep mode thereby saving energy and this concept incorporated into the scheduling is known as the TMAC option.

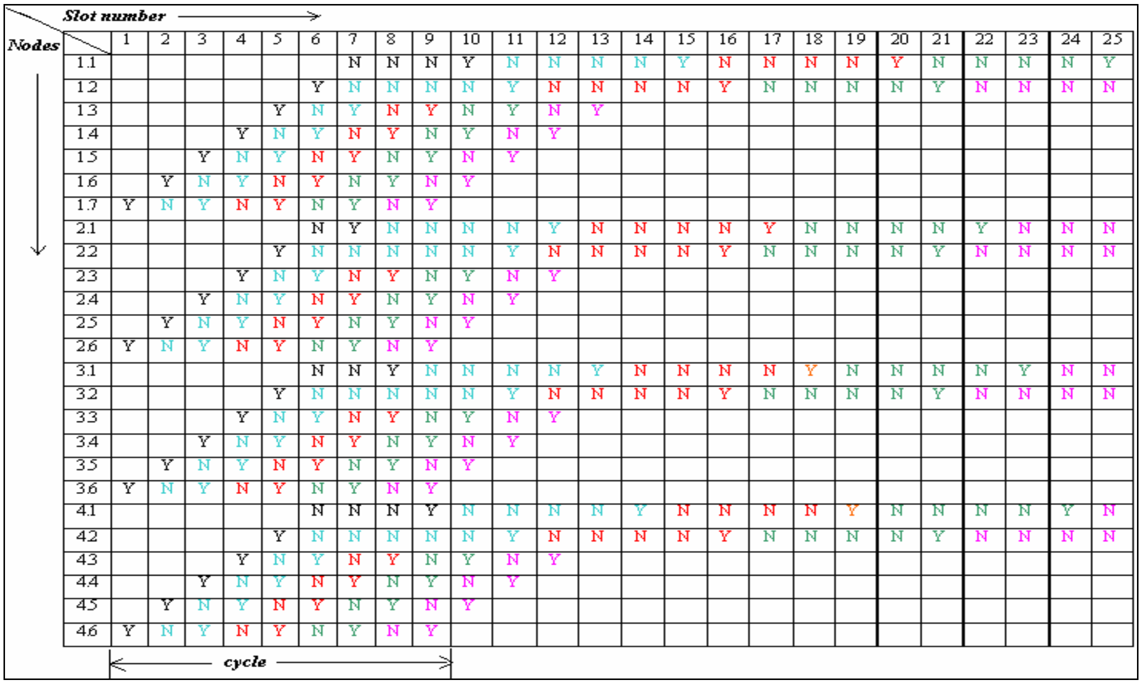
4.2 Scheduling Algorithm

The pseudo code for the scheduling algorithm is given in Fig. 4.6. The controller knows the parent and child relationship for nodes in its cluster and creates a linear tree for every cluster with the farthest node on a segment being the last node on the tree. The controller starts to assign the slots from the farthest node which is also called *end member*

```
CG: conflict graph matrix T: current time slot to be scheduled
T.SendSet: set of nodes scheduled to send in T
T.RecvSet: set of nodes scheduled to receive in T
CS: current candidate segment
CS.NextSegment: next candidate segment following CS
CS.Sender: current candidate sender node in CS
CS.FirstNode: first node in CS
CS.StartSlot: start slot assigned to CS for this round
CS.Sender.Parent: parent node of CS.Sender
CS.rounds[i]: total rounds completed per segment in $i^{th}$ interval
S: Number of slots in one cycle
Cycle_length.schedued : an event which triggers when S slots have been completely scheduled since the last event
START:
T=0; CS=1; CS.Sender=CS.FirstNode; CS.rounds = 0;
while (1)
        if (Cycle_length.schedued )
                flag = SET;
                for (k=i to k=i-20)
                        if (CS.roounds[k] != CS.rounds[k-1])
                                flag = NOT_SET;
                                break;
                        endif
                end for
                if (flag == SET)
                        break;
                else
                        CS.rounds_prev = CS.rounds_curr
                        CONFLICT_TEST_LOOP:
                        for each node n in T.SendSet,
                                if CG(CS.Sender.Parent, n)>0,
                                        T=T+1; GOTO CONFLICT_TEST_LOOP
                                endif
                        end for
                        for each node n in T.RecvSet,
                                if CG(CS.Sender, n)>0,
                                            T=T+1; GOTO CONFLICT_TEST_LOOP
                                else if AGGRESSIVE*CG(CS.Sender.Parent,n)>0,
                                           T=T+1; GOTO CONFLICT_TEST_LOOP
                                endif
                        end for
                       T.SendSet = T.SendSet + CS.Sender
                       T.RecvSet = T.RecvSet + CS.Sensder.Parent
                       if CS.Sender == CS.FirstNode
                                CS.StartSlot = T
                      endif
                      if CS.Sender.Parent is a controller
                                CS.Sender = CS.FirstNode
                                CS.rounds[i] = CS.rounds[i] + 1
                     endif
                     CS.Sender = CS.Sender.Parent
                    CS = CS.NextSegment
                endif
        endif
end while
```

Figure 4.6: Scheduling algorithm.

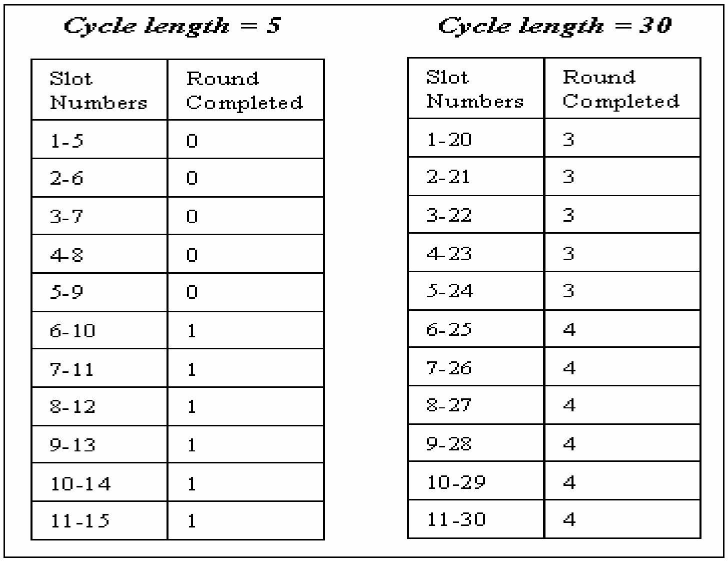
of the cluster. It assigns the first slot to the end member of one cluster and tries to assign the same slot to the end member of the other cluster thereby ensuring fairness between the slots assigned to nodes of different clusters. In case of conflicts the latter node would be assigned the nearest non-conflicting slot. Once one node of every cluster has been assigned a slot the controller starts assigning slots to the parent of the scheduled node of every cluster and the process repeats iteratively. The controller tries to assign the parent the closest possible non-conflicting slot following the slot in which the child was scheduled for transmission. This slot assignment strategy is followed to avoid sleep latency. After all the nodes on the cluster have been scheduled the controller starts assigning slots again to the end members of the clusters and thus the next round of assignment begins. A sample run of the scheduling algorithm over the network shown in Fig. 4.1 and the conflict graph show in Fig. 4.2 is shown in Fig. 4.7, where Y and N indicate the slots in which the node can transmit and can not transmit respectively.



Slot number → Nodes ↓	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1.1							N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y
1.2						Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N
1.3					Y	N	Y	N	Y	N	Y	N	Y												
1.4				Y	N	Y	N	Y	N	Y	N	Y													
1.5			Y	N	Y	N	Y	N	Y	N	Y														
1.6		Y	N	Y	N	Y	N	Y	N	Y															
1.7	Y	N	Y	N	Y	N	Y	N	Y																
2.1						N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N
2.2					Y	N	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N
2.3				Y	N	Y	N	Y	N	Y	N	Y													
2.4			Y	N	Y	N	Y	N	Y	N	Y														
2.5		Y	N	Y	N	Y	N	Y	N	Y															
2.6	Y	N	Y	N	Y	N	Y	N	Y																
3.1						N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N
3.2					Y	N	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N
3.3				Y	N	Y	N	Y	N	Y	N	Y													
3.4			Y	N	Y	N	Y	N	Y	N	Y														
3.5		Y	N	Y	N	Y	N	Y	N	Y															
3.6	Y	N	Y	N	Y	N	Y	N	Y																
4.1						N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N
4.2					Y	N	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N	Y	N	N	N	N
4.3				Y	N	Y	N	Y	N	Y	N	Y													
4.4			Y	N	Y	N	Y	N	Y	N	Y														
4.5		Y	N	Y	N	Y	N	Y	N	Y															
4.6	Y	N	Y	N	Y	N	Y	N	Y																

Figure 4.7: Sample run of scheduling algorithm

Table 4.1: Rounds completed for different cycle lengths



Cycle length = 5

Slot Numbers	Round Completed
1-5	0
2-6	0
3-7	0
4-8	0
5-9	0
6-10	1
7-11	1
8-12	1
9-13	1
10-14	1
11-15	1

Cycle length = 30

Slot Numbers	Round Completed
1-20	3
2-21	3
3-22	3
4-23	3
5-24	3
6-25	4
7-26	4
8-27	4
9-28	4
10-29	4
11-30	4

In Fig. 4.7 it is seen that the nodes are represented as 1.1…1.7, 2.1 … 2.6, 3.1 … 3.6, 4.1 … 4.6 and the slot numbers assigned are 1, 2,… 25. Number of *rounds* completed is defined as the minimum of the number of slots possessed by nodes on a segment at any given time. A *cycle length* to assign schedules to the nodes is initially agreed upon by the different controllers. The controller starts executing the scheduling algorithm and keeps monitoring the rounds completed on the segments for a given *window*. Here window is the same as cycle length and the controller slides the window along the slot numbers to monitor the rounds completed in different windows. When the rounds completed in different windows stabilizes it stops executing the scheduling algorithm. Table 4.1 shows the rounds completed by sliding the window of different cycle lengths along the sample run of scheduling algorithm shown in Fig. 4.7. The controller then selects the cycle for assigning the schedules as the window that gives the maximum rounds completed.

The scheduling algorithm can be modified to assign slots in a weighted manner as there might be scenarios where some segments require higher data rate compared to other segments. Weighted assignment can thereby provide proper division of the bandwidth that can improve the overall network throughput. The scheduling algorithm proposed can perform better with certain variations that are explained in the following section.

4.3 Variants of Scheduling Algorithm

In order to avoid collisions it is necessary to ensure the following rules:

1. The intended sender is not in the carrier sensing range of any other active transmitter.
2. The intended sender is not in the carrier sensing range of any actively receiving node other than the intended receiver.
3. The intended receiver is not in the carrier sensing range of any active transmitter other than the intended transmitter.
4. The intended receiver is not in the carrier sensing range of any other active receiver.

The scheduling solution can be used with different versions to decide a more aggressive schedule for the nodes. There are different assignments possible for the scheduling algorithm and are described below.

Non-aggressive assignment:

The non-aggressive assignment enforces all four rules for conflict avoidance mentioned earlier. Essentially, no other nodes within the carrier sensing range of a scheduled sender and receiver pair are allowed to either transmit or receive.

Aggressive assignment without offset:

By relaxing the fourth rule, more simultaneous transmissions are possible, with the risk of an acknowledgment from one receiver colliding with another valid data transmission. A close inspection of all such events reveals that, due to radio capture effects, not all such conflicts will corrupt the transmissions. In the proposed wireless sensor network the topology structure is such that most of the data transmissions will have high reception power compared to an interfering signal and thus the capture effect helps in correct reception of data packet. This statement can be supported with an example for our network.

Fig. 4.8 shows the scenario used for simulation. The sensors are placed at a distance of 100 meters apart and thus the transmitter and receiver are 100 meters apart. The transmission power at sensors is adjusted such that the transmission range is 128 meters and the carrier sensing range is 256 meters. Because of rule 3 there is no active interfering node transmitting data in the carrier sensing range of the receiver B. Rule 1 and rule 2 ensure that there is no node transmitting or receiving within 256 meters from transmitter A, which implies that there is no interfering receiver within 156 meters of receiver B. The received power at B for a transmission by A is such that any node beyond a radius of 175 meters from B cannot cause interference at the receiver. The *interfering region* is defined as the region outside the sensing range of the transmitter and inside the interfering range (175 meters from B). Thus only a receiver in the interfering region shown in Fig. 4.8 transmitting an acknowledgement can cause collision at the receiver B. The aggressive scheduling is thereby expected to achieve better performance compared to non-aggressive scheduling.

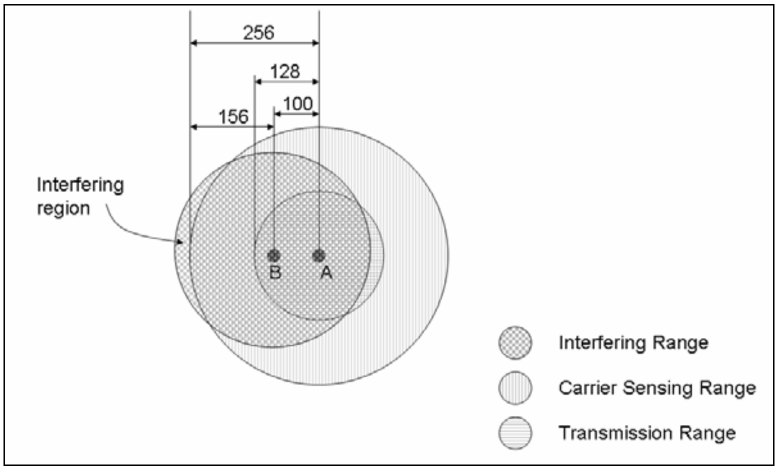


Figure 4.8: Example arrangement of transmitter and receiver

Aggressive assignment with offset:

Fig. 4.9 shows that there are four possible scenarios representing the order in which a data packet and a simultaneously interfering acknowledgement are received. These scenarios are represented as scenario A, B, C and D. It is assumed that the difference in the power of the data packet being received and that of the interfering acknowledgement is more than the capture ratio κ. Under such conditions the capture effect can help the reception of the data packet provided the packets are received in a certain order. Fig. 4.9 shows that only in the fourth scenario (scenario D) will the receiver not correctly receive the data packet. The capture effect helps the reception of the correct packet for scenario A while in the scenarios B and C the packets are received at different times and thus there is no chance of a collision.

To look at an example, consider our sample network in Fig.4.1, where nodes are denoted as *segment.member* and node 1.2 is sending data packets to 1.1 while node 2.3 is sending data packets to 2.2 in the same slot.

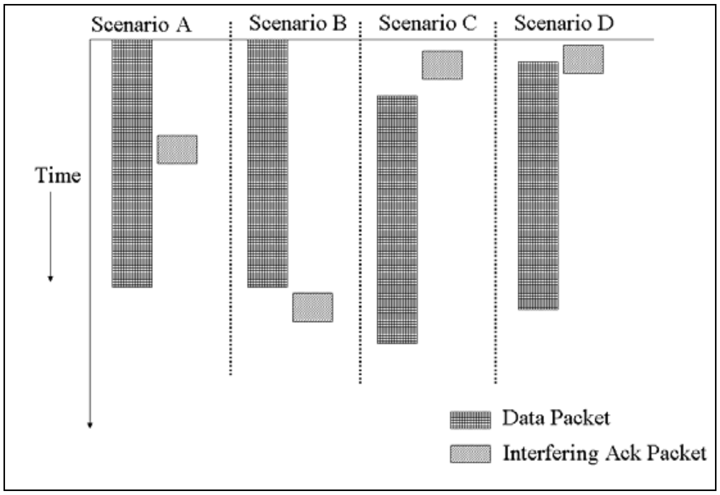


Figure 4.9: Capture effect and offsets

Let us assume that the third data packet from 2.3 starts when 1.1 is transmitting its acknowledgement to 1.2, node 2.2 will not be able to correctly receive from 2.3 since the received power is not much less than the ongoing acknowledgement's. Instead, if node 2.3 had started to transmit data prior to the start of the acknowledgement, node 2.2 would have successfully locked the reception of the packet. If nodes are provided different offsets then the occurrence of scenario D can be minimized. This suggests the benefit of nodes scheduled in the same slot to start transmission with different offsets to their slot's start time. As shown in Fig. 4.9 the order of arrival of packets at the receiver determines whether the packet is lost or captured and this order in turn is determined by the time at which the first packet is sent out at the beginning of the time slot. As shown in Fig. 4.10, offsets help to reduce the possibility of a packet loss but do not totally avoid the possibility of a loss. In wireless sensor networks for highway traffic monitoring, the data is predictable and collected periodically. If the data packet lengths are predictable then the offsets help to provide the necessary change in timing to avoid collisions.

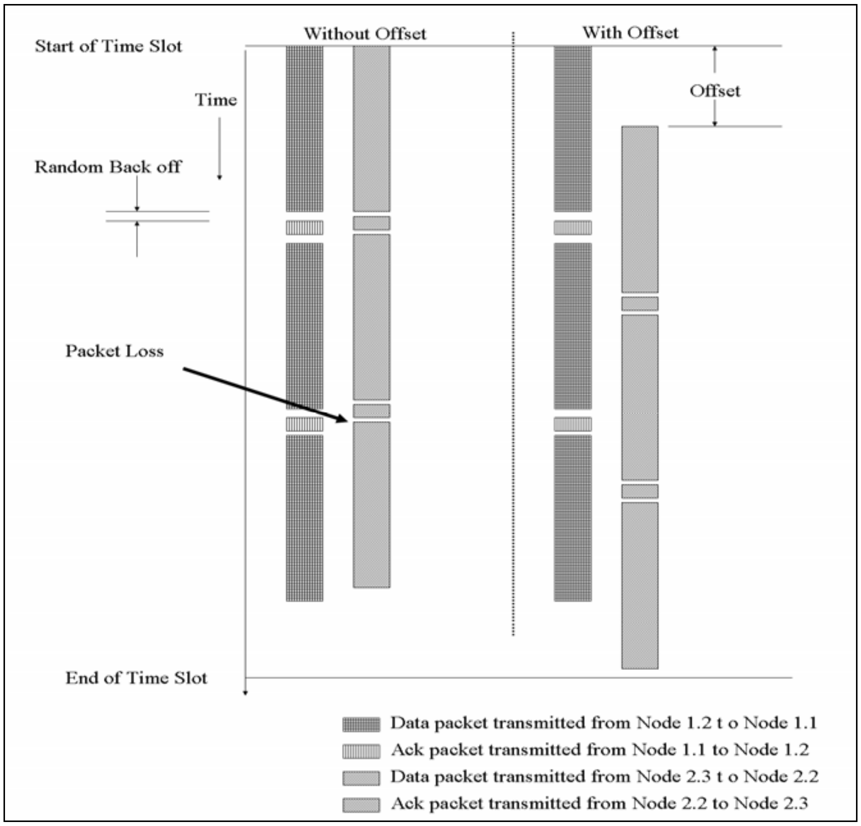

Start of Time Slot
Without Offset
With Offset
Time
Offset
Random Back off
Packet Loss
End of Time Slot
Data packet transmitted from Node 1.2 t o Node 1.1
Ack packet transmitted from Node 1.1 to Node 1.2
Data packet transmitted from Node 2.3 t o Node 2.2
Ack packet transmitted from Node 2.2 to Node 2.3

Figure 4.10: Without offset and with offset packet transmissions

The choice of the offset is very critical in determining the achieved improvement in the throughput. The *slot length* is defined as the time duration of a single slot. Given a slot length there can be *k* packets expected to be transmitted in that particular slot length. The time taken to completely transmit one packet is defined as the packet *completion time* and this time is the sum of DIFS, packet transmission time, SIFS, acknowledgement transmission time and a random back off between transmissions. The completion time is thus variable depending on the random back off and the *average completion time* is defined as the average of completion times for different packets of the same length. Thus the total time necessary *T* to transmit k packets is given by

$$T = k*average\ completion\ time$$

The value of (slot length – T) gives the *upper threshold* on the total offset that can be provided to the nodes. The offset is used to avoid the packet loss between a weak acknowledgement and a strong data packet and if there are *N* segments then each will require an offset that is a multiple of *basic offset* θ. The value of θ is determined as:

$$\theta = (slot\ length - T) / N$$

Effects of slot length

The choice of slot length greatly affects the end-to-end delay. Within a cycle length, based on the scheduling assignment, there are certain slots assigned to every node for transmission. Fig. 4.7 shows the assignment of the first 25 slots and it is seen that node 1.1 gets the 10th, 15th, 20th and 25th slot for transmission in a cycle length of 30 slots. The system will be in its stable state if the input rate can be supported by the bottleneck node. Node 1.1 is a bottleneck node for cluster 1 as it is located in high contention region and thereby receives the least number of slots. Considering the packet size to be 1000 bytes, 4 packets (32 kbits) can be transmitted in a slot length of 0.1 seconds. The bottleneck node 1.1 gets 4 slots in one cycle of length 30 and thus in every cycle it can transmit 128 kbits of data. The cycle length is 30 * 0.1 = 3 seconds. The *scheduled throughput* is the input rate that can be supported by the bottleneck node on the segment and its value for segment 1 is 42.6 kbps. If we consider a slot length of 0.2 seconds then every node can transmit 8 packets in a slot and thus after 4 slots a node can transmit 256 kbits of data. However the total time for the cycle length is 30 * 0.2 = 6 seconds showing that the supported input rate can be 42.6 kbps. Thus the scheduled throughput for every segment remains the same irrespective of the slot length.

The end-to-end delay is proportional to the slot length. For comparison let us consider the same two scenarios as mentioned earlier with the input rate constrained such that the system is in the stable state. Fig. 4.7 shows the sample run of scheduling algorithm and considering a cycle of 30 slots node 1.7 is assigned slot number 1, 3, 5, 7, 9, 11, …, for transmission and node 1.1 is assigned slot number 10, 15, 20 and 25 for transmission. It is observed that 4 slots in every cycle are sufficient to support the input rate and thus node 1.7 uses only 4 of its assigned slots for transmitting data. It is assumed that these slots are the first four slots of the cycle and this is done to achieve fairness in comparison. Table 4.2 shows the end to end delay for packets transmitted along segment 1. Looking at Table 4.2, the average end-to-end delay for a slot length of 0.1 and 0.2 are given in equations 4.1 and 4.2 respectively.

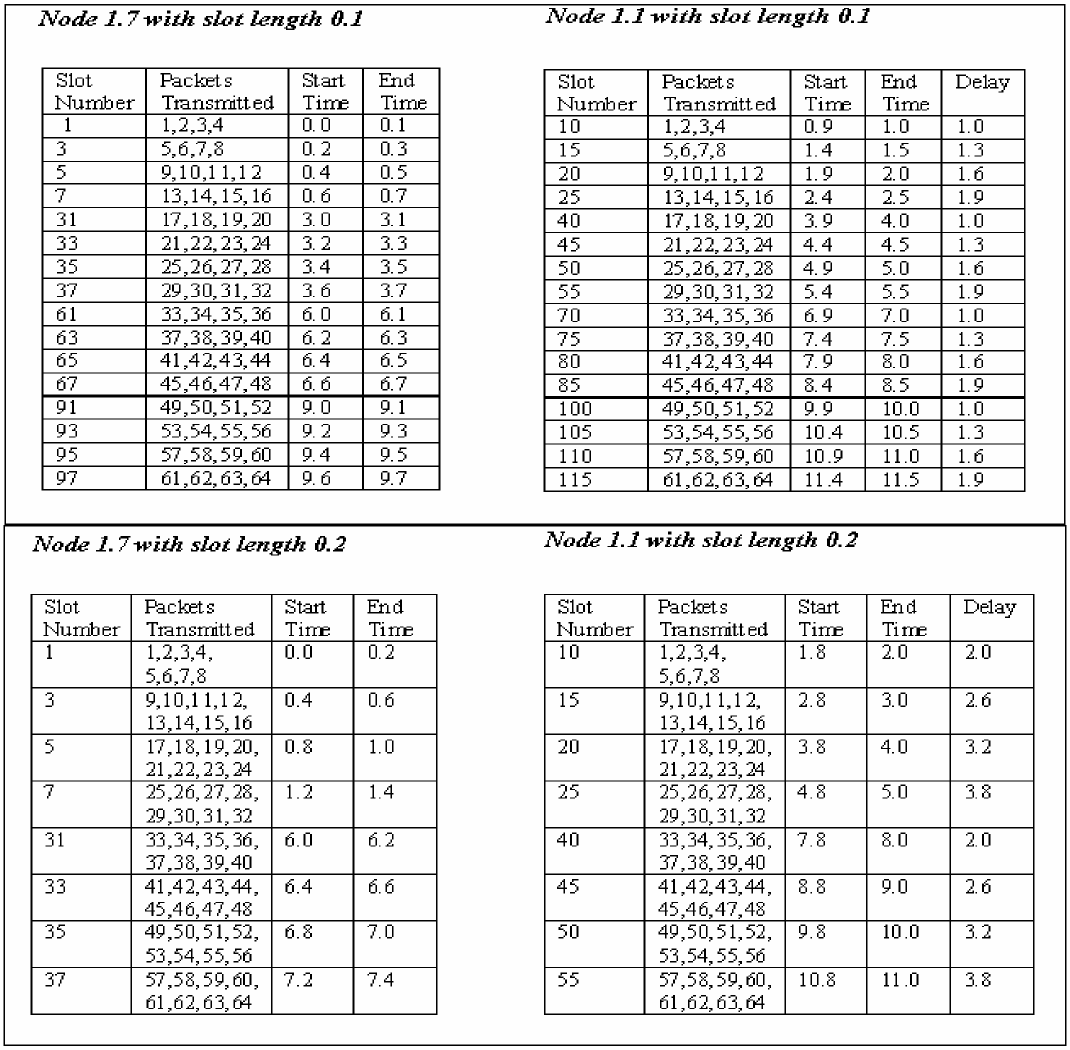
average end-to-end delay (for slot length 0.1)=(1.0+1.3+1.6+1.9)/4 = 1.45 seconds (4.1)

average end-to-end delay (for slot length 0.2)=(2.0+2.6+3.2+3.8)/4 = 2.90 seconds (4.2)

The average end-to-end delay with a slot length of 0.1 seconds is 1.45 seconds and that with a slot length of 0.2 seconds is 2.9 seconds thereby indicating that the end-to-end delay is directly proportional to the slot length.

average end-to-end delay = slot length * constant

Table 4.2: End-to-end delay and slot length



Node 1.7 with slot length 0.1

Slot Number	Packets Transmitted	Start Time	End Time
1	1,2,3,4	0.0	0.1
3	5,6,7,8	0.2	0.3
5	9,10,11,12	0.4	0.5
7	13,14,15,16	0.6	0.7
31	17,18,19,20	3.0	3.1
33	21,22,23,24	3.2	3.3
35	25,26,27,28	3.4	3.5
37	29,30,31,32	3.6	3.7
61	33,34,35,36	6.0	6.1
63	37,38,39,40	6.2	6.3
65	41,42,43,44	6.4	6.5
67	45,46,47,48	6.6	6.7
91	49,50,51,52	9.0	9.1
93	53,54,55,56	9.2	9.3
95	57,58,59,60	9.4	9.5
97	61,62,63,64	9.6	9.7

Node 1.1 with slot length 0.1

Slot Number	Packets Transmitted	Start Time	End Time	Delay
10	1,2,3,4	0.9	1.0	1.0
15	5,6,7,8	1.4	1.5	1.3
20	9,10,11,12	1.9	2.0	1.6
25	13,14,15,16	2.4	2.5	1.9
40	17,18,19,20	3.9	4.0	1.0
45	21,22,23,24	4.4	4.5	1.3
50	25,26,27,28	4.9	5.0	1.6
55	29,30,31,32	5.4	5.5	1.9
70	33,34,35,36	6.9	7.0	1.0
75	37,38,39,40	7.4	7.5	1.3
80	41,42,43,44	7.9	8.0	1.6
85	45,46,47,48	8.4	8.5	1.9
100	49,50,51,52	9.9	10.0	1.0
105	53,54,55,56	10.4	10.5	1.3
110	57,58,59,60	10.9	11.0	1.6
115	61,62,63,64	11.4	11.5	1.9

Node 1.7 with slot length 0.2

Slot Number	Packets Transmitted	Start Time	End Time
1	1,2,3,4, 5,6,7,8	0.0	0.2
3	9,10,11,12, 13,14,15,16	0.4	0.6
5	17,18,19,20, 21,22,23,24	0.8	1.0
7	25,26,27,28, 29,30,31,32	1.2	1.4
31	33,34,35,36, 37,38,39,40	6.0	6.2
33	41,42,43,44, 45,46,47,48	6.4	6.6
35	49,50,51,52, 53,54,55,56	6.8	7.0
37	57,58,59,60, 61,62,63,64	7.2	7.4

Node 1.1 with slot length 0.2

Slot Number	Packets Transmitted	Start Time	End Time	Delay
10	1,2,3,4, 5,6,7,8	1.8	2.0	2.0
15	9,10,11,12, 13,14,15,16	2.8	3.0	2.6
20	17,18,19,20, 21,22,23,24	3.8	4.0	3.2
25	25,26,27,28, 29,30,31,32	4.8	5.0	3.8
40	33,34,35,36, 37,38,39,40	7.8	8.0	2.0
45	41,42,43,44, 45,46,47,48	8.8	9.0	2.6
50	49,50,51,52, 53,54,55,56	9.8	10.0	3.2
55	57,58,59,60, 61,62,63,64	10.8	11.0	3.8

4.4 Simulation Studies

The key concepts explained in the section 4.1 are analyzed with the help of simulations carried out in ns-2. The following sections provide the simulation results.

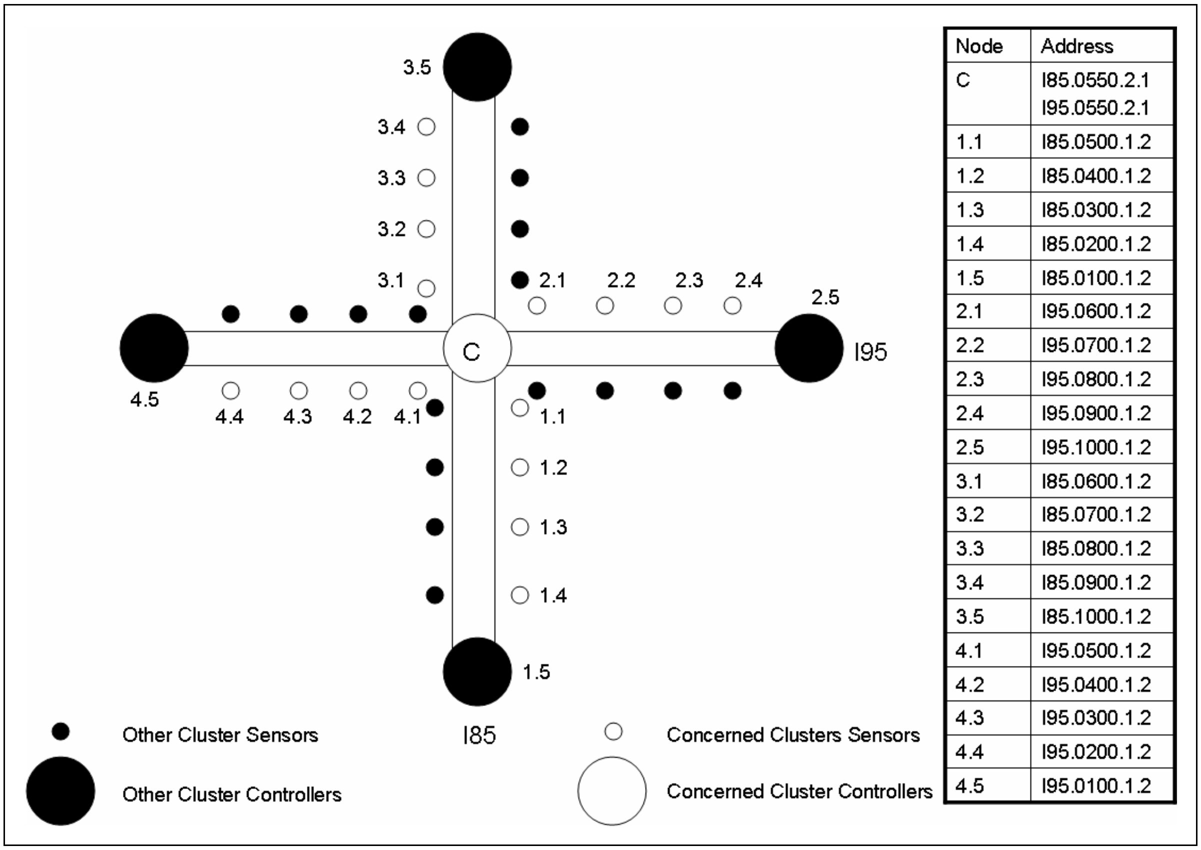
4.4.1 Simulation Settings

The scenario shown in Fig. 4.11 involves 4 clusters along the 4 associated segments of the concerned cluster controller, each consisting of 5 sensors. The number of sensors on the segment indicates the *cluster length*. The sensors are placed at a distance of 100 meters and are addressed according to their location on the highway. The sample addresses are shown in the Fig. 4.11. The simulation parameters are given in Table 4.3.

Each road segment has one exponential ON/OFF 50%-duty-cycle UDP flow from each "other cluster controller" to the concerned cluster controller. The packet size used for the simulations is fixed at 1000 Bytes. The input rate during the experiment is varied from 15 kbps to 165 kbps. Since only one intersection is considered, no traffic is generated along the other direction, and the "other cluster sensors" are inactive in this scenario. The simulation has four exponential flows set up from the four end sensors (i.e 1.5, 2.5, 3.5 and 4.5) flowing towards the intersection controller which is also their cluster head. This is the basic simulation settings and it is referred to as the *basic single intersection* scenario. *Conflict set* is defined as the set of nodes in a given network such that all the nodes are interfering with each other. A node X is said to be interfering with another node Y, if either the node itself or its receiver is in carrier sensing range of the node Y. The *conflict degree* of a network is the number of nodes in the maximum conflict set.

Table 4.3 Simulation parameters

Parameter	*Abbreviation*	*Value*
Receiving Threshold	RXThresh	4.99e-9
Sensing Threshold	CSThresh	3.12e-10
Transmit Power	Px	0.265 W
Capture Threshold	CPThresh	10.0
Transmission Range	Tx	128 meters
Carrier Sensing Range	Cx	256 meters



Node	Address
C	I85.0550.2.1 I95.0550.2.1
1.1	I85.0500.1.2
1.2	I85.0400.1.2
1.3	I85.0300.1.2
1.4	I85.0200.1.2
1.5	I85.0100.1.2
2.1	I95.0600.1.2
2.2	I95.0700.1.2
2.3	I95.0800.1.2
2.4	I95.0900.1.2
2.5	I95.1000.1.2
3.1	I85.0600.1.2
3.2	I85.0700.1.2
3.3	I85.0800.1.2
3.4	I85.0900.1.2
3.5	I85.1000.1.2
4.1	I95.0500.1.2
4.2	I95.0400.1.2
4.3	I95.0300.1.2
4.4	I95.0200.1.2
4.5	I95.0100.1.2

Figure 4.11: Four segment single intersection scenario

4.4.2 Performance metrics for topology changes

The following section explains the effects of topology changes on the available network throughput.

Throughput versus Cluster Length

The basic single intersection scenario shown in Fig. 4.11 has a conflict degree of 8. The nodes that are away from the controller are in a less dense area and thus can be allotted transmission slots more frequently than the nodes in the denser area. Even if the number of sensors along the highways is different for different segments the achievable end-to-end throughput is limited by the bottleneck node. To demonstrate this effect, simulation scenarios were set up with different number of sensors along four segments of the same basic single intersection scenario. There were 5 different scenarios set up with 5, 10, 15, 20 and 25 sensors along each of the four segments. It was found that for each of the five scenarios the bottleneck nodes got 50 slots out of a cycle length of 400 slots. The nodes got 50 slots as they were facing a conflict degree of 8 and thus the 400 slots were divided among the 8 nodes fairly thereby showing that the available throughput is independent of the cluster length. To demonstrate this by simulations, the different scenarios were simulated and exponential flows were set up over the 4 segments. Figure 4.12 shows that the saturation throughput achieved is independent of the number of sensors on the segment. *Saturation throughput* is defined as the input rate that can be supported with 90% of the packets delivered successfully.

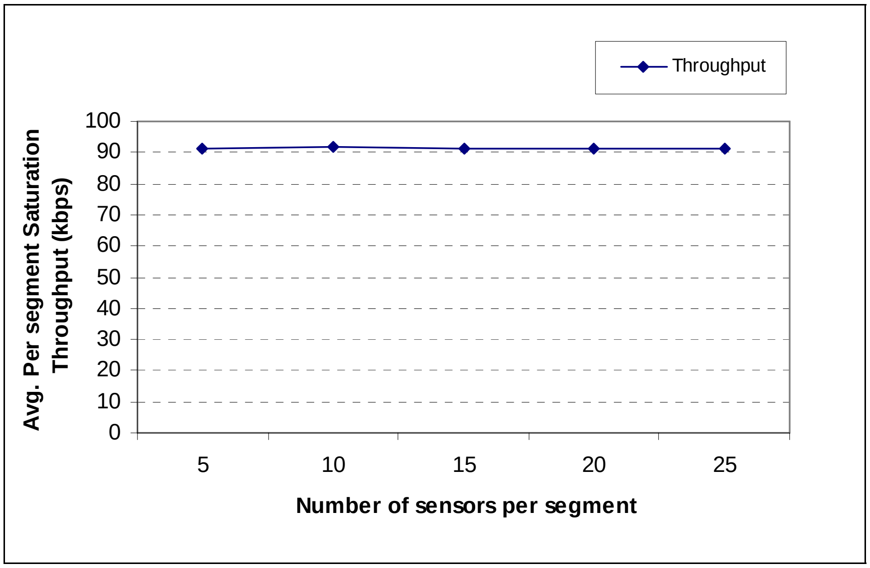


Figure 4.12: Throughput VS Number of sensors per segment

Throughput versus Degree of Conflict

As the degree of conflict increases the available end-to-end throughput decreases because the available slots are divided between more nodes which can not be transmitting at the same time. As shown in Fig. 4.13, four scenarios A, B, C and D with degree of conflict 4, 8, 12 and 14, respectively, are simulated. Scenario E is a modified version of D which has the node placements such that degree of conflict is 16. It can be seen from Fig. 4.14 that the saturation throughput achieved is inversely proportional to the degree of conflict.

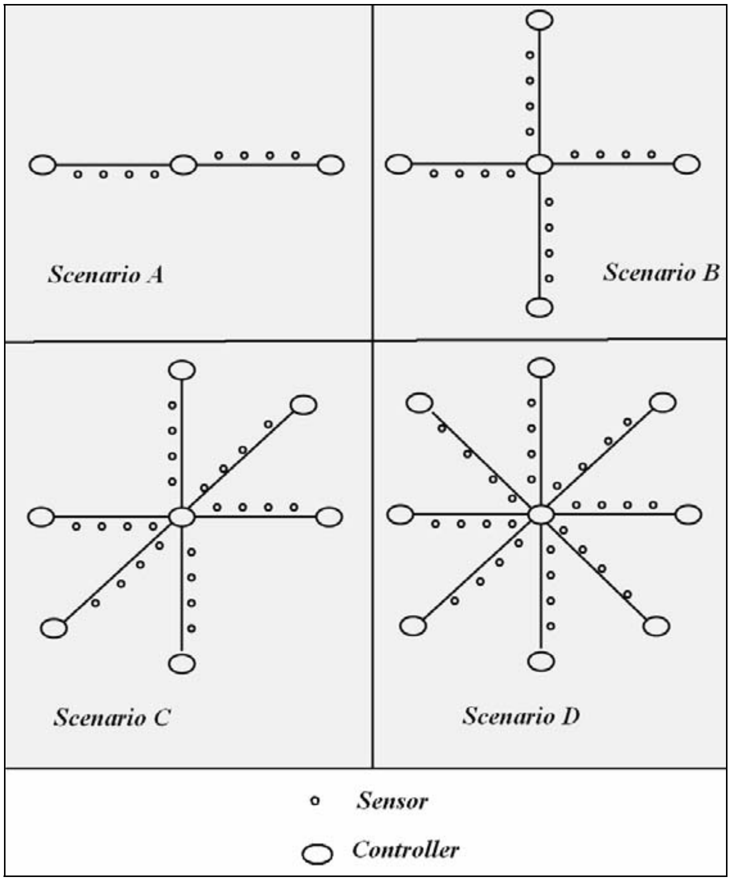


Figure 4.13: Topologies corresponding to different degree of conflict

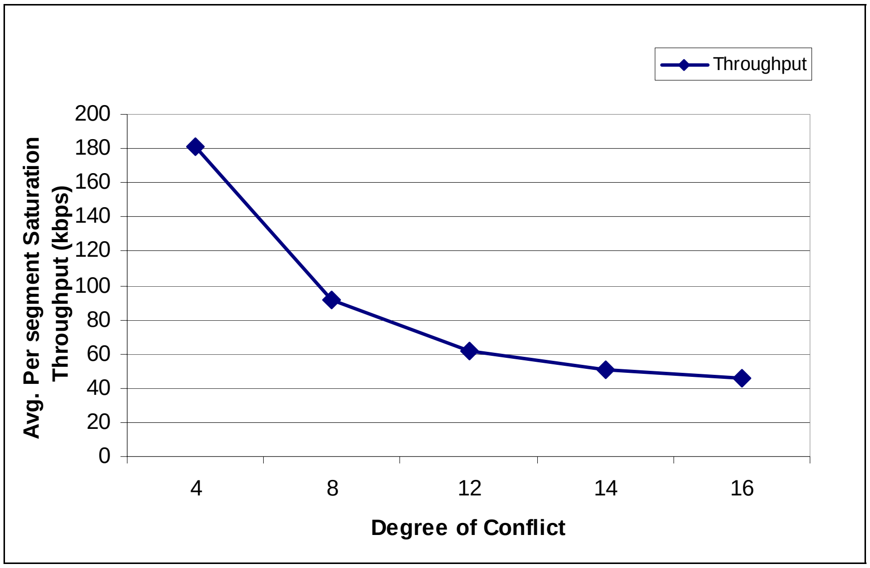


Figure 4.14: Throughput VS degree of conflict

4.4.3 Performance metrics for algorithm variations

As explained in the earlier sections there are possible variations to the algorithm which try to improve the throughput performance available from the scheduling algorithm. Fig. 4.15 shows the per-segment end-to-end throughput for the non-aggressive, the aggressive without offset, and the aggressive with offset algorithms for the basic single intersection scenario shown in Fig. 4.11. A threshold line is plotted to indicate the 90% success rate boundary, beyond which the network becomes unstable and the throughput and delay measures becomes less meaningful due to the excessive number of packets dropped. As shown in Fig. 4.15, unscheduled IEEE 802.11 is able to provide stable per-segment throughput up to approximately 58 kbps, while the non-aggressive method achieves 82 kbps, a 35% increase. The aggressive without offset provided a stable throughput at 92 kbps, which is a 51% increase. The aggressive with offset gave a stable throughput at 100 kbps, which is a 63% increase.

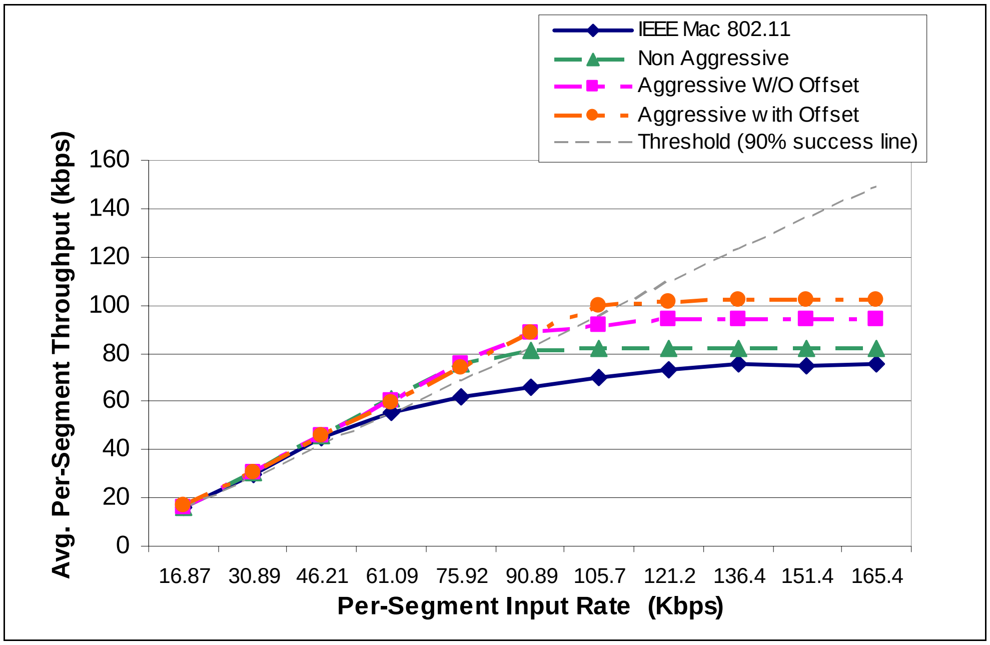


Figure 4.15: Throughput for different algorithms

Fig.4.16 shows the average per-segment end-to-end packet forwarding delays measured for the different algorithms and the unscheduled IEEE 802.11. The end-to-end delay consists mainly of the queuing and transmission delays incurred at each node. Without collisions, the transmission time with the scheduled approach is approximately constant (initial random backoff + data transmission + inter-frame spacing + acknowledgement). The end-to-end delay is almost the same for the different algorithms but smaller for unscheduled IEEE 802.11 as it does not incur any sleep latency. As shown in the Fig. 4.15 the different algorithms have different saturation throughputs and it is seen in Fig. 4.16 that the end-to-end delay starts rising as the systems get close to saturation. This is because the queuing delay starts increasing rapidly thereby affecting the end-to-end delay.

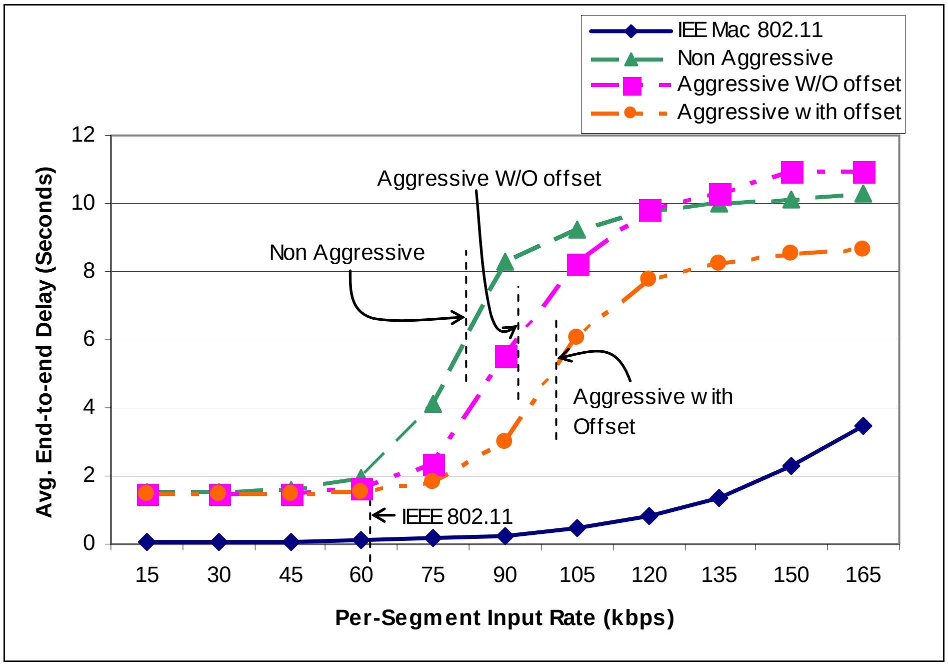


Figure 4.16: End-to-end delay for different algorithms

4.4.4 Performance metrics for different slot lengths

Experiments were run with different slot lengths for one selected algorithm (Aggressive without offset). The time to complete the transmission of a packet is denoted by *tx*. In every slot a node can not transmit in the last *tx* seconds as it has to finish transmitting the packet before the end of slot. It is explained in section 4.3 that the scheduled throughput remains the same for different slot lengths (*sl*). However, the achievable throughput differs slightly based on the slot length:

$$\textit{achievable throughput} = (sl\text{-}tx)/tx * \textit{scheduled throughput} \qquad (4.3)$$

It can be seen from the Fig. 4.17 that the throughput differs slightly based on the slot length. However, as the slot length increases the improvement is throughput becomes smaller.

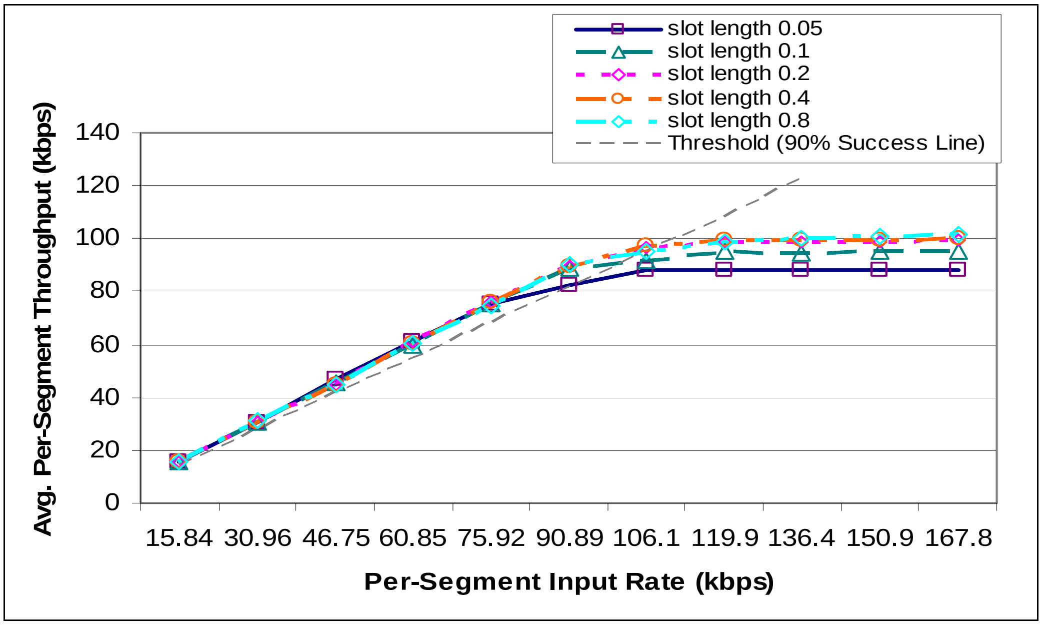


Figure 4.17: Throughput for different slot lengths

The slot length is an important parameter determining the end-to-end delay. The end-to-end delay consists of the queuing and transmission delays and sleep latency incurred at each node. The sleep latency depends on the chosen slot length; the longer slot length, the longer average sleep latency. Fig. 4.18 shows that the end-to-end delay for the scheduled approach is clearly proportional to the chosen slot length. As the slot length reduces, the delay reduces to approach that of IEEE 802.11. For example, the smallest slot length simulated was 0.05 seconds incurring a small end-to-end delay of approximately 0.7 seconds. Thus it is seen that by changing the slot length from 0.05 seconds to 0.8 seconds the achievable throughput changes from 82 kbps to 98 kbps, an increase of 19%. On the other hand the end-to-end delay changes from 0.7 seconds to 10.7 seconds, an increase of 1428%.

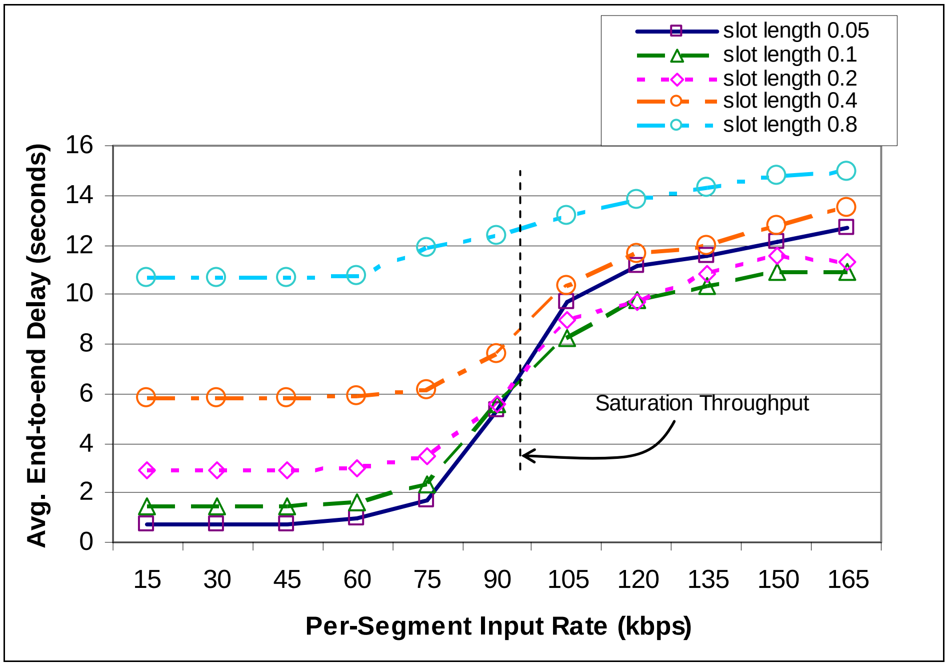


Figure 4.18: End-to-end delay for different slot lengths

Per Hop Delay Analysis

An analysis of per-hop delay shows that the maximum delay is incurred when packets are forwarded by the bottleneck nodes. Considering the network in Fig. 4.11 it is seen that the nodes that are not more than three hops away from the controller are the bottleneck nodes as they are assigned only 50 slots for transmission in a cycle length of 400 slots. Given a schedule cycle every node is assigned certain slots for transmission. One frame interval is defined as the time elapsed from the beginning of one scheduled slot to the beginning of the next scheduled slot. The average frame size F for a node is defined as the average of all frame intervals for that node in the given schedule cycle. For a given network, bottleneck frame size F_{BTN} indicates the avg. frame size of the bottleneck nodes

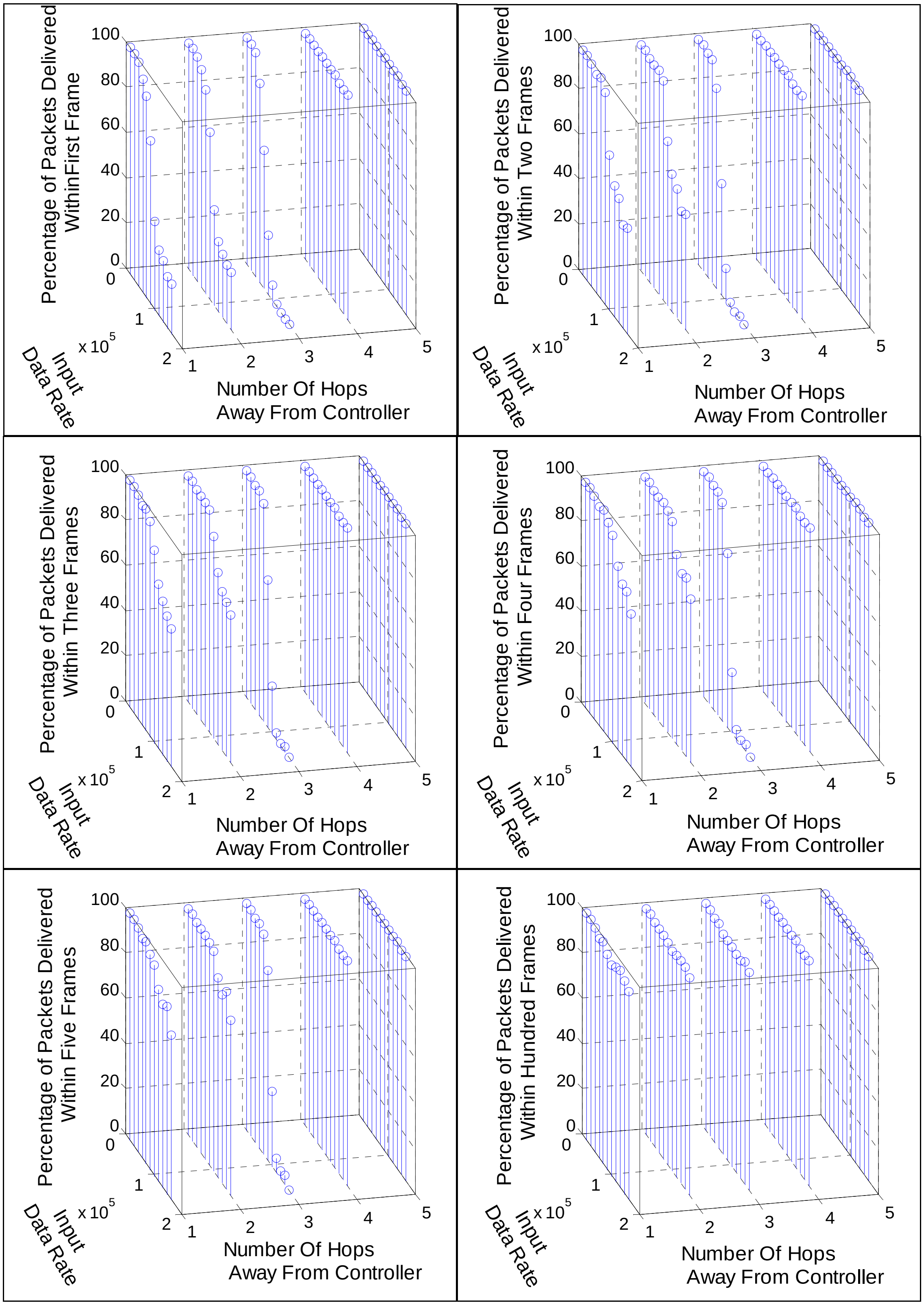


Figure 4.19: Per hop delay analysis

For the network in Fig. 4.11, F_{BTN} is 0.8 seconds. Fig. 4.19 shows that at high input rates the nodes that are more than 3 hops away from the controller forward most of their packets with a delay less than one F_{BTN} as they have more scheduled slots to transmit as compared to the other nodes. However, the nodes that are one, two and three hops away from the controller have fewer slots and thus are not able to transmit all the packets with a delay less than one F_{BTN} at high input rates. The longest queuing delays are at the nodes 3 hops away from the controller as these nodes are the first bottleneck from where the packets have to be forwarded.

4.4.5 Performance metrics for weighted assignments

Some road segments might require high rate of data communication as compared to others. In such scenarios it is advantageous to divide the total capacity among the segments according to their needs. In this section the advantages of having weighted assignment over non-weighted assignment are shown. Non-weighted assignment assigns the weight of 1:1:1:1 on the four segments irrespective of the throughput requirements whereas the weighted assignment considers the throughput requirements and assigns the weights accordingly. With different weights assigned in the scheduling algorithm, the same basic single intersection scenario with different throughput requirements on the four segments as listed in Table 4.4 are studied. It was seen earlier that with a ratio of 1:1:1:1 all the segments are able to achieve a throughput of 92 kbps thereby achieving a total network throughput (network capacity) of 368 kbps. The total network capacity is divided according to the ratio of traffic on segments to give the input rate that the system should be able to support as shown in Table 4.4.

Table 4.4: Input rates supported by system

	Ratio for segments seg1:seg2:seg3:seg4	Network Capacity	Supported Input Rate (kbps)
Scenario A	1:1:1:1	368 kbps	92:92:92:92
Scenario B	2:1:1:1	368 kbps	147.2:73.6:73.6:73.6
Scenario C	1:2:2:2	368 kbps	52.57:105.1:105.1:105.1
Scenario D	1:2:3:4	368 kbps	36.8:73.6:110.4:147.2

Fig. 4.20 shows the achieved throughput for scenario A. It is seen that weighted scheduling and non-weighted scheduling provide the same throughput for all the four segments and unscheduled IEEE 802.11 provides lower throughput comparatively. Fig. 4.21 shows the achieved throughput for scenario B and it is seen that the weighted scheduling provides higher throughput for the first segment as it can support 142 kbps as compared to 102 kbps for unscheduled IEEE 802.11 and 92 kbps for non weighted scheduling. For the other three segments the input rate is less than 92 kbps and thereby both weighted and non-weighted scheduling can support that input rate to give a per-segment throughput of 73 kbps. Fig. 4.22 shows the achieved throughput for scenario C. The last three segments have an input rate of 105 kbps which cannot be supported by non weighted scheduling. It is able to provide a throughput of 92 kbps and unscheduled IEEE 802.11 is able to provide a throughput of 80 kbps. The weighted scheduling is able to support the input rate of 105 kbps on all the three segments. Fig. 4.23 shows the achieved throughput for scenario D. It is seen that segments three and four have an input rate greater than 92 kbps and thus the non-weighted scheduling can only provide throughput of 92 kbps and the weighted scheduling is able to provide a throughput of 105 kbps and 142 kbps for segments 3 and 4 respectively.

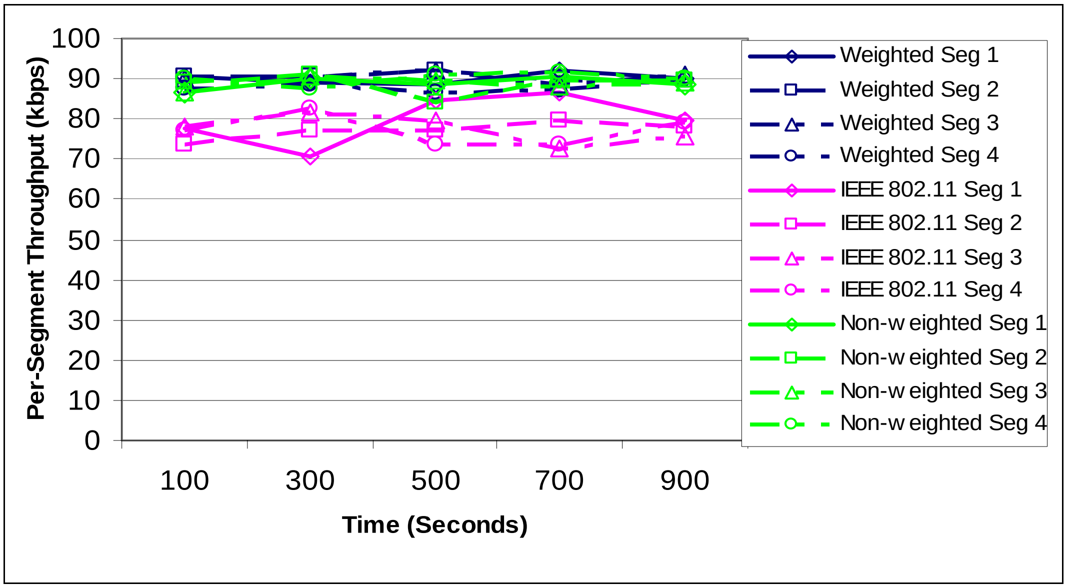


Figure 4.20: Per segment throughput - scenario A

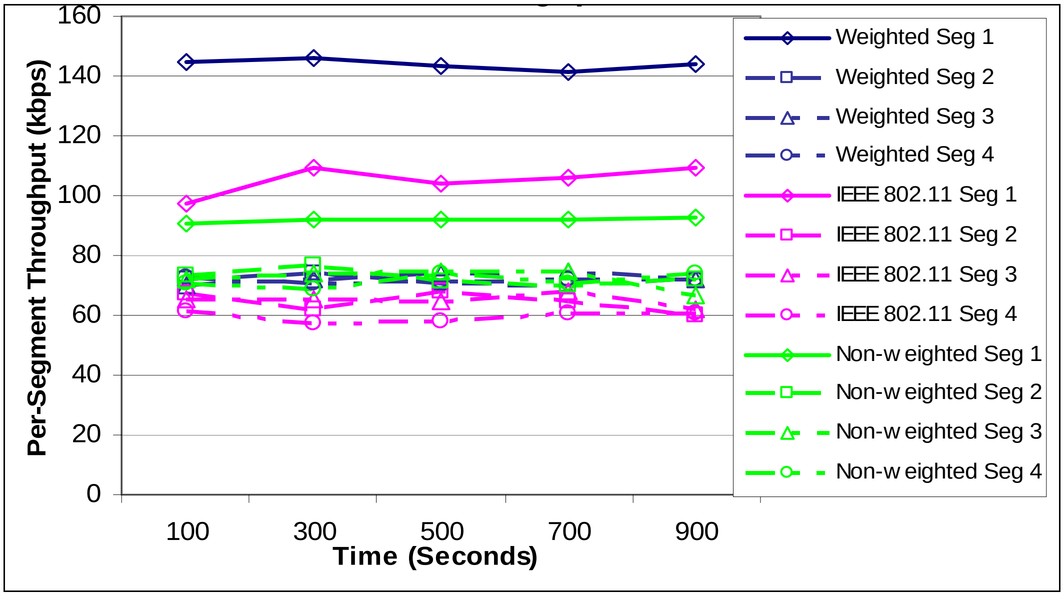


Figure 4.21: Per segment throughput - scenario B

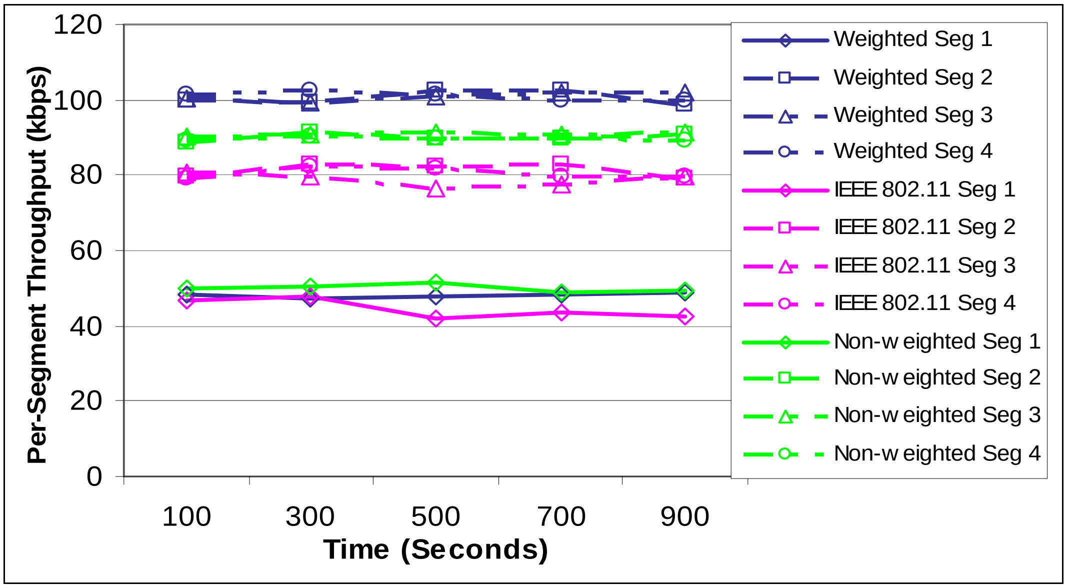


Figure 4.22: Per segment throughput - scenario C

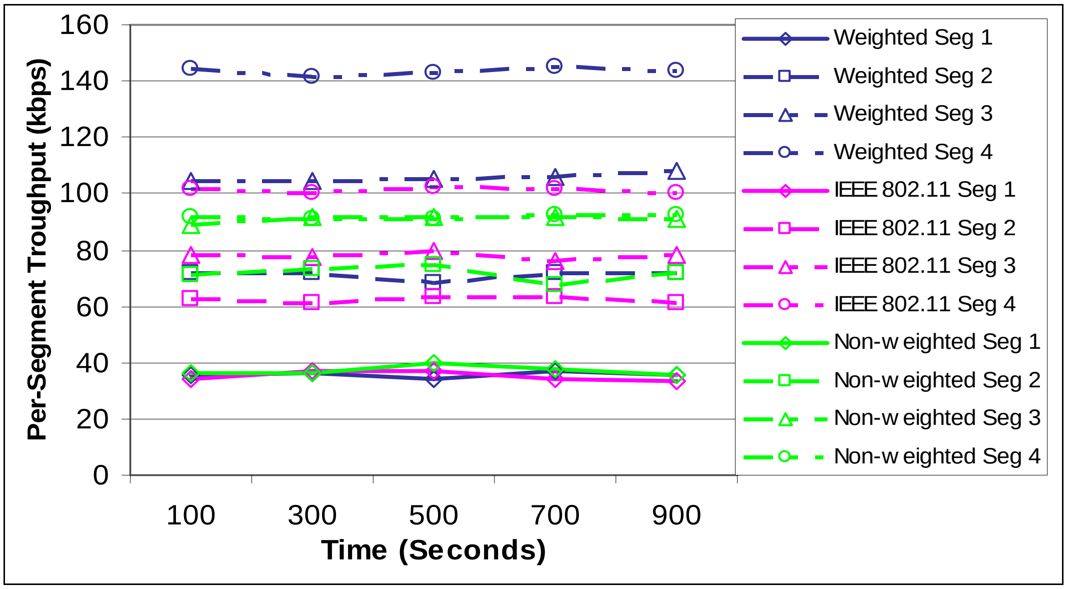


Figure 4.23: Per segment throughput - scenario D

The throughput analysis for the four scenarios is summarized in Table 4.5.

Table 4.5: Per segment throughput and network throughput

Weighted Assignment										
	Input Rate for segments (kbps)				Total	Output Rate for segments (kbps)				Total
	Seg 1	Seg 2	Seg 3	Seg 4		Seg 1	Seg 2	Seg 3	Seg 4	
Scenario 1	92	92	92	92	368	90	90	90	90	360
Scenario 2	147.2	73.6	73.6	73.6	368	144	72	72	72	360
Scenario 3	52.57	105.14	105.14	105.14	368	48	102	102	102	354
Scenario 4	36.8	73.6	110.4	147.2	368	36	71.5	105	143.5	356
Non-Weighted Assignment										
	Input Rate for segments (kbps)				Total	Output Rate for segments (kbps)				Total
	Seg 1	Seg 2	Seg 3	Seg 4		Seg 1	Seg 2	Seg 3	Seg 4	
Scenario 1	92	92	92	92	368	90	90	90	90	360
Scenario 2	147.2	73.6	73.6	73.6	368	92	72	72	72	308
Scenario 3	52.57	105.14	105.14	105.14	368	50	92	92	92	326
Scenario 4	36.8	73.6	110.4	147.2	368	37	72	92	92	293
Unscheduled IEEE 802.11										
	Input Rate for segments (kbps)				Total	Output Rate for segments (kbps)				Total
	Seg 1	Seg 2	Seg 3	Seg 4		Seg 1	Seg 2	Seg 3	Seg 4	
Scenario 1	92	92	92	92	368	80	80	80	80	320
Scenario 2	147.2	73.6	73.6	73.6	368	103	68	65	63	299
Scenario 3	52.57	105.14	105.14	105.14	368	47	80	79	81	287
Scenario 4	36.8	73.6	110.4	147.2	368	37	62	78	101	278

Figure 4.24 shows the total throughput achieved by weighted scheduling, non-weighted scheduling, and unscheduled IEEE 802.11 for the four different scenarios. It is assumed that weighted assignment changes the slot assignment instantaneously based on the throughput requirement on the associated segments and achieves almost constant throughput at 360 kbps. The throughput achieved by non-weighted scheduling and unscheduled IEEE 802.11 is different for different scenarios.

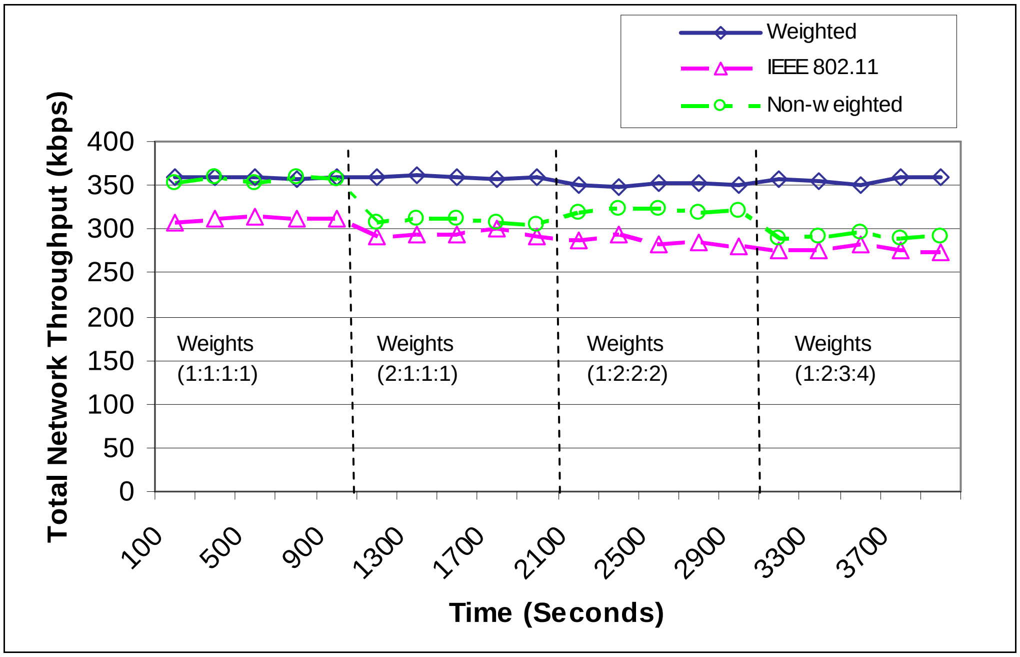


Figure 4.24: Total throughput achieved on all segments

4.4.6 Energy Efficiency

To simulate the amount of energy saved the same basic single intersection scenario is simulated with the aggressive without offset algorithm. The average number of slots allotted to the nodes is shown in Table 4.6. Fig. 4.25 shows the number of slots saved at different nodes when switching to sleep option (TMAC option) is used when there is no data available to exchange. The numbers of slots saved are out of a total of 6000 available slots for the entire simulation length. The nodes at 4 hops away from the controller saved the most number of slots as shown in Fig. 4.25. The ON time percentage for the nodes is the percentage of time the nodes are actively transmitting and receiving data. The ON time percentage is shown in Fig. 4.26. The 4 hop nodes are active for 75% of time without the TMAC option. However, with TMAC option being used they end up being active 30% of time when the input rate is close to the saturation limit. The one and two hop nodes use almost all their slots with the input rate near saturation and thus are active 24% of time.

Table 4.6: Number of assigned slots at different nodes

	# of assigned slots (sending, receiving)	Total slots	On time percentage without TMAC
1 Hop Nodes	50,50	400	25
2 Hop Nodes	50,50	400	25
3 Hop Nodes	50,150	400	50
4 Hop Nodes	150,150	400	75
5 Hop Nodes	150,0	400	37.5

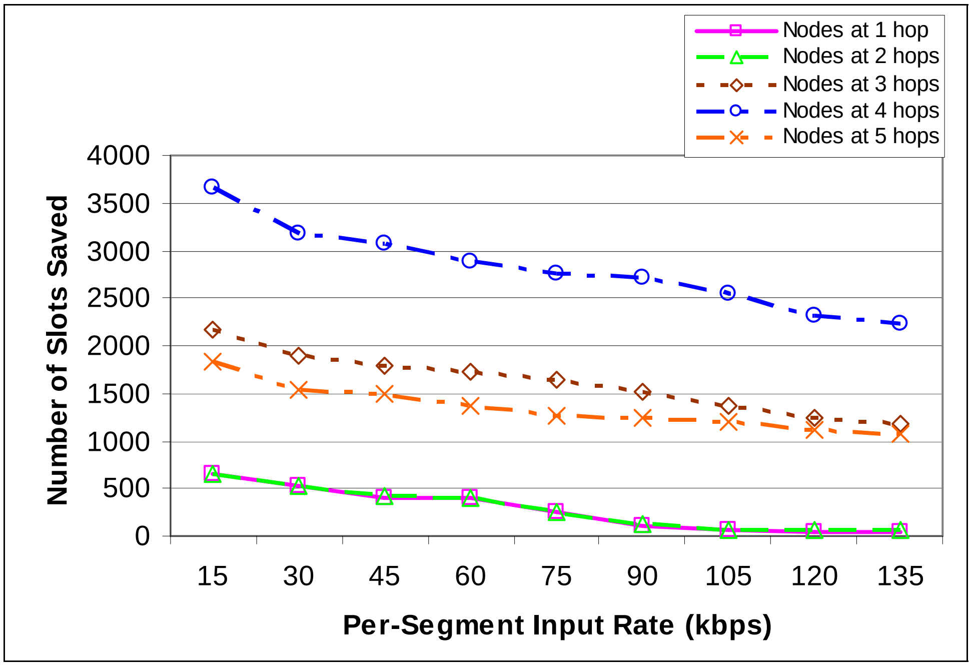


Figure 4.25: Energy efficiency (Number of saved slots)

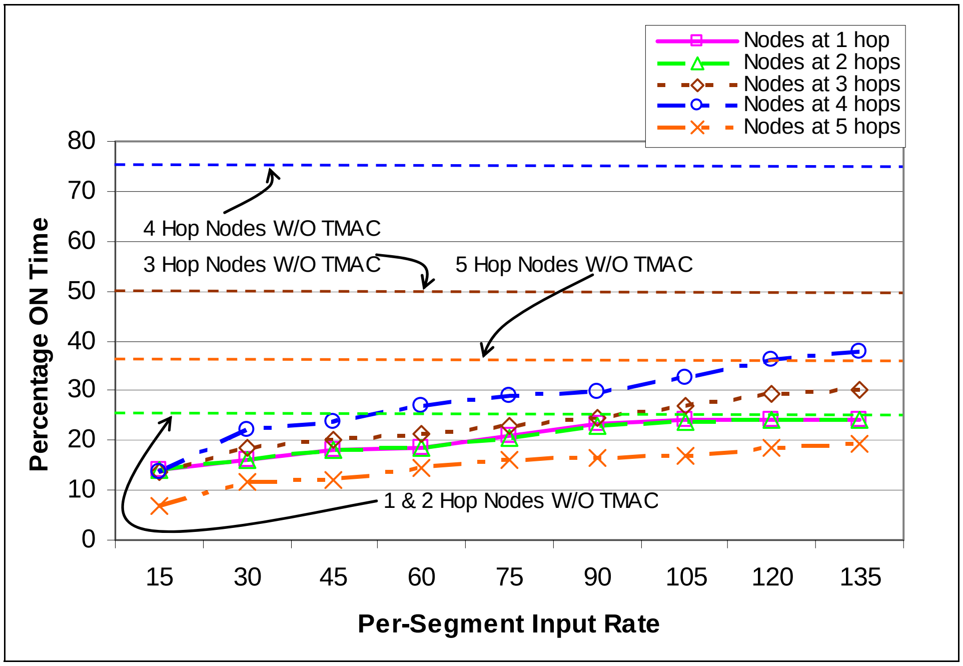


Figure 4.26: Energy efficiency with TMAC option

4.5 Conclusion

In this chapter the single intersection transmission scheduling is studied and the scheduling algorithm is explained. Fundamental concepts to lower delivery latency, increase network throughput and, increase energy efficiency are studied and incorporated.

Simulations are conducted with different network topologies to show that the bottleneck end-to-end throughput is independent of the cluster length and inversely proportional to the degree of conflict. The end-to-end throughput varies depending on the chosen algorithm and the non-aggressive, aggressive without offset and aggressive with offset algorithms are able to achieve a 35%, 51% and 63% increase over the end-to-end throughput achieved by unscheduled IEEE 802.11. The weighted assignment is shown to achieve better network throughput as compared to the non-weighted assignment as it divided the capacity among segments based on the communication needs.

It is shown that packet delivery latency is proportional to the slot length and this is confirmed with the help of simulations. It is also noticed that the end-to-end throughput is slightly affected by the slot length as longer slot lengths gave slightly higher end-to-end throughput.

The algorithm assigned slots such that high end-to-end throughput is achieved and thus every node was assigned high number of transmission slots. However, all the slots would not be used as there might not be data available at the node to be transmitted. During such opportunities the nodes saved their energy by switching to the sleep mode. Simulations supported this claim by showing that certain nodes ended up spending only 30% energy where they would have otherwise spent 75% energy.

CHAPTER 5

MULTIPLE INTERSECTION TRANSMISSION SCHEDULING

The single-intersection algorithm produces a conflict free schedule for nodes associated with a single intersection without considering any surrounding nodes. With multiple intersections in place, their independently discovered schedules are bound to have conflicts as the controllers do not know the topology information of the nodes belonging to clusters controlled by neighboring controllers. To resolve conflicts, controllers at nearby intersections must share their topology information and schedules to derive or negotiate for a mutually compatible schedule in a *centralized* or *distributed* manner. The details of the two approaches are given in the later sections.

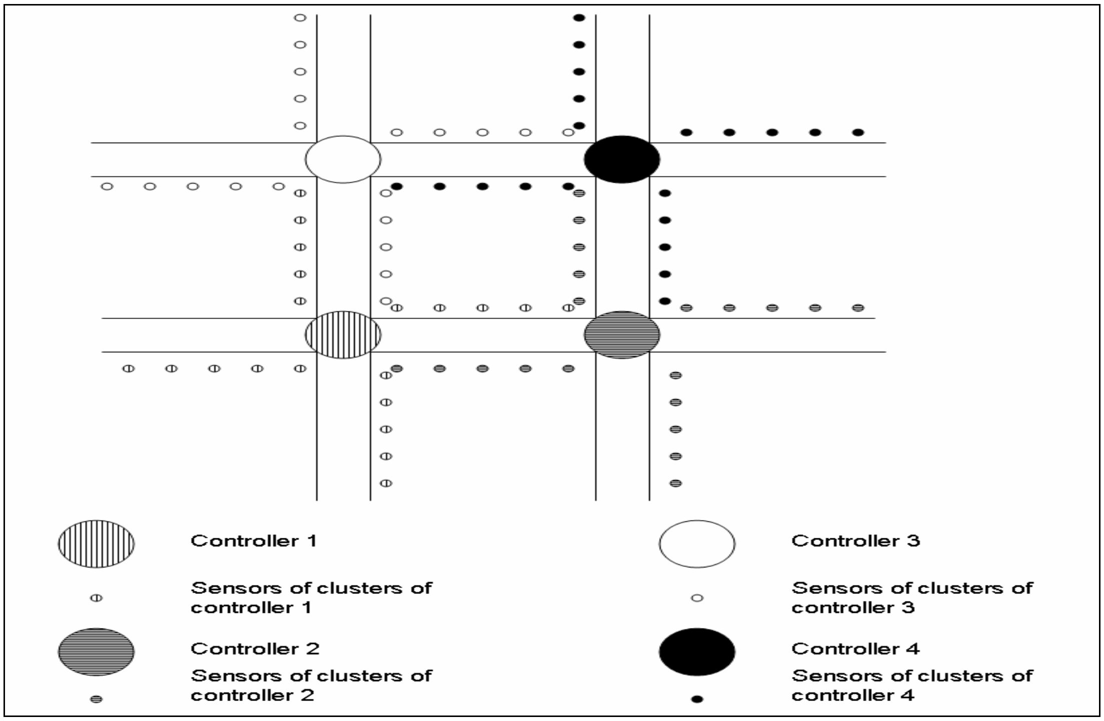

Controller 1
Sensors of clusters of controller 1
Controller 2
Sensors of clusters of controller 2
Controller 3
Sensors of clusters of controller 3
Controller 4
Sensors of clusters of controller 4

Figure 5.1: Controllers and sensors of multiple intersections.

5.1 Centralized Scheduling

The centralized approach collects the entire topology of all nodes in the set of concerned intersections at a node called the *leader*. The controllers collect all the topology information in the topology discovery phase and send this information to the leader. The leader constructs a *complete* conflict graph of all the nodes in the set of concerned intersections. The single-intersection algorithm described in Chapter 4 is applied to all the segments with the knowledge of the complete conflict graph to determine the schedules for all the nodes in the concerned intersections. In a hierarchical network, a level 2 controller can be a leader of multiple level 1 controllers that it oversees. Fig. 5.1 shows an example with 4 intersections. Let controller 1 be the chosen leader, then it collects topology information from all other controllers, executes the single-intersection algorithm considering 16 road segments, and distributes the schedule to the respective controllers. The controllers in turn distribute the schedules to the nodes in their clusters.

5.2 Distributed Scheduling

The distributed approach allows each intersection to individually determine its schedule and then iteratively negotiate its schedules with adjacent controllers to resolve any conflicts. For the distributed scheduling the process initiates at one controller which is called the *seed*; its associated clusters are denoted as the *seed clusters*. A higher level controller can be a natural candidate for the task. A seed controller iteratively requests its adjacent controllers to submit their schedules and their topology and it constructs an extended conflict graph using this information. Given the schedules and an extended conflict graph, the seed controller determines the new schedules as follows:

1. The seed controller compares its schedule with each adjacent controller's schedule one slot at a time to determine if in any slot conflicting senders have been scheduled. For each conflict found, among the conflicting nodes one of the two conflicting nodes will give up the slot. There can be different approaches adopted to decide which node gives up the slot.

Throughput Maximizing Heuristic

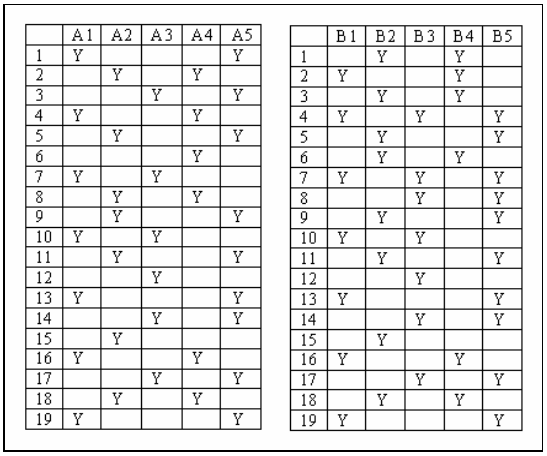
The node in the seed cluster gives up the slot, until it becomes a *bottleneck node*. A bottleneck node is the node with the least number of slots on its segment. The achievable end-to-end throughput, i.e., the *bottleneck throughput*, along a particular segment is limited by the number of slots allotted to the bottleneck node. Nodes with more slots than the bottleneck node can give up the excessive slots without degrading the bottleneck throughput. If the node in the seed cluster has remaining slots equal to the bottleneck, than the node in the adjacent cluster must give up the slot. If both nodes have reached their bottleneck slots, then the node belonging to the seed cluster will give up the slot. It is seen that this approach tries to ensure that the nodes not belonging to the seed cluster save most of their slots which are later on given up when their cluster becomes the seed cluster thereby achieving high throughput.

Fairness Heuristic

The node with more *additional slots* gives up the slot. *Additional slots* are defined as the number of slots in excess of the slots available to the bottleneck node.

2. The extended conflict graph also reveals conflicts between any two of the seed's adjacent intersections, which may not be directly adjacent to each other. For example in Fig. 5.1, controllers 2 and 4 are both adjacent to controller 1 but not each other. Controllers 2 and 4 later on become the seed controllers and negotiate their schedules with their adjacent controllers but since they are not adjacent to each other they do not get a chance to negotiate the inter-cluster conflicts in their schedules. Controller 1 has the opportunity to solve for their conflicts with its extended conflict graph. The seed hence compares the schedules of its adjacent intersections and resolves any conflicts with a heuristic approach. The fairness heuristic is used to determine which node will give up the slot in case of conflicts.
3. The seed concludes by distributing all schedule changes to the corresponding controllers.

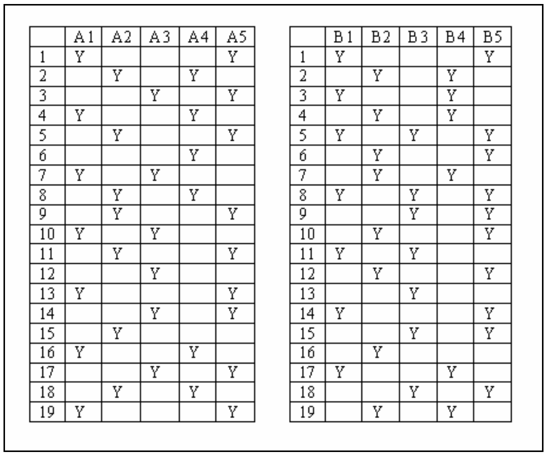
Controllers iteratively assume the role of a seed, until all controllers have performed the three-step procedure once. The quality of the conflict-free schedule resulting from the distributed algorithm, in terms of per-segment end-to-end throughput, is dependent on the initial schedules produced at each individual intersection. The temporal relation between the allotted slots may contribute positively or negatively to the eventual negotiated schedule. The achievable bandwidth can vary with slight changes to the slots, such as shifting the schedule of all nodes in one segment by a same number of slots. An example is given in Fig. 5.2 to illustrate such opportunities.



	A1	A2	A3	A4	A5
1	Y				Y
2		Y		Y	
3			Y		Y
4	Y			Y	
5		Y			Y
6				Y	
7	Y		Y		
8		Y		Y	
9		Y			Y
10	Y		Y		
11		Y			Y
12			Y		
13	Y				Y
14			Y		Y
15		Y			
16	Y			Y	
17			Y		Y
18		Y		Y	
19	Y				Y

	B1	B2	B3	B4	B5
1		Y		Y	
2	Y			Y	
3		Y		Y	
4	Y		Y		Y
5		Y			Y
6		Y		Y	
7	Y		Y		Y
8			Y		Y
9		Y			Y
10	Y		Y		
11		Y			Y
12			Y		
13	Y				Y
14			Y		Y
15		Y			
16	Y			Y	
17			Y		Y
18		Y		Y	
19	Y				Y

Fig. 5.2: Distributed without sliding



	A1	A2	A3	A4	A5
1	Y				Y
2		Y		Y	
3			Y		Y
4	Y			Y	
5		Y			Y
6				Y	
7	Y		Y		
8		Y		Y	
9		Y			Y
10	Y		Y		
11		Y			Y
12			Y		
13	Y				Y
14			Y		Y
15		Y			
16	Y			Y	
17			Y		Y
18		Y		Y	
19	Y				Y

	B1	B2	B3	B4	B5
1	Y				Y
2		Y		Y	
3	Y			Y	
4		Y		Y	
5	Y		Y		Y
6		Y			Y
7		Y		Y	
8	Y		Y		Y
9			Y		Y
10		Y			Y
11	Y		Y		
12		Y			Y
13			Y		
14	Y				Y
15			Y		Y
16		Y			
17	Y			Y	
18			Y		Y
19		Y		Y	

Figure 5.3: Distributed with sliding

Consider that A1, A2, …, A5 are part of cluster A and B1, B2, …, B5 are part of cluster B. The controllers of A and B have allotted the local schedule as shown in Fig. 5.2. The controllers are unaware that A1 and B1 are conflicting with each other. When the distributed algorithm is run over the slots it is found that A1 and B1 end up conflicting in slots 4, 7, 10, 13, 16 and 19. Thus either of them ends up giving up slots. However, it can be seen that if the cluster B slides its schedules by one slot then the conflicts are avoided as shown in Fig. 5.3. It can be seen that the conflict occurring at slots 4, 7, 10, 13, 16 and 19 have been resolved at the cost of a new conflict in slot 1. Therefore sliding schedules can help in resolving conflicts.

If we keep the schedule of one cluster fixed and slide the schedule of the other cluster one slot at a time for *g* times, where g is the cycle length then several possible schedules are evaluated. The best schedule is selected as the version which gives the least conflicts and thereby allowing more slots to be allotted to every bottleneck node. However, it is important that we select a schedule such that the conflicts affecting the bottleneck nodes are removed and thus we can achieve a higher end-to-end throughput for all the segments. In case of large cycle lengths sliding the entire schedule might incur a lot of computational overhead. There is a trade-off between the computational overhead for sliding and the available throughput. It is observed that when *g* is selected to be 10% of the cycle length the achievable throughput is close to that achieved with *g* equal to 100% of cycle length. This version of distributed protocol with *g* selected to be 10% of cycle length is called 10% distributed and it involves lower computation as compared to 100% distributed.

5.3 Communication Costs

The centralized algorithm can provide conflict free schedules for all the nodes with the help of the complete conflict graph and the scheduling algorithm. The distributed algorithm also provides a conflict free schedule but the available throughput is not as high as the centralized algorithm. The disadvantage of the centralized algorithm is that it requires the complete graph and thus for any change in the network the entire algorithm has to be rerun and this incurs a lot of communication cost. An analysis of the communication costs of the centralized and distributed algorithms is given below:

Centralized Algorithm

Dependent on the size of the network, the centralized approach is not always more costly than the distributed. The communications incurred in the centralized approach include:

A. The leader floods an announcement to all subsidiary controllers requesting them to submit their clusters topology information.

B. Each controller unicasts its topology to the leader

C. The leader distributes the final schedule to corresponding controllers

D. Each controller distributes the schedule to subsidiary sensors.

We can consider a tree consisting of all the controllers as nodes and the elected leader as the root. The topology is such that every controller has four neighboring controllers and thus the tree is such that each node in the tree has four branches. An approximate estimation considers the cost of steps (A) and (D) much less than steps (B) and (C) as steps (A) and (D) are message that do not carry the information of the complete topology of the cluster. Steps (B) and (C) are symmetric and both trace a complete spanning tree. Assuming each node has 4 downstream branches, the cost to traverse the tree one way is

$$4^{h} \cdot h + 4^{h-1}\left(h-1\right) + \ldots + 4 \cdot 1$$

where, *h* is the depth of the tree and the cost to relay a message between two adjacent controllers is 1. Assuming a complete spanning tree and depth *h* of the tree we can calculate the number of nodes in the tree to be *N,*

$$N = 1 + 4 + 4^{2} + \ldots + 4^{h} = \left(4^{h+1} - 1\right) / \left(4 - 1\right)$$

solving for h,

$$h = \log_{4}\left(3N + 1\right).$$

Thus the total cost for the centralized algorithm is mainly dominated by (2) and (3) and for a network of N intersection controllers it is:

$$\text{cost}_{CENT} = 2 *[\ 4^{h} \cdot h + 4^{h-1}\left(h-1\right) + \ldots + 4 \cdot 1\ ] \qquad (5.1)$$

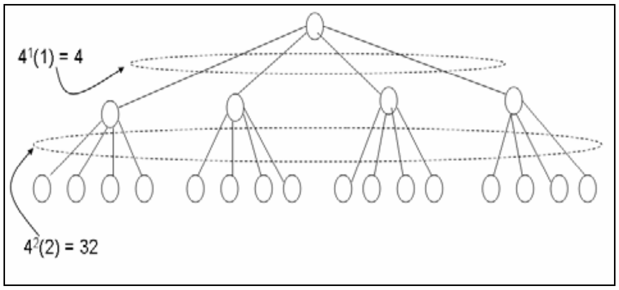


Figure 5.4: Cost computation

Distributed Algorithm

The distributed approach incurs the following communications:

A. The seed requests schedules and topology from each adjacent intersection controller

B. The adjacent controllers send the schedule to the seed

C. The seed sends final schedule (or changes) to each adjacent controller

D. Steps (A), (B) and (C) repeats for each controller.

Assuming cost of step (A) as negligible and steps (B) and (C) as symmetric, the cost for the negotiation initiated by each controller is 4+4=8, and the total cost for a network of *N* controllers with a tree of depth *h* is 8N. Hence the total cost for the distributed algorithm is mainly dominated by steps (B) and (C) and for a network of N controllers it is:

$$\text{cost}_{DIST} = 8N \tag{5.2}$$

Equations 5.1 and 5.2 were derived assuming schedules of all nodes in the network must be changed. Such changes are not necessary for a small change in some particular cluster. With distributed scheduling, the nearby clusters can accommodate such changes with slight modifications to their existing schedules and this helps in reducing the cost for communication.

5.4 Simulation Studies

To study the effect of centralized and distributed algorithms over a simple network, multiple intersection scenario is simulated.

5.4.1 Simulation Settings

To analyze the performance of our scheduling algorithm in a multiple intersection scenario, the scenario as shown in Fig. 5.1 consisting of four complete intersections is studied. However, the nodes belonging to different intersections are within conflicting range of each other. This multiple intersection scenario is the *basic multiple intersection scenario* that is used for the simulations. The simulations consider 2 variations of the distributed algorithm and 10% distributed. It was observed that sliding the slots improves the throughput. Sliding involves computational overhead and thus experiments were conducted to observe the improvement in throughput possible by sliding the schedules by different percentages of the schedule length. It was noticed that the best available throughput was mostly available by just sliding the slots by 10% as several combinations were considered by sliding the slots by 10% and thus 10% distributed version was selected for running the experiments.

5.4.2 Performance metrics for different algorithms

It can be seen in the Fig. 5.1 that the available bandwidth was divided between 12 nodes as the degree of conflict was 12. As seen in Fig. 4.15 the saturation throughput per segment should be 60 kbps for a conflict degree of 12. The centralized algorithm achieved a saturation throughput of 58 kbps. The distributed algorithm achieved a throughput of 46 kbps and the 10% distributed algorithm achieved a throughput of 48 kbps. The throughput achieved by the distributed algorithms as expected was much lower than that achieved by the centralized algorithm. The per segment saturation throughput achieved by unscheduled IEEE 802.11 is approximately 42 kbps.

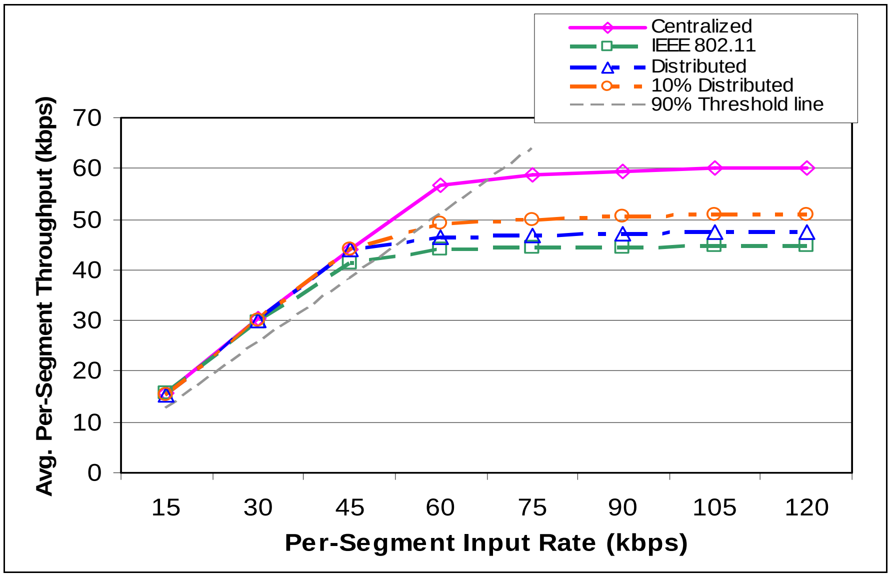


Figure 5.5: Throughput for different algorithms for multiple intersection scenario

It is important to notice that it is possible to have a hybrid solution combining the advantages of both distributed and centralized algorithms that can initially use the centralized algorithm to allot the schedules and switch to distributed algorithms for adjusting to small changes in the networks. This will reduce communication costs as well as achieve high overall network throughput.

The delay analysis shows that both the centralized and distributed algorithms have a small end-to-end delay at low input rates. The end-to-end delay starts increasing as the input rate starts getting closer to the saturation throughput supported by that particular algorithm. When the input rate is lesser than the saturation throughput it is noticed that distributed algorithm has a higher end-to-end delay as compared to centralized algorithm. This is because the distributed algorithm gives up slots from the original schedules in case of conflicts following a throughput maximizing heuristic. Thus the slot assignment which was done to minimize the delay is disturbed leading to higher end-to-end delay.

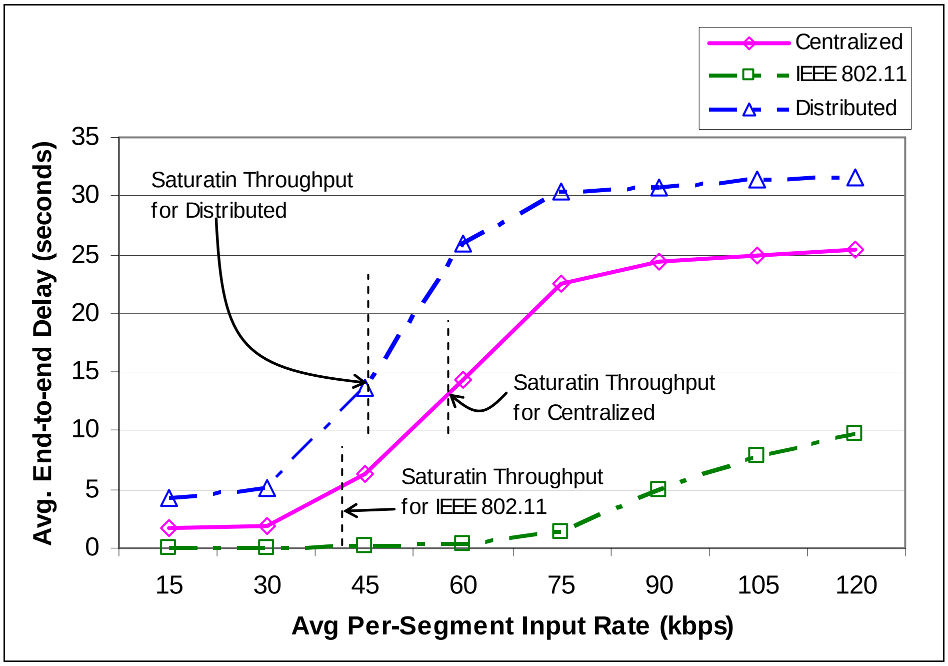


Figure 5.6: End-to-end delay for different algorithms for multiple intersection scenario

5.4.3 Performance metrics for different slot lengths

The schedule obtained from the centralized algorithm was used to run experiments with varying slot lengths. Fig. 5.7 shows that the throughput for different slot lengths is almost the same and only varies slightly as explained for the single intersection scenario. With the slot length of 0.05 seconds the saturation throughput achieved is 57 kbps and for a slot length of 0.4 seconds the saturation throughput achieved is 60 kbps. Fig. 5.8 shows that the end-to-end delay for the schedule obtained by centralized algorithm with different slot lengths is directly proportional to slot length. For a slot length of 0.05 seconds the end-to-end delay is approximately 0.8 seconds and for a slot length of 0.1 seconds it increases to 1.6 seconds. For a slot length of 0.2 seconds and 0.4 seconds the end-to-end delays are 3.2 and 6.4 seconds respectively.

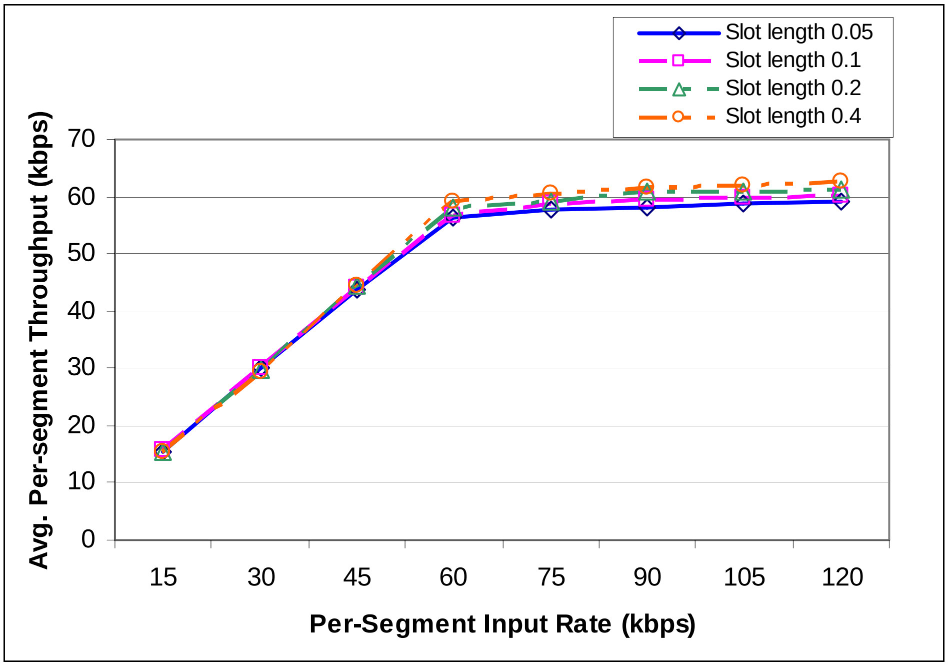


Figure 5.7: Throughput for different slot length for multiple intersection scenario

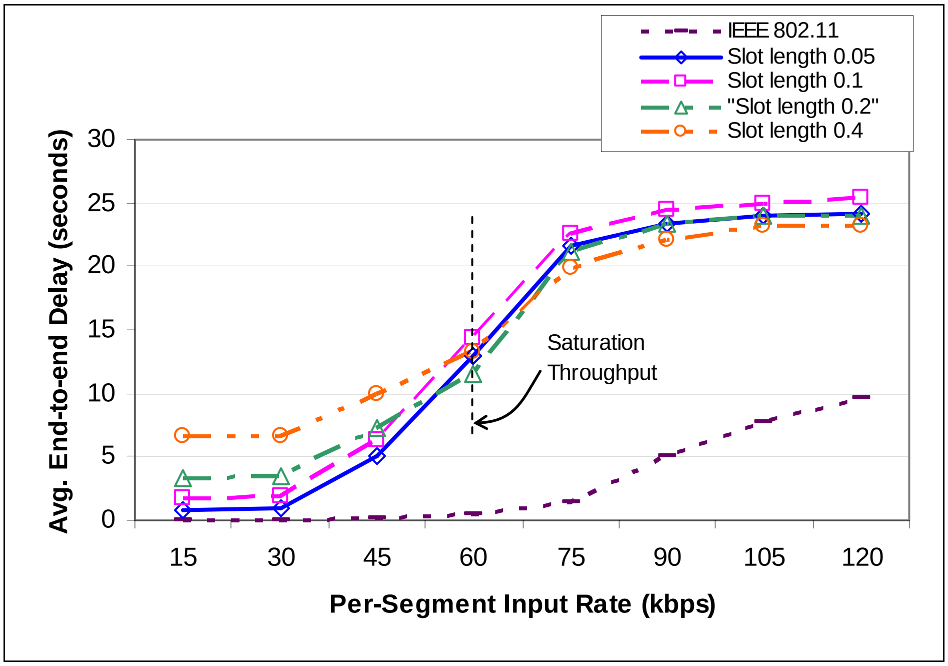


Figure 5.8: End-to-end delay for different slot length for multiple intersection scenario

5.4.4 Performance metrics for topology changes

Fig. 5.9 and 5.10 represent multiple intersection scenarios that are similar to the scenario shown in Fig. 5.1 with slight variations. Fig. 5.9 and Fig. 5.10 have 10 sensors and 15 sensors along four segments that initially had 5 sensors for the scenario shown in Fig. 5.1. The three scenarios vary in the network topology as they have different number of sensors on the 4 segments. However, after running the scheduling algorithm the bottleneck nodes are assigned 66 slots out of a cycle of 800 slots for all the three scenarios. Thus it can be concluded that the end-to-end throughput is only limited by the number of slots available to bottleneck nodes and is independent of the number of sensors along the highways. Fig. 5.11 shows the throughput achieved for the centralized algorithm for the three scenarios is approximately the same. Scenario 1 (5 sensors), Scenario 2 (10 sensors), Scenario 3 (15 sensors) are the three scenarios simulated.

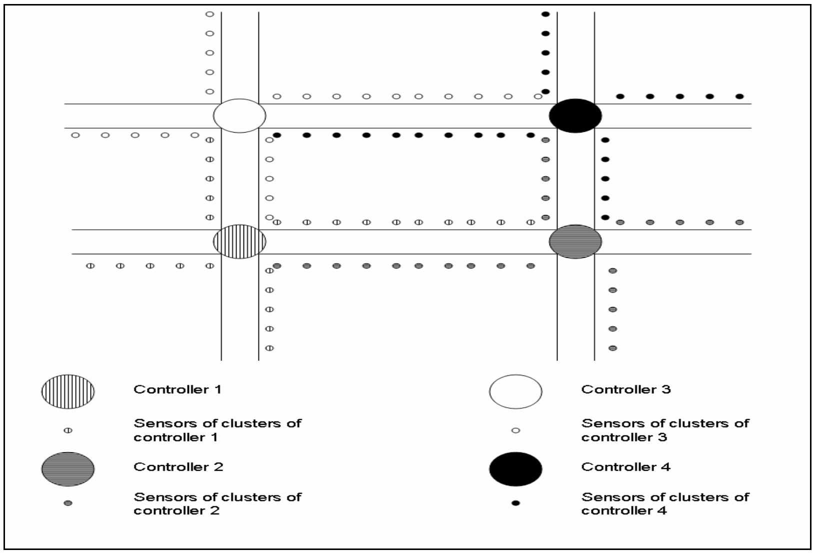

Controller 1
Sensors of clusters of controller 1
Controller 2
Sensors of clusters of controller 2
Controller 3
Sensors of clusters of controller 3
Controller 4
Sensors of clusters of controller 4

Figure 5.9: Multiple intersection scenario – 10 segment scenario – scenario 2

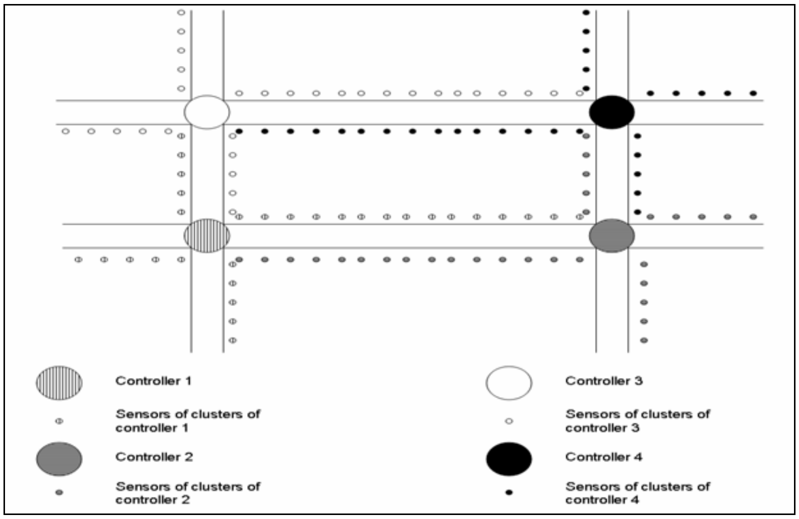

Controller 1
Sensors of clusters of controller 1
Controller 2
Sensors of clusters of controller 2
Controller 3
Sensors of clusters of controller 3
Controller 4
Sensors of clusters of controller 4

Figure 5.10: Multiple intersection scenario – 15 segment scenario – scenario 3

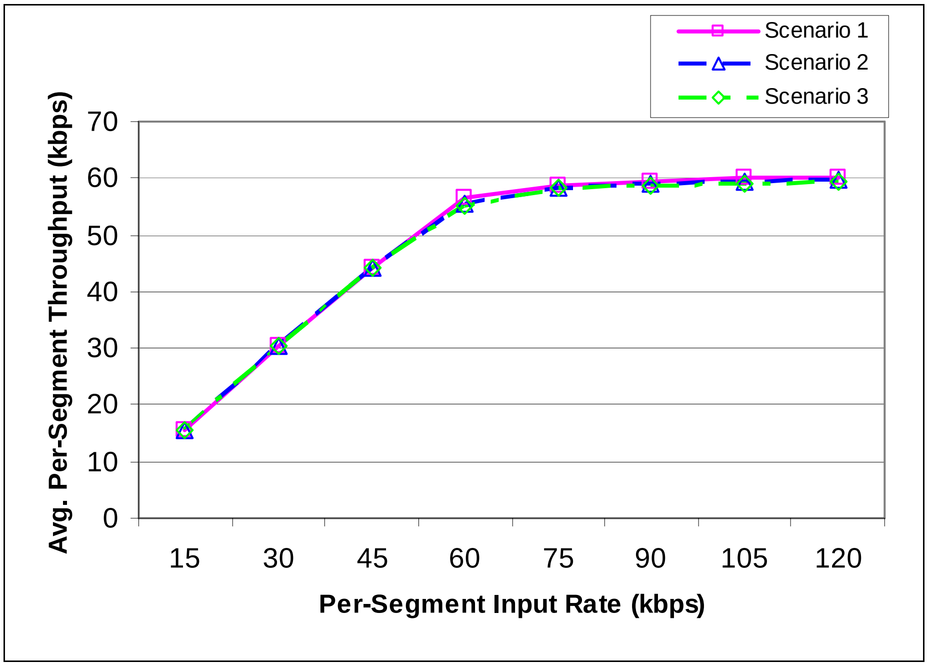


Fig. 5.11: Throughput for different multiple intersection scenarios

It is important to notice that because there is a bottleneck on both ends of the segments the nodes in between the bottleneck nodes also end up getting fewer numbers of slots in spite of them not being in high conflict region. This happens because we schedule any parent node in the slot immediately after its child. Since the end member of a cluster is also a bottleneck node it is assigned fewer slots according to the degree of conflict and this leads to the parent nodes being assigned fewer slots.

5.5 Conclusion

This chapter proposes different techniques that can be adopted to produce conflict free schedules that can spread across multiple intersections. It proposes two approaches *centralized* and *distributed* to resolve the conflicts between nodes belonging to different intersection controllers. The centralized algorithm involves high communication costs as compared to the distributed algorithm in case of large networks. However, it is observed that the centralized algorithm achieved better throughput performance as compared the distributed algorithm because it had all the necessary information for constructing the complete conflict graph and assigning conflict free schedules. Simulations are conducted for a multiple intersection scenario to show that centralized assignment strategy achieved 25% higher throughput as compared to distributed assignment strategy. It is important to note, for small changes in the network it is cost efficient to solve the conflicts in a distributed manner whereby a few controllers exchange their schedules and resolve conflicts without needing to change the schedules of nodes belonging to any other cluster. Thus it can be concluded that it is a good idea to start with a centralized assignment and make the network adapt to local topology changes distributively.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis a transmission scheduling solution for highway traffic monitoring wireless sensor network is proposed.

6.1 Conclusions

Highway traffic monitoring has been done centrally and this raises a bottleneck as all the traffic management applications require communication with the TMC. A distributed system can enhance scalability. Based on the hierarchical architecture and addressing scheme previously proposed this thesis describes a practical routing scheme, based on which the transmission scheduling problem on such networks is studied.

The wireless sensor networks have limited lifetime as the sensors are battery powered. The thesis proposes a scheduling solution that provides for improving energy efficiency, network throughput, and packet delivery latency. The scheduling solution uses the concepts of sleep-awake cycle, pipelining and avoiding sleep latency (adjacent nodes assigned nearest slots) to provide for energy efficiency, high network throughput, and low delivery latency. The energy efficiency is further increased by making the nodes switch to sleep mode during scheduled slots when there is no data to be exchanged.

The thesis also proposes certain variations in the algorithm that can help increase the throughput. It explains 3 different types of algorithms: non-aggressive, aggressive without offset, and aggressive with offset which provide 35%, 53% and 61% increase in end-to-end throughput as compared to unscheduled IEEE 802.11. The impact of changing the slot length on the performance metrics is studied. It is seen with simulations that the end-to-end delay is proportional to the slot length and the end-to-end throughput is approximately the same for different slot lengths. The weighted assignment strategy helps divide the available bandwidth between different clusters according to their respective traffic loads, with which the achieved overall network throughput is much higher than a non-weighted assignment strategy.

For multiple intersections, the thesis proposes centralized and distributed approaches to achieve conflict free schedules. The communication costs and performance metrics for the two approaches are analyzed and compared with simulations. The distributed approach is more cost-effective in large networks with few changes, though it achieves less throughput as compared to the centralized approach.

6.2 Future Work

The hierarchical routing scheme routes packet along the roads and the controller hierarchy to deliver it to the packet destination. The paths identified are not necessarily the shortest path and can have path lengths far above the shortest path. Optimization of the routes is possible by nodes actively collecting or passively overhearing topology information of nearby clusters, such that a shorter path can be used for routing packets to nearby destinations.

Our proposed slot assignment strategy reduces end-to-end delay on a segment in a particular direction. It is seen in the multiple intersection scenario that there are unused slots which can be reassigned to nodes in the low conflict region. These slots can be assigned to the nodes to reduce the message latency in the opposite direction. The proper usage of these slots needs a detailed study of the type of applications that are to be supported by the system.

Support of real-time communication in such networks has not been addressed as the algorithm tries to reduce the delay but does not guarantee to meet hard deadlines for routing packets. Support for real time communication needs to be studied in much detail and new concepts need to be incorporated into the scheduling solution in order for it to meet the deadlines provided for real time communication.

The scalability of the hierarchical routing protocol depends on the traffic patterns and the cluster organization. The scenarios studied in this thesis are still limited in scope. Formal validation of the scalability remains to be conducted. Realistic scenarios with large networks involving distributed operations should be studied for accurate analysis of the routing and scheduling solution proposed.

APPENDIX

The detailed routing procedure is explained in this section. The complete procedure for local topology discovery, cluster formation and data routing are explained in this section.

A.1 Message Format

Each packet has a common header with routing specific or data specific headers. Fig. A.1 shows the different packet headers.

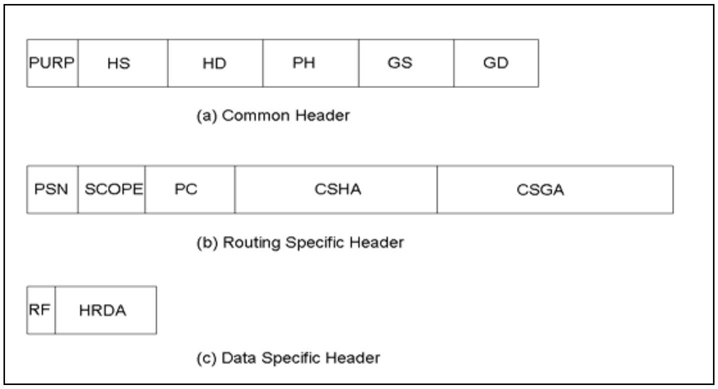


Figure A.1: Packet header

The common header fields are defined below:

PURP (Purpose): A 8 bit field indicating the type of information carried by the packet. The different types of messages are topology request (TOPO_REQ), topology response (TOPO_RESP), cluster announcement (CLUS_ANN), cluster discovery (CLUS_DISC) and data packet (DATA).

HS (Hierarchical Source Address): A 88 bit field indicating the hierarchical address of the source

HD (Hierarchical Destination Address): A 88 bit field indicating the hierarchical address of the destination

PH (Previous Hop): A 88 bit field indicating the previous hop forwarding the message

GS (Geographic co-ordinates of Source): A *k* bit field indicating the geographic(X, Y) co-ordinates of the source

GD (Geographic co-ordinates of Destination): A *k* bit field indicating the geographic(X,Y) co-ordinates of the destination

The routing specific header fields are defined below:

PSN (Packet Sequence Number): A 8 bit field indicating the packet sequence number

SCOPE: A *k* bit field indicating the geographical scope of the packet

PC (Previous Controller): A 88 bit field indicating the previous controller forwarding the message

CSHA (Cluster Sensor Hierarchical Address): A *n*88* bit field indicating the address of the sensors belonging to the cluster. Here n is the number of sensors in the cluster. Sensors append their hierarchical addresses in this field.

CSGA (Cluster Sensor Geographical co-ordinates): A *n*k bit* field indicating the geographical co-ordinates of the sensors belonging to the cluster. Sensors append their geographical addresses in this field.

The data specific header fields are defined below:

RF (Redirection Flag): A 1 bit field which when set indicates that the packet is redirected

HRDA (Hierarchical Redirected Destination Address): A 88 bit field that carries the hierarchical address of the redirected destination which is the destination of the packet when the packet is redirected

A.2 Local Topology Discovery

Upon reboot, each node is aware of its preprogrammed address. Based on the level field of the address, each node determines its logical function (as a sensor or controller). To establish local network connectivity and form local clusters, the *local topology discovery* phase is performed as follows. The local topology discovery algorithm is given in Fig. A.2.

Upon reboot, a controller starts periodically broadcasting a *topology request* message with increasing packet sequence number. This message is received by surrounding sensors and the sensors along the associated segments of the packet initiating controller record the packet sequence number. The sensors next broadcast the message if the packet sequence number is higher than the earlier recorded packet sequence number. The sensors determine whether they are on the associated segments by comparing the Mileage and DIR field of its address with that of the message initiating controller. The sensor is along the associated segment if its DIR field is ‘1’ and it is at a higher mileage than the packet initiating controller or its DIR field is ‘2’ and it is at a lower mileage then the packet initiating controller. The sensors on the associated segment will hear the topology request packet from several nearby nodes among which it will record the nearest sensor on the associated segment towards the packet initiating controller as it *parent*. The sensor determines its nearest sensor by checking the mileage field of the address of the previous sensor forwarding the packet. The process continues until a controller on the other end of the segment, namely the end controller, receives the packet. Fig. A.3 explains the flow of topology request message.

Upon receiving the topology request message the end controller recognizes the nearest sensor towards the packet initiating controller as its next hop to reach the initiating controller and sends a *topology response* message to the initiating controller. The unicast topology response message is relayed by each sensor to its parent until it reaches the initiating controller. The sensors while relaying the topology response message record the closest neighbor towards the end controller as their *child*. The initiating controller upon receiving the topology response message recognizes the nearest sensor towards the end controller on its associated segment as its *child*. Each node is assumed to be aware of its geographical co-ordinates and while relaying the topology response message appends its geographical co-ordinates and hierarchical address information in the packets CSGA and CSHA fields respectively.

The topology request message is transmitted periodically with increasing packet sequence number for certain time duration, called the *topology discovery time,* as some nodes on the initiating controllers associated segment might not receive the broadcast message because of packet collision and thereby do not register with the cluster.

```
N is the set of hierarchical addresses of a node.
For incoming packet P at node N:
if N is a controller
    if P is a topology request
        if P originates from N
            P.forwarder = N
            P.source = N
            Broadcast P
        else
            if P.forwarder is closer to N than N.next_hop
                Neighbor_cont = P.source
                Neighbor_cont.next_hop = P.forwarder
            endif
            construct topology response packet R
            R.source = N
            R.destination = N.next_hop
            R.forwarder = N
            Store N and its geographic coordinates in R
            Unicast R to R.destination
        endif
    endif
    if P is a topology response
        if P.destination == N
            N.child = P.forwarder
            Local topology discovery completed, proceed to slot assignment phase
        endif
    endif
endif

if N is a sensor
    if P is a topology request
        if ((N.RID == P.forwarder.RID) && (P.forwarder resides between N and P.source))
            if((N.DIR == '1' && N.MIL > P.source.MIL) ||   (N.DIR == '2' && N.MIL < P.source.MIL))
                if(P.forwarder is closer to N than N.parent)
                    N.parent = P.forwarder
                    P.forwarder = N
                endif
                if (P.PSN > recorded.PSN)
                    Broadcast P
                    recorded.PSN = P.PSN
                endif
            endif
    endif
    if P is a topology response
        N.child = P.forwarder
        P.forwarder = N
        Store N and its geographic coordinates in P
        Unicast P to N.parent
    endif
endif
```

Figure A.2: Local topology discovery algorithm

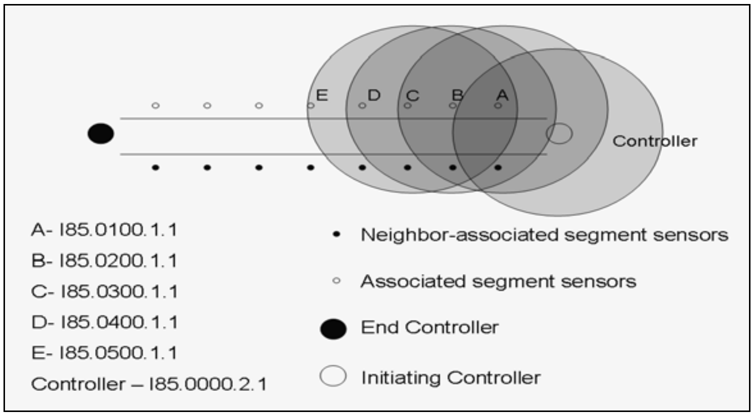


Figure A.3: Topology request message propagation

The topology discovery phase concludes when the topology discovery time has elapsed and at the end of this phase the initiating controller knows the hierarchical addresses and geographical locations of all nodes along its associated segments.

A.3 Hierarchical Cluster Formation

The controllers identify the complete cluster structure and build their hierarchical routing tables in the hierarchical cluster formation phase.

Type of messages & Routing

For sensors:

The sensors forward the messages to its parents if they have been forwarded to it by its child and vice versa. While forwarding they update the PREV_HOP field to its hierarchical address. If the packet is destined for that particular sensor then it will send it to the application layer.

For controllers

1. Cluster Announcement: A controller of level *i>2,* floods a route specific message of with the purpose field CLUS_ANN along all its neighbor-associated segments. The controller specifies its own level, scope, and populates the HA and PC field with its hierarchical address. Each controller receiving the message updates the PC field to be its hierarchical address before forwarding the packet. All controllers of level *i-1,* that have not registered with a higher level cluster will record the source controller as its parent controller and register to the cluster formed by its parent controller. Duplicate copies of the packet received at a controller from its associated segments can be identified using the PSN field and are discarded. The scope of the packet is a circular region with the source controller at the centre of the circle and the radius specified in the scope field of the packet. The process concludes when the packet has exceeded its scope. Fig. A.4 illustrates the handling of a CLUS_ANN message.
2. Cluster Join: On receiving the CLUS_ANN message each controller of level *i-1* sends a CLUS_JOIN message to its parent. The controller puts its hierarchical and geographical addresses and its parent hierarchical and geographical addresses in the HS, GS, HD and GD fields, respectively. The CLUS_JOIN message is forwarded like a CLUS_ANN message with its scope being limited using similar protocol to location aided routing [32]. The scope of the packet is exceeded when the packet has traveled more than *D* meters away from the X and Y co-ordinates range of the source and destination. The process concludes when the packet reaches the parent controller. Fig. A.5 explains the message flow.

The CLUS_ANN and CLUS_JOIN process is repeated iteratively by controllers at different levels until the highest level controller is reached and a hierarchical tree structure is formed.

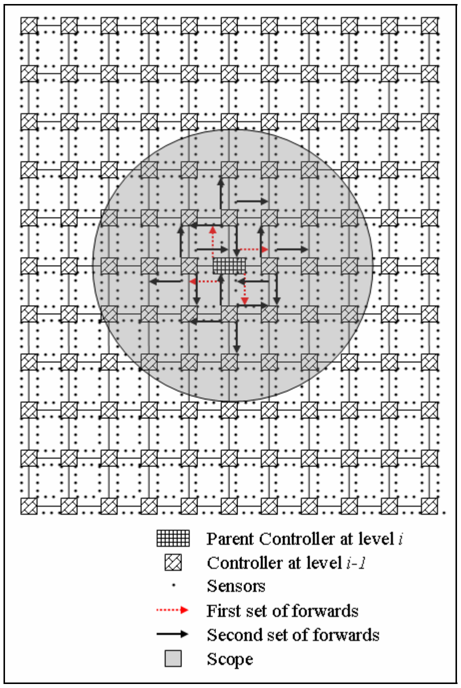


Figure A.4: Cluster announcement

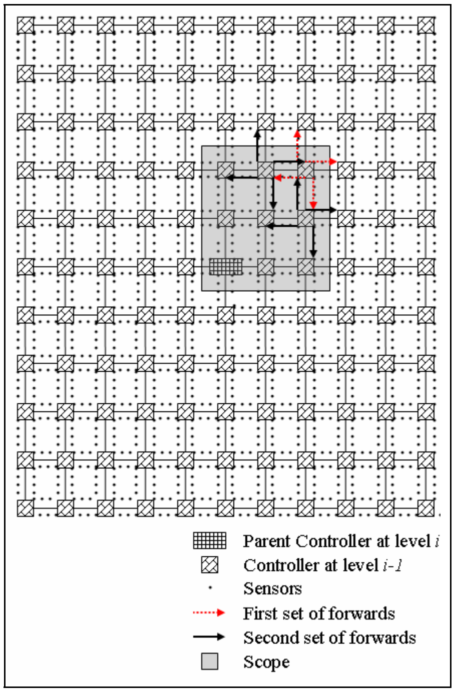
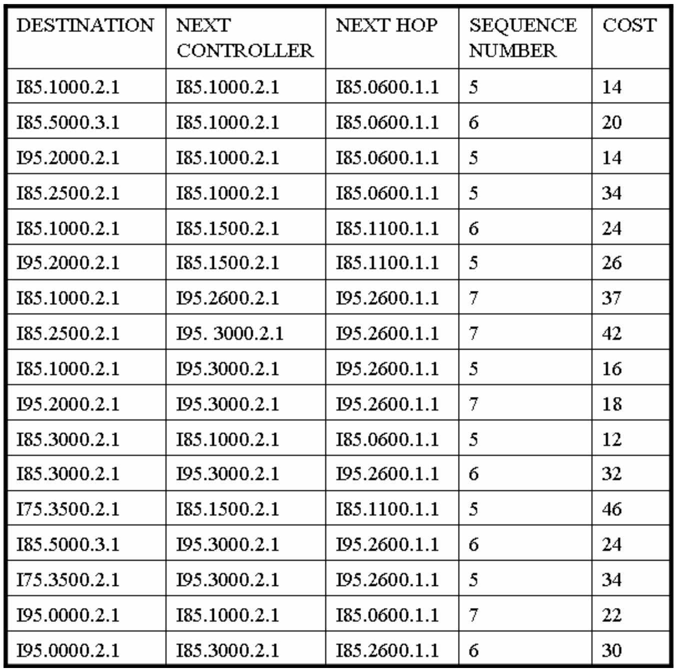


Figure A.5: Cluster join

As the CLUS_ANN and CLUS_JOIN messages are forwarded every intermediate controller receiving the packet will record an entry in the routing table, where the DESTINATION is the packet initiating controller, NEXT CONTROLLER is the previous controller (PC) that has forwarded the message and NEXT HOP is the next hop on its neighbor-associated segment connecting it to the previous controller that has forwarded the message. An example of a segment of the routing table at the controller with addresses I85.0500.1.1 and I95.2500.1.1 is shown in Table A.1. The controller stores the cost to reach each controller from which it has heard the CLUS_ANN or CLUS_JOIN message. Multiple entries can exist for the same destination and the least cost path will be used for routing the packet. In scenarios where the least cost path fails to route the packet, alternative routes are available from the table for routing the packet.

Table A.1: Segment of routing table at controller



DESTINATION	NEXT CONTROLLER	NEXT HOP	SEQUENCE NUMBER	COST
I85.1000.2.1	I85.1000.2.1	I85.0600.1.1	5	14
I85.5000.3.1	I85.1000.2.1	I85.0600.1.1	6	20
I95.2000.2.1	I85.1000.2.1	I85.0600.1.1	5	14
I85.2500.2.1	I85.1000.2.1	I85.0600.1.1	5	34
I85.1000.2.1	I85.1500.2.1	I85.1100.1.1	6	24
I95.2000.2.1	I85.1500.2.1	I85.1100.1.1	5	26
I85.1000.2.1	I95.2600.2.1	I95.2600.1.1	7	37
I85.2500.2.1	I95. 3000.2.1	I95.2600.1.1	7	42
I85.1000.2.1	I95.3000.2.1	I95.2600.1.1	5	16
I95.2000.2.1	I95.3000.2.1	I95.2600.1.1	7	18
I85.3000.2.1	I85.1000.2.1	I85.0600.1.1	5	12
I85.3000.2.1	I95.3000.2.1	I95.2600.1.1	6	32
I75.3500.2.1	I85.1500.2.1	I85.1100.1.1	5	46
I85.5000.3.1	I95.3000.2.1	I95.2600.1.1	6	24
I75.3500.2.1	I95.3000.2.1	I95.2600.1.1	5	34
I95.0000.2.1	I85.1000.2.1	I85.0600.1.1	7	22
I95.0000.2.1	I85.3000.2.1	I85.2600.1.1	6	30

A.4 Packet Routing Algorithm

<u>*For sensors*</u>

Packet routing at the sensors remains same as explained in section A.3. If the packet is originated at the sensor then it sends the packet to its *presiding* controller. The presiding controller of a sensor is the controller managing the cluster to which it belongs

<u>*For controllers*</u>

Controllers upon receiving a packet from another node or from the application layer will take the following actions:

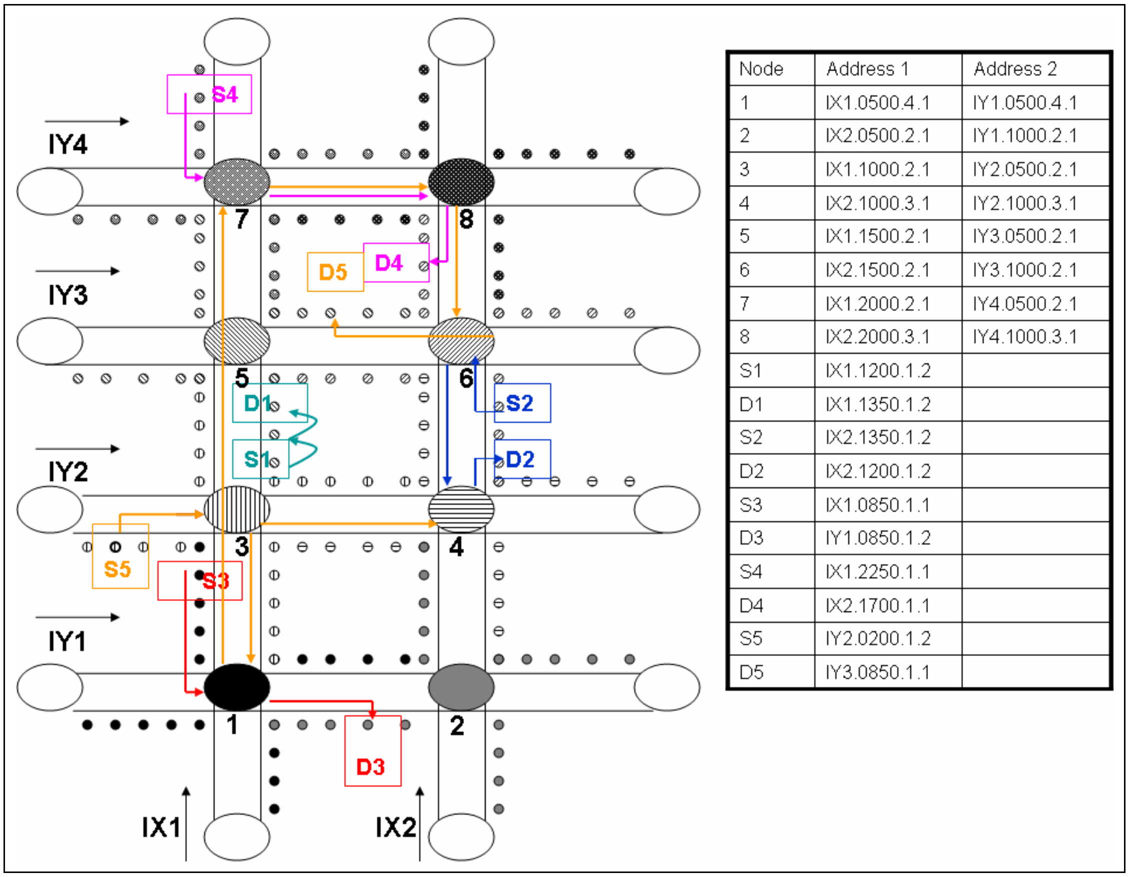
1. If destination address RID matches one of its RIDs, the controller compares the mileage of the destination address and forwards the packet to the next hop on its neighbor-associated segment having the same RID and mileage closer to the destination.
2. If the destination RID is different from all of its RIDs, the controller checks in its table if there is any reachable controller having a matching RID. If there is a match, the controller sends a packet to the next controller corresponding to the matched entry.
3. Otherwise, the packet is redirected to the parent controller by setting the RF field and putting the parent controllers address in the HRDA field.

```
If a packet P is received by a node N
If N is controller, Controller Address is cont
If N is a sensor, Sensor Addressv is sens
The parent of N, Parent Address is parent
The child of N, Child Address is child
Packet Destination Address: P.dest
Set of next hops along all of the controller's neigbor-associated  segments: NH

For Sensors
if ((P.PREV_HOP == parent) && (P.dest != sens))
  forward P to child
if ((P.PREV_HOP ==child) && (P.dest != sens))
  forward P to parent
if (P.dest == sens)
  send P to application

For Controllers
if (P.dest.RID == cont.RID) {
  if (P.dest.Mileage > cont.Mileage)
    forward P to next_hop where (next_hop IN NH && P.dest.Mileage >next_hop.Mileage > cont.Mileage)
  if (dest.Mileage < cont.Mileage)
    forward P to next_hop where (next_hop IN NH && P.dest.Mileage <next_hop.Mileage <  cont.Mileage)
}
else {
   if (dest.RID == table_ent.DESTINATION.RID)
      forward P to table_ent.NEXT_HOP
   else {
      P.RF = SET; P.HRDA = parent;
      forward P to table_ent.NEXT _HOP where (table_ent.DESTINATION ==parent)
          }
      }
```

Figure A.6: Packet routing algorithm

A.5 Routing Examples

To demonstrate the routing procedure, five different type of routing examples to route packet from source to destination were considered. Fig. A.7 has two higher level clusters with each higher level cluster controlling 4 intersections (16 clusters). The controllers 1, 2, 3 and 4 form the first level 2 cluster which is controlled by controller 4. The controllers 5, 6, 7 and 8 form the second level 2 cluster which is controlled by controller 8. The level 3 cluster consists of both the level 2 clusters and is controlled by controller 1. Every node will always forward packet on the segment giving lesser delay as described in the scheduling algorithm. We show the routing of five different packets in the following section.



Node	Address 1	Address 2
1	IX1.0500.4.1	IY1.0500.4.1
2	IX2.0500.2.1	IY1.1000.2.1
3	IX1.1000.2.1	IY2.0500.2.1
4	IX2.1000.3.1	IY2.1000.3.1
5	IX1.1500.2.1	IY3.0500.2.1
6	IX2.1500.2.1	IY3.1000.2.1
7	IX1.2000.2.1	IY4.0500.2.1
8	IX2.2000.3.1	IY4.1000.3.1
S1	IX1.1200.1.2	
D1	IX1.1350.1.2	
S2	IX2.1350.1.2	
D2	IX2.1200.1.2	
S3	IX1.0850.1.1	
D3	IY1.0850.1.2	
S4	IX1.2250.1.1	
D4	IX2.1700.1.1	
S5	IY2.0200.1.2	
D5	IY3.0850.1.1	

Figure A.7: Routing examples

***S1**(IX1.1200.1.2) → **D1**(IX1.1350.1.2)*

S1 sends the packet to its parent and the parent forwards it to its parent till it finally reaches the destination D1.

***S2** (IX2.1350.1.2) → **D2** (IX2.1200.1.2)*

S2 sends the packet to its parent who forwards the packet to its parent and the packet after multi-hop forwarding reaches their cluster head (controller 6). The controller 6 checks the destination address RID field and finds that destination is on the same highway. It then compares the mileage field of the destination and recognizes that the destination is at a lower mileage and thereby forwards the packet to the next hop along its neighbor-associated segment having the same RID as that of the destination and at a lower mileage. The controller 4 gets the packet and again runs the same checks and forwards the packet which gets delivered at the destination by multi-hop forwarding.

***S3** (IX1.0850.1.1) →**D3** (IY1.0850.1.1)*

The source S3 as before routes the packet to its cluster head (controller 1). The controller 1 checks the RID fields of the destination address and recognizes that the destination is on one of its connected segment. It then checks the mileage to find that the destination is at a higher mileage and thereby forwards the packet to the next hop on its neighbor-associated segment having the same RID and located at a higher mileage. The packet finally reaches the destination by multi-hop forwarding.

***S4** (IX1.2250.1.1) →**D4** (IX2.1700.1.1)*

The source S4 as before routes the packet to its cluster head (controller 7). Controller 7 tries to check the RID of the destination address but it fails to find a match with its RIDs. It then looks into the table and figures out that RID of the address of controller 8 matches with that of the destination and thereby routes the packet to controller 8. Controller 8 then delivers the packet to the destination.

***S5** (IY2.0200.1.2) →**D5** (IY3.0850.1.1)*

The sensor sends the packet to its cluster head (controller 3). Controller 3 tries to check the RID of the destination address but it fails to find a match with its RIDs and thereby tries to look up the table. The routing table entries at controller 3 also do not show a matching controller with the same RID as the destination. The controller 3 thereby routes the packet to its parent – its level 2 cluster head (controller 4). Controller 4 also fails to find a route to the destination as it only has the entries for controllers belonging to its cluster and thereby forwards the packet to its parent - its level 3 cluster head (controller 1). Controller 1 upon receiving packet checks to see its routing table entries and the routing table at controller 1 has the entry of all the controllers in the simulated network. It finds a match with the routing table entry of controller 5 and thereby forwards the packet towards controller 5. Controller 5 upon receiving the packet delivers it to the correct destination.

REFERENCES

[1]European Commission, “Intelligent transport systems,” http://europa.eu.int/comm/transport/.

[2] New South Wales, Australia, “Sydney coordinated adaptive traffic system (SCATS),” http://www.traffic-tech.com/pdf/scatsbrochure.pdf

[3] U.S. Dept. of Transportation, “Intelligent transport systems: Technology overview,” http://www.its.dot.gov/index.htm

[4] U. C. Berkeley, “Freeway Performance Measurement System (PeMS),” http://pems.eecs.berkeley.edu/

[5] Speedinfo Inc., “SpeedInfo deploys real time traffic sensor network for SFO Bay area,” http://www.speedinfo.com

[6] D. Estrin, L. Girod, G. Pottie, and M. Srivastava, “Instrumenting the world with wireless sensor networks,” *in Proceedings of IEEE ICASSP*, pp.2033-2036, 2004

[7] K.-C. Wang, M. Chowdhury, R. Fries, M. Atluri, and N. Kanher, “Real-time traffic monitoring and automated response with wireless sensor networks,” *in Proceedings of the ITS World Congress*, Oct. 2005

[8] W. Ye, J. Heidemann, D. Estrin, “Medium access control with coordinated adaptive sleeping for wireless sensor networks”, *IEEE/ACM Transactions on Networking*, Volume: 12, Issue: 3, Pages:493 - 506, June 2004

[9] T.V. Dam and K. Langendoen, “An Adaptive energy-efficient MAC protocol for wireless sensor networks”, *The First ACM Conference on Embedded Networked Sensor Systems (Sensys‘03*), Los Angeles, CA, USA, November, 2003

[10] Gang Lu, Bhaskar Krishnamachari, Cauligi Raghavendra, “An Adaptive Energy-Efficient and Low Latency MAC for Data Gathering in Sensor Networks”, in *4th IEEE International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks WMAN*, April 2004.

[11] P. Lin, C. Qiao, and X. Wang, “Medium access control with a dynamic duty cycle for sensor networks”, *IEEE Wireless Communications and Networking Conference*, Volume: 3, Pages: 1534 - 1539, 21-25 March 2004

[12] Tao Zheng, Sridhar Radhakrishnan, Venkatesh Sarangan, "PMAC: An adaptive energy-efficient MAC protocol for wireless sensor networks," *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 12*, p.237.1, Apr 04-08, 2005.

[13] C. C. Enz, A. El-Hoiydi, J-D. Decotignie, V. Peiris, “WiseNET: An ultralow-power wireless sensor network solution”, *IEEE Computer*, Volume: 37, Issue: 8, August 2004

[14] V. Rajendran, K. Obraczka, J.J. Garcia-Luna-Aceves, “Energy-efficient, collision-free medium access control for wireless sensor networks”, *Proc. ACM SenSys*

[15] R. Kalidindi, L.Ray, R. Kannan, S. Iyengar, “Distributed energy aware MAC layer protocol for wireless sensor networks”, *in International Conference on Wireless Networks* , Las Vegas, Nevada, June 2003

[16] Injong Rhee, Ajit Warrier, Mahesh Aia, and Jeongki Min, “ZMAC: a hybrid mac for wireless sensor networks”, *ACM Sensys*, Nov. 2005

[17] S. Lindsey, C. S. Raghavendra and K. Sivalingam, "Data gathering in sensor networks using the energy*delay metric", *in the Proceedings of the IPDPS Workshop on Issues in Wireless Networks and Mobile Computing*, San Francisco, CA, April 2001

[18] “ITS survey 2004,” http://itsdeployment2.ed.ornl.gov/its2004/, 2004

[19] M. Papageorgiou, C. Diakaki, V. Dinopoulou, A. Kotsialos, and Y.Wang, “Review of road traffic control strategies,” *in Proceedings of the IEEE*, 2003, pp. 2043–2067

[20] Federal Highway Administration, “Intelligent transportation systems in work zones,” http://ops.fhwa.dot.gov/wz/technologies/springfield/

[21] S. Coleri-Ergen and P. Varaiya, “Pedamacs: Power efficient and delay aware medium access protocol for sensor networks”, *IEEE Trans. on Mobile Computing*, 5 (2006), pp. 920 - 930, 2004.

[22] Now Wireless Limited, “Mesh-enabled solutions for intelligent transportation systems,” http://www.nowwireless.com/nm/app transportation.htm.

[23] K.-C. Wang and P. Ramanathan, “Location-centric networking in distributed sensor networks*,” in Frontiers in Distributed Sensor Networks*, CRC Press, 2003.

[24]Y. Yu, D. Estrin, and R. Govindan, “Geographical and energy-aware routing: A recursive data dissemination protocol for wireless sensor networks,” *UCLA Computer Science Department Technical Report*, UCLA-CSD TR-01-0023, May 2001.

[25] Tomasz Imieliński , Julio C. Navas, “GPS-based geographic addressing, routing, and resource discovery”, *Communications of the ACM*, v.42 n.4, p.86-92, April 1999

[26] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu and Jorjeta Jetcheva "A performance comparison of multi-hop wireless ad-hoc network routing protocols", *in Proceedings of the Fourth Annual International Conference on Mobile Computing and Networking (MobiCom'98),* ACM, Dallas, TX, October 1998.

[27] T. Imielinski and S. Goel, "DataSpace: Querying and monitoring deeply networked collections in physical space," *IEEE Personal Communications Magazine*, vol. 7, pp. 4–9, October 2000.

[28] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless sensor networks," *in the Proceeding of the Hawaii International Conference System Sciences*, Hawaii, January 2000.

[29] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power Efficient Gathering in Sensor Information Systems," *in the Proceedings of the IEEE Aerospace Conference*, Big Sky, Montana, March 2002.

[30] M. Stemm and R. H. Katz, "Measuring and reducing energy consumption of network interfaces in hand-held devices*," IEICE Trans.Commun.,* vol. E80-B, no. 8, pp. 1125–1131, Aug. 1997.

[31] O. Kasten. Energy consumption. Eldgenossische Technische Hochschule Zurich. [Online]. Available: http://www.inf.ethz.ch/~kasten/research/bathtub/energy_consumption.html

[32] Young-Bae Ko , Nitin H. Vaidya, "Location-aided routing (LAR) in mobile ad hoc networks", *in Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking,* p.66-75, October 25-30, 1998, Dallas, Texas, United States

[33] B. Crow et al., "Investigation of the IEEE 802.11 Medium Access Control (MAC) Sublayer Functions," *in Proceedings of INFOCOM 97*, Kobe, Japan, Apr. 1997, pp. 126–33.

[34] K. Whitehouse, A. Woo,F. Jian, J. Polastre, and D. Culler, "Exploiting The Capture Effect For Collision Detection and Recovery", *in Proceedings of the IEEE Em-NetS-II Workshop*, May 2005.

[35] Z. Hadzi-Velkov and B. Spasenovski, "On the capacity of IEEE 802.11 DCF with capture in multipath-faded channels," *International Journal of Wireless Information Networks*, vol. 9, no. 3, pp. 191–199, July 2002.