2007

Application-specific reliable data transfer in wireless sensor networks

Damayanti Datta
Louisiana State University and Agricultural and Mechanical College, ddatta8@gmail.com

APPLICATION-SPECIFIC RELIABLE DATA TRANSFER IN WIRELESS SENSOR NETWORKS

A Dissertation

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Computer Science

by
Damayanti Datta
B.E., Jadavpur University, 1996
M.S., Georgia Southwestern State University, 2001
August 2007

Acknowledgements

I am very grateful to those people who have made this dissertation possible and to those who have made my experience in the graduate school one that I will always remember fondly.

I want to deeply thank my major professor Dr. Sukhamay Kundu for his patient guidance throughout the doctoral study. Without his knowledge, expertise, patience and guidance, this work would not have been possible. He challenged me to do my best in learning, thinking, writing, research and personal growth and his support provided me with the motivation and determination to complete this degree.

My committee members Dr. Doris Carver, Dr. Subhash Kak, Dr. Jianhua Chen and Dr. Young H. Chun, deserve my deepest thanks and appreciation for their time and willingness to endure this long journey with me.

I want to thank my friends Alina Trifas and Lulin Zhang in the graduate school for always being there for me.

Last but not the least, I want to express my gratitude to two very dear friends Ms. Vera Watkins and Ms. Lynette Jackson for loving and encouraging me always through out this long journey.

Table of Contents

List of Tables

List of Figures

Abstract

A wireless sensor network (WSN) is a collection of sensor nodes and base stations connected via wireless medium. It sends data collected from the nodes to the base stations for generating information. The size and low cost of the sensor nodes as well as the WSN's ability to connect without wired links are its key advantages which enable it to be deployed in hostile or inaccessible environments at low cost. However, WSNs suffer from high data loss due to the inherent weaknesses in a wireless transmission medium, transmission problems in hostile environments due to human interference, etc. and node failures due to limited energy of sensor nodes. Hence ensuring data transfer with minimum loss i.e. reliable data transfer is very important in WSNs. The amount of loss tolerated is application dependent.

We present a reliable protocol for data transfer from a base station to sensor nodes for time-critical applications in WSNs with zero tolerance for data loss. The protocol is based on hop-by-hop detection and recovery of lost data packets, out-of-sequence forwarding of packets and delayed request for missing packets at each node with non-acknowledgement of packets at each receiving node. We present a detailed analysis of the advantages of the key features of our protocol over other alternatives. The superiority of our protocol over an established protocol PSFQ is demonstrated via extensive simulations, in terms of both the delivery time of the entire data (sent from the base station to the sensor nodes) and the number of messages exchanged in the network during this process.

In addition, we present two methods, one, for ensuring that at least one packet is delivered to a node in non-acknowledgement based systems and another, for sending reports from destination nodes to the base station respectively.

We explore different methods for further improvement of protocol performance: (1) use of effective degree of a node in determining the optimum delay for requesting data packets missing at nodes, (2) variation of the priority order for sending different types of messages at nodes and, (3) selective response to requests for packets at nodes.

Chapter 1

Introduction

1.1 Overview of Wireless Sensor Networks (WSNs)

Computing technology becomes exponentially smaller and cheaper with each passing year according to Moore's law. While silicon scaling marches on, the same semiconductor manufacturing processes are being utilized to build microscopic mechanical structures that interact with the physical world. Scientists and engineers are now utilizing these phenomena in ways that enable a new role for computing in science. They use them to build processors, radios and exceptionally small electromechanical structures that sense fields and forces in the physical world; these can be combined to form inexpensive, low power communication devices deployed throughout a physical space, providing dense sensing close to physical phenomena as well as processing and communication of this information, and coordination of actions with other nodes, thus forming a network known as a Wireless Sensor Network (WSN).

A wireless sensor network consists of tiny and unobtrusive sensor nodes which configure themselves to aid the formation of the wireless network, and is inexpensive. Hence, wireless sensor networks can be deployed in areas which are difficult to access e.g. in enemy territory for surveillance purposes, animal/plant habitat monitoring in dense forests, within human body parts for monitoring diseases like cancer, etc. This has lead to increased application and research in the area of wireless sensor networks.

1.1.1 Application Areas of WSNs

- **Military**. Wireless sensor networks are used for battlefield surveillance, monitoring vehicular traffic, tracking the position of the enemy, etc. For example, in Iraq's Al Jazirah desert, only the most observant eye would note the new smattering of stones spread across a scraggly acre near the Syrian border. An even closer look would reveal these stones for what they truly are: a network of several thousand camouflaged sensors scattered the night before by a low-flying U.S. military plane. These sensors will scout the border for evidence of arms smuggling. As dusk turns to night and stars spill across

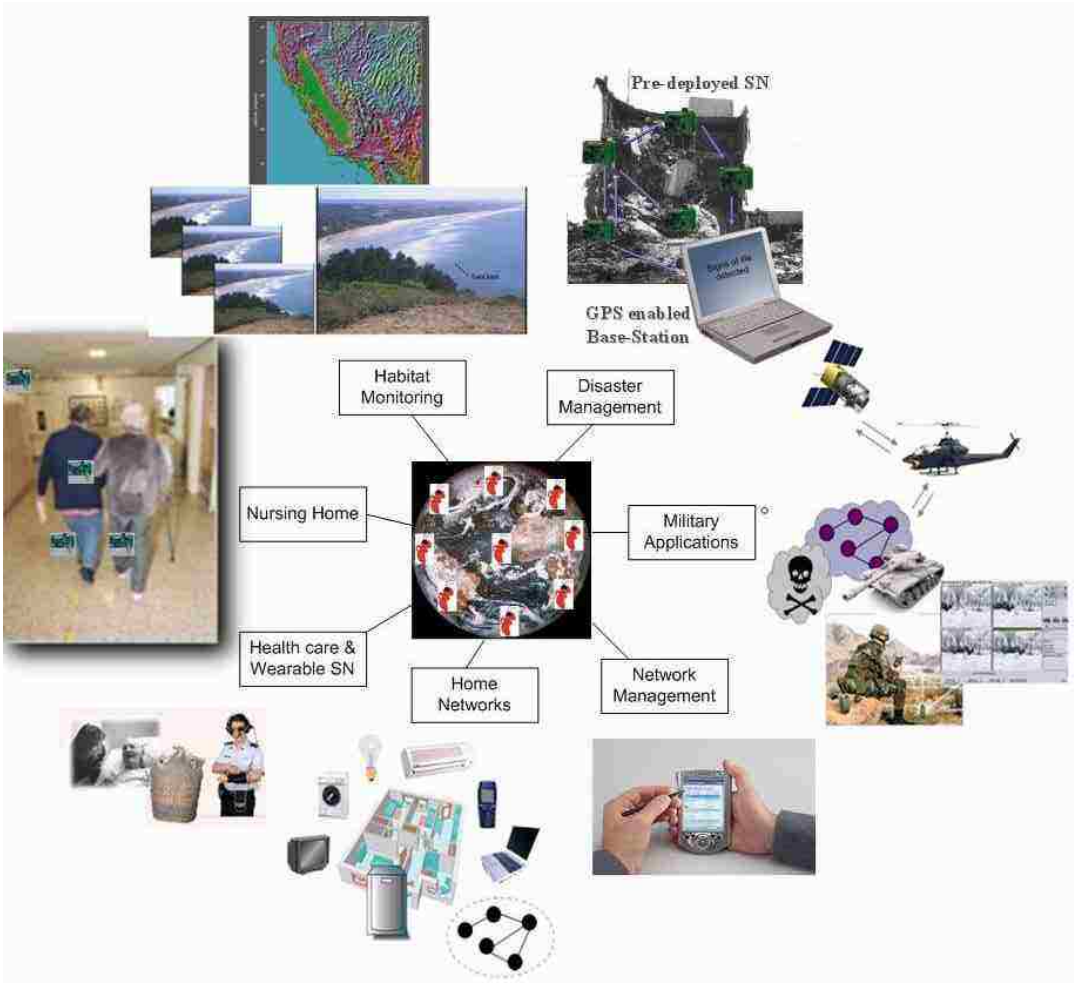


Figure 1.1: Applications of wireless sensor networks.

the sky, a faint rumble stirs the sand. The sensors detect the rumble and match it to the acoustic signature of a heavy truck, perhaps a half-mile away. The information is detected and relayed to the base station.

- **Environmental observation**. Wireless sensor networks can be used to monitor environment like forest fire detection, flood detection, air pollution detection, rainfall observation in agriculture, etc. One example is water pollution detection in a lake that is located near a factory that produces toxic waste illegally dumped into the lake. Sensor nodes can be randomly deployed here and used to relay the exact origin of a pollutant to a centralized authority which then takes appropriate measures to limit the spread of pollution. Without the wireless sensor network , it would be difficult to get the data without the nearby factory's knowledge in which case the factory would prevent the data gathering process.

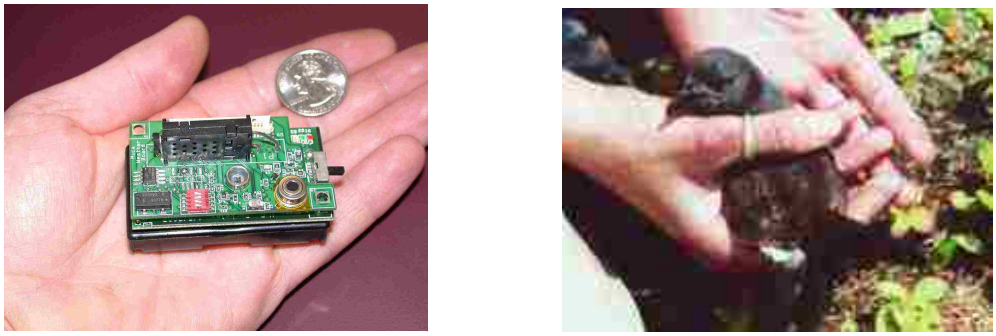


Figure 1.2: Sensor nodes used to monitor the behavior of storm petrel.

- **Habitat Monitoring**. Wireless sensor networks are used to gather information on the habitat of a plant/animal. This data is later analyzed to learn the optimal environment favourable for the plant/animal's growth. Examples include dispersal patterns of wind-borne seeds, the water profiles experienced by spawning salmon, insect densities across riparian environments, and the microclimate of meadow and woodland transects.

  Wireless sensor networks have been used to capture a detailed picture of the complex spatial variation and temporal dynamics of the microclimate surrounding a coastal redwood tree [10] shown in Figure 1.4. When one walks in a redwood forest it is temperate and moist, despite the wide variation in weather conditions. The top of the tree experiences wide variation in temperature, humidity, and, of course, light, whereas the bottom is typically cool, moist, and shaded. This variation was understood to create non-uniform gradients, essentially weather fronts, that move through the structure of the tree. For example, as the sun rises, the top of the canopy warms quickly. This warm front moves down the tree over time until the entire structure stabilizes or until cooling at the canopy surface causes the process to reverse.

  Another example of application of wireless sensor networks in habitat monitoring is the Great Duck Island System (GDI) [1] developed at the Great Duck Island, Maine by the researchers of UCB/Intel Research Laboratory to monitor the behaviour of storm petrel shown in Figure 1.2 For scientists studying the behavior of storm petrel, monitoring the shy seabird's nest activity meant sticking a cumbersome remote camera or a daring arm into burrows which was difficult because of the seabird's nature. It involved obtaining an accurate count of the elusive seabirds which in turn involved

expensive, carefully planned trips to the island with pen, paper and a portable video system dubbed the "petrel peeper" that was transported by wheelbarrow or by several biology students. However, with wireless sensor networks, a wireless network of more than 20 miniaturized sensors, or motes, were installed in the burrows of the storm petrels on the nearby Great Duck Island. Each sensor, slightly bigger than the two AA batteries powering it, beamed back raw data about the conditions in the burrows and the island's microclimate. This process was less expensive and cumbersome and the data had helped biologists understand why the the storm petrels favor Great Duck Island over thousands of other islands off the coast of Maine which was particularly important in answering questions related to conservation of habitat.

Figure 1.3: Enclosure to protect sensor nodes used to monitor the behavior of storm petrel.

- **Building monitoring**. Sensors can be used in buildings to monitor fire and smoke detection. A network of sensors, capable of detecting smoke, can be deployed in a huge building which is on fire to track the source and the direction in which the fire is expanding. In addition, sensors can be used to monitor vibration that could damage the structure of a building.
- **Healthcare**. Sensors are used in biomedical applications to improve the quality of the provided care. Sensors are implanted in the human body to monitor medical problems like cancer and help patients maintain their health. Smart Sensors and Integrated Microsystems (SSIM) [2] builds retina prosthesis chips consisting of 100 microsensors that can be implanted within human eye allowing patients with limited or no vision to see at an acceptable level. Wireless sensor networks can also be used to monitor

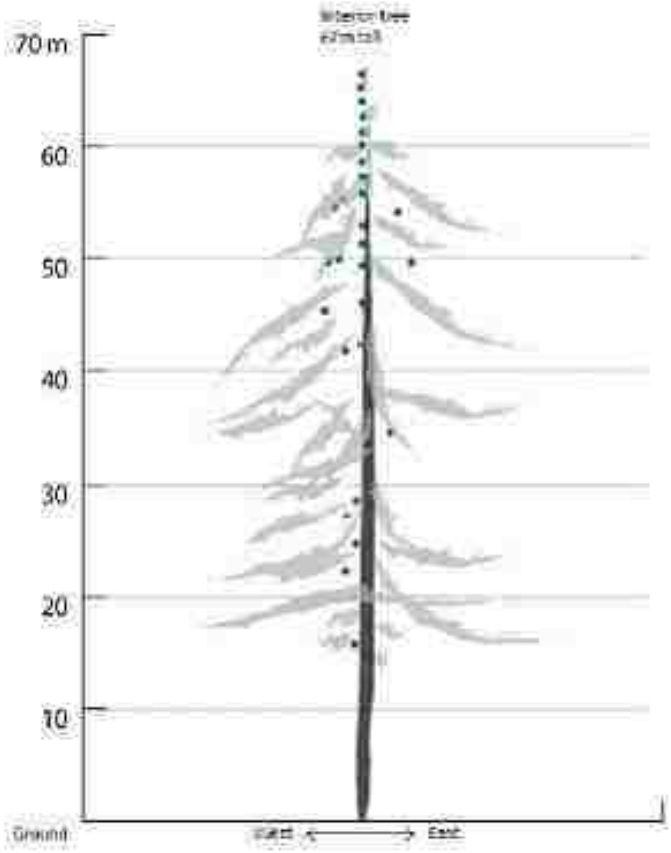


Figure 1.4: Placement of sensor nodes on a redwood tree.

various conditions of the body and relay information which save costly trips to the physician.

- **Home and other commercial applications**. Wireless sensor networks can be used for automation of various devices used at homes and in offices.

1.1.2 Architecture of WSNs

A wireless sensor network consists of tens or hundreds or thousands of low cost nodes which could either have a fixed location or are randomly deployed. Sensor nodes collect data and send it to special nodes called base stations (sometimes they are also referred to as sinks). The sensor nodes communicate with each other and with the base station via wireless. Base stations are linked to the user interface via wired links in many cases. Though sensor nodes and base stations are generally static, sometimes some of them can be mobile e.g. mobile base stations are used to gather data from sensors in enemy territory for military applications. An example of a mobile sensor node is a node attached to an animal which gathers data on

the animal and stores it till the animal moves to a position where there is a static sensor node. The mobile node then transmits the data to the static node which then routes it to the base station.

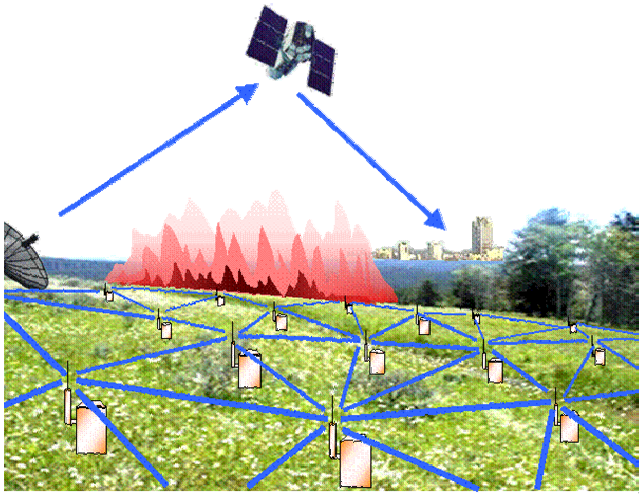


Figure 1.5: A wireless sensor network.

Sensor nodes can be dropped on a designated territory by airplane, missile, etc. or they can be placed there by humans and robots. After deployment, the sensor nodes will configure themselves to form the wireless sensor network. The **WSN topology** can vary dynamically because of addition of more nodes and node failures due to energy loss, enviromental disturbances, etc.

A **sensor node** has the following components:

- Processor and memory. The processor is responsible for control of the sensors, execution of communication protocols, etc. Memory is present for storage of data for various purposes like data aggregation, etc.
- Sensors and analog-to-digital converters. The sensors gather data from the environment and send it to the processor. If they are analog sensors, the signals are sent to the analog-to-digital converters which convert them to digital format and then send them to the processor.
- Transceiver unit. This transmits and receives radio signals or optical signals.
- Power unit. This may be batteries or solar cells.

Sensor nodes are characterized by small size, low power and short transmission ranges. The problem of short transmission ranges are overcome by high node density.

Figure 1.6: MICA mote.

MICA mote is a commercially available sensor node which has been developed by University of California, Berkeley [38]. These motes come in two sizes:

- Rectangular, measuring 2.25 x 1.25 by 0.25 inches (5.7 x 3.18 x.64 centimeters), it is sized to fit on top of two AA batteries that provide it with power.
- Circular, measuring 1.0 by 0.25 inches (2.5 x .64 centimeters), it is sized to fit on top of a 3 volt button cell battery.

The MICA mote uses an Atmel ATmega 128L processor running at 4 MHz. The 128L is an 8-bit microcontroller that has 128 kilobytes of onboard flash memory to store the mote's program. This CPU is about as powerful as the 8088 CPU found in the original IBM PC (circa 1982). The big difference is that the ATmega consumes only 8 milliamps when it is running, and only 15 microamps in sleep mode. This low power consumption allows a MICA mote to run for more than a year with two AA batteries. A typical AA battery can produce about 1,000 milliamp-hours. At 8 milliamps, the ATmega would operate for about 120 hours if it is operated constantly. However, a programmer can ensure that the CPU is asleep most of the time, allowing the battery life to be extended considerably. For example, a mote might sleep for 10 seconds, wake up and check status for a few microseconds, and then go back to sleep. MICA motes come with 512 kilobytes of flash memory to hold data. They

also have a 10-bit A/D converter so that sensor data can be digitized. Separate sensors on a daughter card can connect to the mote. Sensors available include those for sensing temperature, motion, light, sound and magnetic fields. Advanced sensors for GPS signals are under development. The final component of a MICA mote is the radio. It has a range of several hundred feet and can transmit approximately 40,000 bits per second. When it is off, the radio consumes less than one microamp. When receiving data, it consumes 10 milliamps. When transmitting, it consumes 25 milliamps. Conserving radio power is key to long battery life. All of these hardware components together create a MICA mote. A programmer writes software to control the mote and make it perform a certain way. Software on MICA motes is built on an operating system called TinyOS. TinyOS is helpful because it deals with the radio and power management systems and makes it much easier to write software for the mote.

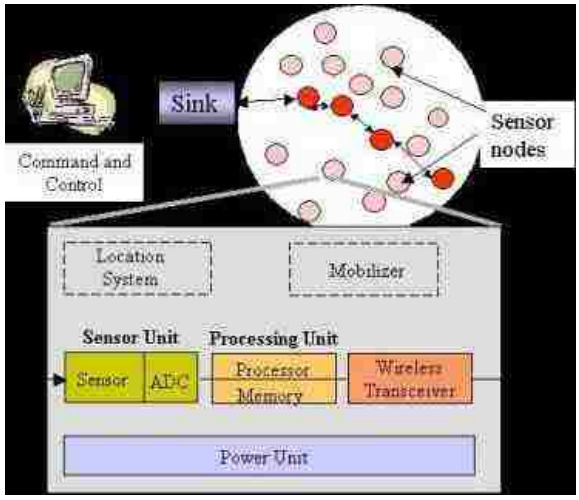


Figure 1.7: Internal structure of a sensor node.

The **base station** links the wireless sensor network to user interface to disseminate the data or filtered data for further processing. Base stations have enhanced capabilities over simple sensor nodes since they must do complex data processing; this justifies the fact that base stations have workstation/laptop class processors, and of course enough memory, energy, storage and computational power to perform their tasks well. The communication between base stations is initiated over high bandwidth links. Base stations can be connected to user interfaces via wired links.

The **network communication model** for wireless sensor networks [35] consists of

five basic layers namely physical layer, data link layer, network layer, transport layer and application layer.

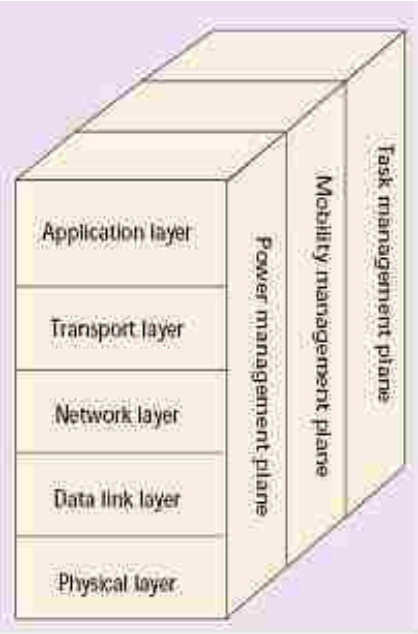


Figure 1.8: Network communication model for wireless sensor networks.

- **Physical layer.** The responsibilities of the physical layer are frequency selection, carrier frequency generation, signal detection, and modulation. Frequency generation and signal detection depend on hardware design. Two factors are important in the design of the physical layer of wireless sensor networks: (i) Energy minimization (ii) Propagation and fading effects of the signal. Minimum power required is inversely proportional to n^{th} power of the distance. For low lying antenna and near ground channels as in wireless sensor network, the value of n is closer to 4. It has been shown that power starts to drop with higher exponents at smaller distances for low antenna heights. This effect can be mitigated by exploiting the spatial density in wireless sensor network. In wireless sensor network, the nodes are placed close to each other and this ameliorates the problem due to signal loss with distance. Energy minimization is a more important concern when designing the physical layer of wireless sensor networks. Choosing the modulation scheme and frequency band is important in wireless sensor networks. There is a trade-off between efficiency vs. energy minimization. For example, consider the binary and M-ary modulation schemes. While the M-ary modulation schemes can

reduce transmission times by sending multiple bits per symbol, it required complex circuitry and increased radio power consumption. The binary modulation scheme is more energy efficient here because it requires low power and simple transceiver circuitry.

- **Data link layer.** The responsibilities of the data link layer are the medium access control (MAC), error control, etc. MAC protocols are especially important as they can reduce collisions and prevent data loss. MAC protocols for other wireless networks are not suitable for wireless sensor networks as sensor nodes are energy constrained. There are two types of MAC protocols for wireless sensor networks:

  In **contention-based** MAC protocols, nodes compete for a shared channel, resulting in probabilistic coordination. Collision happens during the contention procedure in such systems. Contention protocols have several advantages compared to scheduled protocols. First, because contention protocols allocate resources on-demand, they can scale more easily across changes in node density or traffic load. Second, contention protocols are more flexible as topologies change. There is no requirement to form communication clusters, and peer-to peer communication is directly supported. Finally, contention protocols do not require fine-grained time synchronizations as in TDMA protocols. The major disadvantage of a contention protocol is its inefficient usage of energy. Nodes listen at all times and this wastes energy.

  Carrier Sense Multiple Access (CSMA) is a contention-based MAC protocol. In CSMA, a node listens to the channel before transmitting. Its central idea is listening before transmitting. The purpose of listening is to detect if the medium is busy, also known as carrier sense. If it detects a busy channel, it delays access and retries later. However CSMA cannot solve the **hidden terminal problem**.

  The hidden terminal problem is shown in Figure 1.9, where either of nodes A

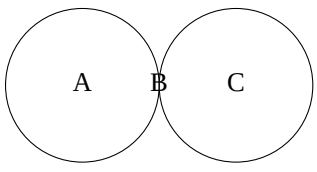


Figure 1.9: Hidden terminal problem.

and C can sense the medium and find it clear and transmit to node B. Because nodes A

and C are not in the range of each other, neither can determine by sensing the medium whether the other is transmitting or not. Due to this, it is possible that node B will receive data from both at the same time which will lead to colllisions and hence, data loss.

CSMA/CA, where CA stands for collision avoidance, was developed to address the hidden terminal problem. The basic mechanism in CSMA/CA is to establish a brief handshake between a sender and a receiver before the sender transmits data. The handshake starts from the sender by sending a short Request-to-Send (RTS) packet to the intended receiver. The receiver then replies with a Clear-to-Send (CTS) packet. The sender then starts sending data after it receives the CTS packet. The purpose of RTS-CTS handshake is to make an announcement to the neighbors, of the transaction between both the sender and the receiver. If a node overhears an RTS or CTS destined to other nodes, it should not send its own packet. CSMA/CA does not completely eliminate the hidden terminal problem, but now the collisions are mainly on the control message packets. Since the RTS and CTS packets are very short, the cost of collisions is greatly reduced. However, RTS/CTS has a considerable overhead. Based on CSMA/CA, Karn proposed MACA [24], which added a duration field in both RTS and CTS packets indicating the amount of data to be transmitted, so that other nodes know how long they should withhold from sending their own packets. Bharghavan et al. further improved MACA in their protocol MACAW [25]. MACAW proposed several additions to MACA, including use of an acknowledgment (ACK) packet after each data packet, allowing rapid link-layer recovery from transmission errors. The transmission between a sender and a receiver follows the sequence of RTS-CTS-DATA-ACK. IEEE 802.11 adopted all these features of CSMA/CA, MACA and MACAW in its distributed coordination function (DCF), and made various enhancements, such as virtual carrier sense, binary exponential back-off, and fragmentation support [23].

Another contention based protocol, PAMAS, proposed by Singh and Raghavendra [4], avoids overhearing by putting nodes into sleep state when their neighbors are in transmission. PAMAS uses two channels, one for data transmission and one for

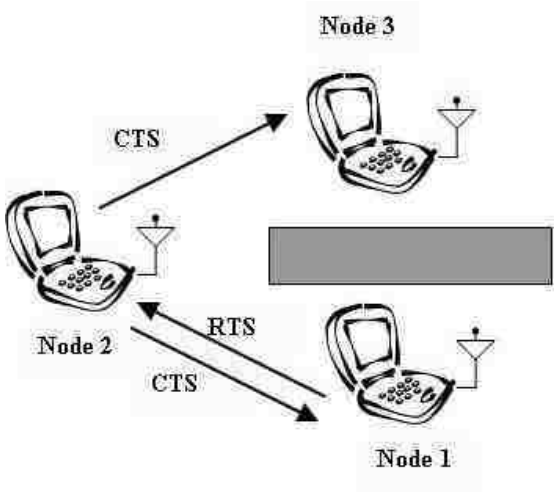


Figure 1.10: RTS-CTS messaging system.

control message transmission. After a node wakes up from sleep, it probes in the control channel to find any possible ongoing transmissions and their durations. If any neighbor answers the probe, the node will sleep again for the specified duration. Probing in the control channel avoids interfering with a neighbors transmission in the data channel, and the neighbor is able to answer the probe in the control channel without interrupting its data transmission. However, the requirement of separate control and data channels makes PAMAS more difficult to deploy, since multiple channels require multiple radios or additional complex channel allocation.

Recent CSMA-based MAC protocols for wireless sensor networks are S-MAC[30], T-MAC[34], Wise-MAC[31] and SIFT[32]. They suffer from the same disadvantages of CSMA-based protocols and overhead due to sleep-listen periods for nodes which aim to save energy consumption.

In **reservation-based** MAC protocols like Time Division Multiple Access (TDMA), the channel is divided into N time slots. In each slot, only one node is allowed to transmit. Hence, it has a natural advantage of collision-free medium access. However, TDMA has some disadvantages that limits its use in wireless sensor networks. TDMA normally requires nodes to form clusters. One of the nodes within the cluster is selected as the cluster head. Nodes are normally restricted to communicate with the cluster head within a cluster; peer-to-peer communication is not directly supported. (If nodes communicate directly, then they must listen during all slots, reducing energy

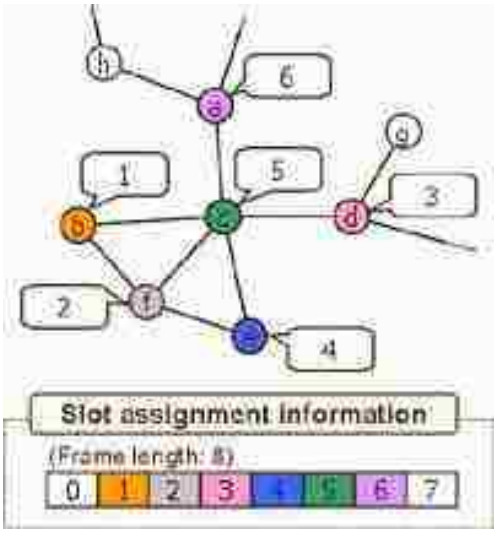


Figure 1.11: Slot allocation for transmission in TDMA.

efficiency.) Inter-cluster communications and interference need to be handled by other approaches, such as Frequency Division Multiple Access (FDMA) or Code Division Multiple Access (CDMA). More importantly, TDMA protocols have limited scalability and adaptivity to the changes on number of nodes. When new nodes join or old nodes leave a cluster, the base station must adjust frame length or slot allocation. However, it is not easy to change the slot assignment within a decentralized environment for traditional TDMA, since all nodes must agree on the slot assignments. Frequent changes may be expensive or slow to take effect. Also, frame length and static slot allocation can limit the available throughput for any given node, and the the maximum number of active nodes in any cluster may be limited. Finally, TDMA protocols depend on distributed, fine-grained time synchronization to align slot boundaries. Many variations on the basic TDMA protocol are possible. Rather than scheduling slots for node transmissions, slots may be assigned for reception with some mechanism for contention within each slot. The base station may dynamically allocate slot assignments on a frame-by-frame basis.

Sohrabi and Pottie proposed a self-organization protocol for wireless sensor networks [26]. The protocol assumes that multiple channels are available (via FDMA or CDMA), and any interfering links select and use different sub-channels. During the time that is not scheduled for transmission or reception, a node turns off its radio to

conserve energy. Each node maintains its own time slot schedules with all its neighbors, which is called a superframe. Time slot assignment is only decided by the two nodes on a link, based on their available time. It is possible that nodes on interfering links will choose the same time slots. Although the superframe looks like a TDMA frame, it does not prevent collisions between interfering nodes, and this task is actually accomplished by FDMA or CDMA. This protocol supports low-energy operation, but a disadvantage is the relatively low utilization of available bandwidth. A sub-channel is dedicated to two nodes on a link, but is only used for a small fraction of time, and it cannot be re-used by other neighboring nodes.

Low-Energy Adaptive Clustering Hierarchy (LEACH), proposed by Heinzelman et al. [27] is an example of utilizing TDMA in wireless sensor networks. LEACH organizes nodes into cluster hierarchies, and applies TDMA within each cluster. The position of cluster head is rotated among nodes within a cluster depending on their remaining energy levels. Nodes in the cluster only talk to their cluster head, which then talks to the base station over a long-range radio. LEACH is an example that directly extends the cellular TDMA model to wireless sensor networks.

Figure 1.12: Bluetooth Smart Sensor Systems.

Bluetooth [28], [29] is designed for personal area networks (PAN) with target nodes as battery-powered PDAs, cell phones and laptop computers. Its design for low-energy operation and inexpensive cost make it attractive for use in wireless sensor

networks. As with LEACH, Bluetooth also organizes nodes into clusters, called piconets. Frequency-hopping CDMA is adopted to handle inter-cluster communications and interference. Within a cluster, a TDMA-based protocol is used to handle communications between the cluster head (master) and other nodes (slaves). The channel is divided into time slots for alternate master transmission and slave transmission. The master uses polling to decide which slave has the right to transmit. Only the communication between the master and one or more slaves is possible. The maximum number of active nodes within a cluster is limited to eight, an example of limited scalability.

However, FDMA brings an additional circuitry requirement to dynamically communicate with different radio channels. This increases the cost of the sensor nodes, which is contrary to the objective of the sensor network systems. CDMA also offers collision-free medium, but its high computational requirement is a major obstacle for energy consumption objective of the sensor networks. In pursuit of low computational cost requirements of wireless CDMA sensor networks, there has been limited effort to investigate source and modulation schemes, particular signature waveforms, designing simple receiver models, and other signal synchronization problems. If it is shown that the high computational complexity of CDMA could be traded with its collision avoidance feature, CDMA protocols could also be considered as candidate solutions for wireless sensor networks.

A recent TDMA-based MAC protocol for wireless sensor networks is DMAC[33]. It suffer from the same disadvantages of TDMA-based protocols.

- **Network layer.** This layer is responsible for data routing. There is also a tradeoff between choosing a route that consumes minimum energy as well as a route which has the minimum number of hops between a source and a destination. The issues [39] to be considered when designing routing protocols are described in detail below.

  - A routing protocol should be able to adapt well to the addition of nodes to a wireless sensor network as well as to the removal of nodes due to node failure in a wireless sensor network i.e. the dynamic topology of a wireless sensor network.
  - While most wireless sensor networks have stationary nodes, some may have mobile

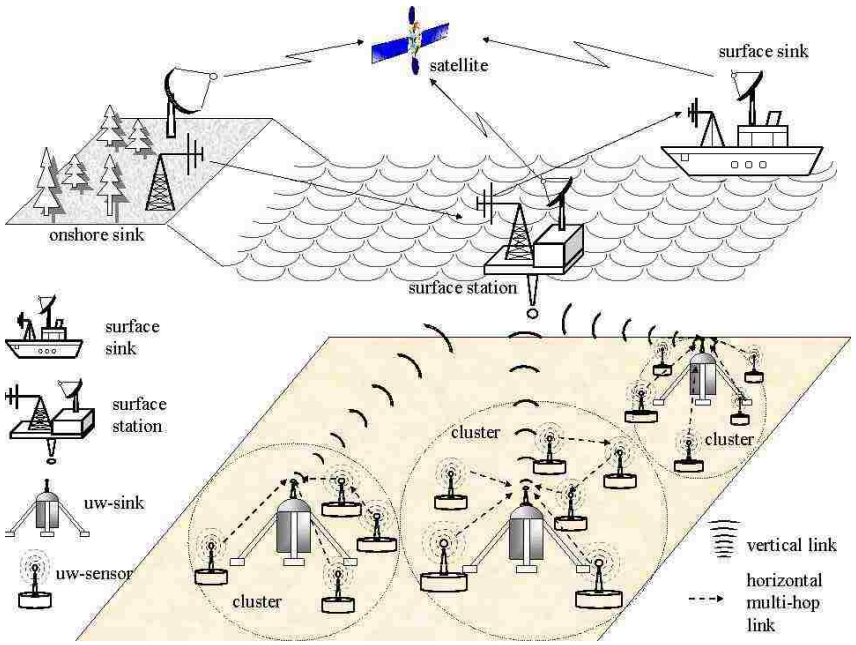


Figure 1.13: Routing in wireless sensor networks.

nodes or even mobile base station. Routing protocols must be designed to consider mobility of nodes/base station during routing.

- Reducing energy consumption without losing accuracy is very important as sensor nodes can use up their limited supply of energy performing computations and transmitting information in a wireless environment. As such, energy-conserving forms of communication and computation are essential in routing protocols for energy-constrained wireless sensor networks.
- Routing protocols should be adapt to the time-critical and QoS requiremets of different applications in wireless sensor networks.
- The coverage achieved by routing protocols when routing from a source to multiple destinations like routing data from a base station to sensor nodes is an important consideration. It should deliver the data to maximum possible targeted destination nodes. Routing protocols should also provide for a wireless sensor network to connect to other wireless sensor networks and the internet.
- For applications in wireless sensor networks having data aggegration at intermediate nodes, routing should be done via cluster-heads where data aggregation takes place within clusters.

There are various ways to classify routing protocols.

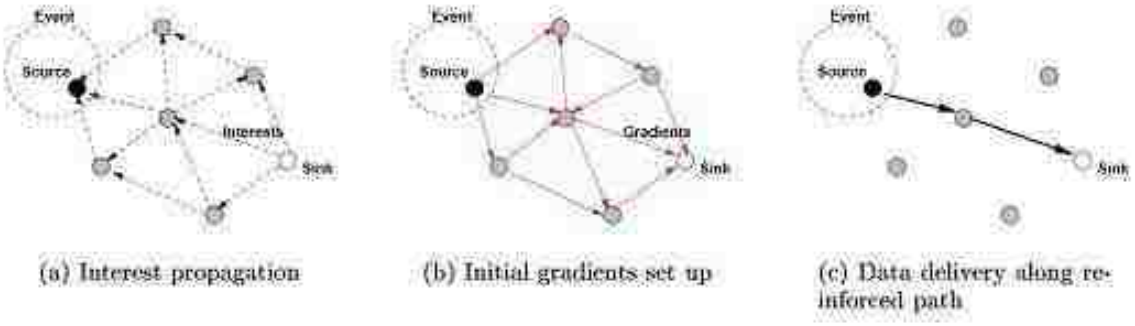


Figure 1.14: Flat routing: directed diffusion.

In flat routing, each node typically plays the same role and sensor nodes collaborate together to perform the sensing task. It is not feasible to assign a global identifier to each node in a wireless sensor network. This has led to data centric routing, where the base station sends queries to certain regions and waits for data from the sensor nodes located in the selected regions. Since data is being requested through queries, attribute-based naming is necessary to specify the properties of data. Examples of flat routing protocols include Directed Diffusion[6], SPIN[8], Rumor Routing[40], Minimum Cost Forwarding Algorithm[41], Gradient-based Routing[42], Information-driven Sensor Querying (IDSQ) and Constrained Anisotropic Diffusion Routing (CADR)[43], COUGAR[44], ACQUIRE[45] and Energy Aware Routing[46].

In hierarchical routing, also called cluster-based routing which was originally proposed in wireline networks, there are well-known techniques with special advantages related to scalability and efficient communication. As such, the concept of hierarchical routing is also utilized to perform energy-efficient routing in wireless sensor networks. In a hierarchical architecture, higher energy nodes can be used to process and send the information while low energy nodes can be used to perform the sensing in the proximity of the target. This means that creation of clusters and assigning special tasks to cluster heads can greatly contribute to overall system scalability, lifetime, and energy efficiency. Hierarchical routing is an efficient way to lower energy consumption within a cluster and performing data aggregation and fusion in order to decrease the number of transmitted messages to the base station. Hierarchical routing is mainly two-layer routing where one layer is used to select clusterheads and the other layer is

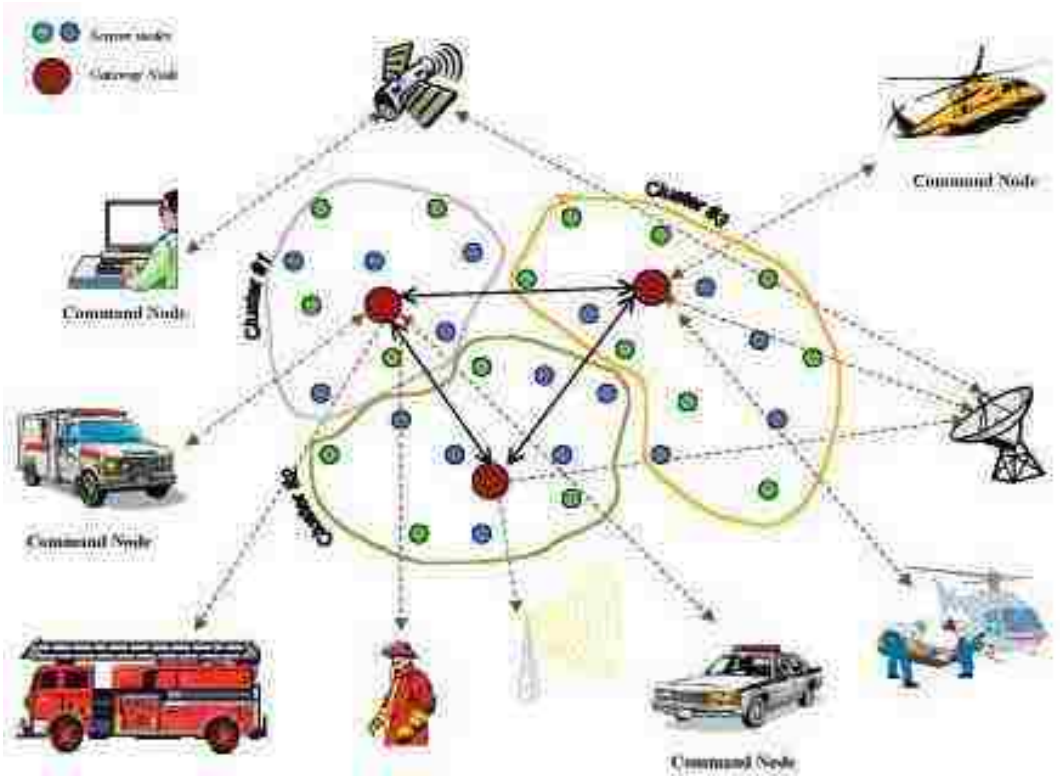


Figure 1.15: Hierarchical or cluster based routing.

used for routing. However, most techniques in this category are not about routing, rather on "who and when to send or process/aggregate" the information, channel allocation etc., which can be orthogonal to the multihop routing function. Examples of routing protocols include LEACH[27], PEGASIS[47], APTEEN[48], Self Organization Protocol[49], Sensor Aggregates Routing[50], Virtual Grid Architecture Routing[51], Hierarchical Power-aware Routing[52] and Two-Tier Data Dissemination[53].

- **Transport layer.** Protocols in the transport layer helps in reliable data transfer in wireless sensor networks. This includes prevention of data loss and recovery of lost data. Transport layer protocols can be of two types: (1) Protocols that try to eliminate reasons for data loss and thus reduce data loss. (2) Protocols that recover lost data. However, protocols designed for wired networks are unsuitable for wireless sensor networks because they assume the primary reason for data loss to be congestion which is not the case for wireless sensor networks. Such assumptions lead to actions which can harm data delivery in wireless sensor networks. Transport layer protocols for other wireless networks are also unsuitable for wireless sensor networks as they do not

consider energy conservation which is essential in wireless sensor networks. A number of reliable data transfer protocols for wireless sensor networks have been described in Chapter 2 as our work considers reliable data transfer in wireless sensor networks for different applications.

- **Application Layer.** At the application layer, processes aim to create effective new capabilities for efficient extraction, manipulation, transport, and representation of information derived from sensor data. In most applications, sensor networks have various functional components: detection and data collection, data fusion, etc. However, application layer protocols for wireless sensor networks is a little explored area.

1.2 Overview of Data Transfer Scenarios in WSNs

Data transfer in wireless sensor networks occurs from sensor nodes to a base station and vice versa. During data transfer from sensor nodes to a base station, either streams of data are sent by the sensor nodes as and when events are detected or the data collected by sensor nodes are filtered at some sensor nodes and then sent to the base station. The filtered data size is small compared to the unfiltered data size. For some applications, some amount of data loss is acceptable. For example, temperature sensors measure the temperature of an environment over time. In this case, only temperatures above a certain threshold value need to be received by the base station. In other applications, all the sensor data must reach the base station. No data loss is acceptable but these applications do not require real-time delivery of data. Real-time delivery of data is required in wireless sensor network applications like those monitoring the location of people trapped in a fire inside a building. Some applications require a huge amount of data transfer in comparison to others like the data captured by image sensors[11].

Data transfer from a base station to sensor nodes consist of instructions for sensor nodes to reprogram themselves to perform a task different from the one they are executing currently. The data sent from the base station to sensor nodes can consist of control code i.e. software programs to enable detection of a particular type of data, sample data for target detection, or queries for data. Sensor nodes in wireless sensor networks tend to

be application specific, and are typically hard-wired to perform specific tasks like sensing temperature only. However, research is being done to build more powerful general-purpose hardware and software environments capable of reprogramming or re-tasking sensors to do a variety of tasks. Such systems are beginning to emerge. For example, the Berkeley motes [12] are capable of receiving code segments from the network and assembling them into a completely new execution image before re-tasking a sensor. To illustrate the application of retasking the sensor nodes, let us consider sensor nodes of a wireless sensor network measuring the intensity of earth movements in any area. A weather monitoring system detects an environmental change which will happen in the area in a short while and wants the sensor nodes to collect data on the temperature during this change. In this case, the base station will send instructions to the sensor nodes to collect data on the temperature instead of data on the intensity of earth movements. A base station can also send instructions to sensor nodes (based on the data already sent by the sensor nodes) to collect data different from that they are already collecting though for the same application. For example, in a border surveillance system, many sensor nodes are scattered near a national border to monitor illegal border crossing activity. When an instrusion is detected, sensor nodes immediatly report this event. A base station gathers the data and based on the information processed, instructs the sensor nodes to change detection parameters [13].

1.3 Reasons for Data Loss or Unreliable Data Transfer in WSNs

The efficiency of any wireless sensor network network depends on the ability of the network to deliver data from a source to one or more destinations without data loss. In other words, a wireless sensor network enabling transfer of data without loss is reliable or supports reliable data transfer. The amount of data loss that can be tolerated by the wireless sensor network is application specific.

However, data transferred over a wireless sensor network is more susceptible to loss than that over wired networks. This is because in wired networks data loss occurs primarily due to congestion in the network while additional reasons for data loss exist in wireless sensor

networks. The reasons for data loss in wireless sensor networks are stated below.

1.3.1 Collision between Transmitting Nodes

Data loss will occur due to collisions between data transmitted by two or more nodes at the same time in a wireless sensor network. Hence, a MAC or media access protocol is necessary to determine which nodes will have sole access of the medium in order to transmit data and thus, reduce the number of collisions. There is a brief discussion of the different MAC protocols in the network communication model in Section 1.1.2. However, these protocols cannot completely eliminate collisions; they can only reduce them.

1.3.2 Congestion

Data loss due to congestion can always occur in wireless sensor networks. When the buffer capacity at a node is exceeded, congestion occurs and the node drops data. A network link is said to be congested when the offered load on the link reaches a value close to the capacity of the link.

Congestion can occur due to a variety of reasons. First, some node may detect events like a forest fire or an earthquake and send large amounts of data suddenly in addition to the data that is normally sent across the network. Second, new nodes may be added to the network which may collect and send additional data thus adding to the traffic load. Third, nodes may die increasing the load on other parts of the network. When large amounts of data are being transferred from sensor nodes to base station, nodes close to the base station may face congestion because they are few in comparison to the nodes sending data to them. Another reason may be that other reasons for data loss like collisions slow the data delivery rate at a node leading to the gradual build up of data at the node and hence, to congestion. Routing through different nodes because some nodes are currently in the sleep mode can also cause congestion. In wireless sensor networks, congestion causes overall channel quality to degrade, leads to buffer drops at nodes and hence, data loss which in turn lowers throughput and wastes energy.

1.3.3 Barriers to Electromagnetic Signals

Both radio/light waves used for transmission in wireless sensor networks are electromagnetic radiation. Electromagnetic radiation can travel through various media to varying degrees, depending on the frequency and the kind of media like solids and fluids. For example

- Light waves can travel through air, water and glass but not other solid material.
- Radio waves can travel through some solids but not through metal.

As such, presence of different types of solid objects in the path of wireless transmission can result in data loss.

1.3.4 Environmental Disturbances

Electromagnetic signals can also fade because of environmental conditions like thunderstorms, earthquakes, etc. They can be absorbed due to atmospheric gases or dielectric state of the atmosphere. Likewise magnetic and electrical effects of the earth's surface and man-made machinery can affect signals. All these are responsible for signal/data loss in wireless sensor networks.

1.3.5 Dead Nodes, Mobile Nodes and Human Interference

In addition to the above, node failure may occur due to energy loss or human interference in wireless sensor networks e.g. if the military of a country places a wireless sensor network in an enemy territory, the enemy may discovery a sensor node and deactivate it. If a node sends data to a failed node, data loss occurs as a result. Also, a node may try to send data to a mobile neighbor which has moved away resulting in data loss.

1.4 Problem Definition: An Application-Specific Reliable Data Transfer Requirement in WSNs

We consider the problem of implementing reliability during data transfer from a base station to sensor nodes for time-critical applications with zero tolerance for data loss like the transfer of instructions from a base station to sensor nodes in order to reprogram them to perform

a task different from the one they are executing currently or to change the parameters of detection of their current task, etc.

Such applications have no tolerance for data loss because for example, if any part of the instructions is lost, the sensor nodes will not be able to interpret the instructions. The application is delay-sensitive because if there is a delay in the delivery of the instructions, the sensor nodes may not be able to capture the data required by the base station. In addition, sensor nodes are energy constrained, thus the number of messages exchanged in the wireless sensor network to implement reliability must be minimum as it is a measure of the energy used by the sensor nodes.

The problem is defined as below:

- A source node s has to deliver data to destinations d which are at a distance of h hops from s where a node x which is within the transmission range of another node y is said to be at a distance of one hop from y.
- Probability of successful transmission of a message between any two nodes seperated by one hop is P.
- Each destination d should receive the entire data.
- Delivery should take minimum time and minimum number of messages.

The minimum time and minimum number of messages mentioned in the problem definition implies that a new solution to the problem should take less/equal time and number of messages in comparison to previous solutions.

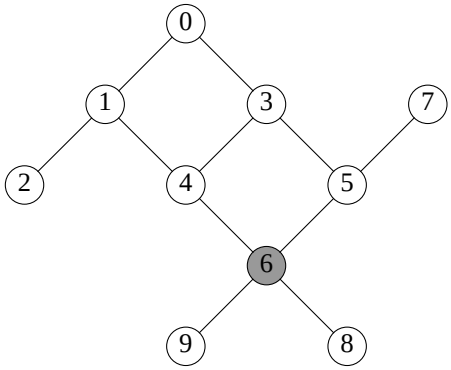


Figure 1.16: A 10-node network for demonstrating the problem discussed in this dissertation.

In the network in Figure 1.16, node 6 is the source and it has to send 5 data packets $\{p_0, p_1, p_2, p_3, p_4\}$ to all nodes 3 hops away from it. The probability of successful transmission of a message between any two nodes seperated by 1 hop is P. The destination nodes are nodes 0 and 2 respectively. All data packets should reach the destination nodes taking minimum time and minimum number of messages.

In the practical scenario, it is not possible to guarantee that all nodes which are at a distance of 3 hops away will receive the data. This is because no node is aware of the entire network topology. However, whenever a node is determined to be at a distance of 3 hops, it should receive all the data packets.

In the following chapters, we give a summary of the existing solutions and their drawbacks and then present our solution and its evaluation.

Chapter 2
Literature Review

Here, we give a brief description of different reliable data transfer protocols in wireless sensor networks and the problems with their implementation for time-critical applications with zero-tolerance for data loss in wireless sensor networks.

2.1 TCP: Transport Control Protocol

The transmission control protocol (TCP) [16][17] is the most predominant transport layer protocol in the Internet today. It transports more than 90% percent of the traffic on the Internet. Its end-to-end congestion control mechanism, byte-stream transport mechanism, and, above all, its elegant and simple design have not only contributed to the success of the Internet, but have also made TCP an influential protocol in the design of many of the other protocols and applications. Its adaptability to the congestion in the network has been an important feature in times of extreme congestion. TCP in its traditional form was designed and optimized only for wired networks.

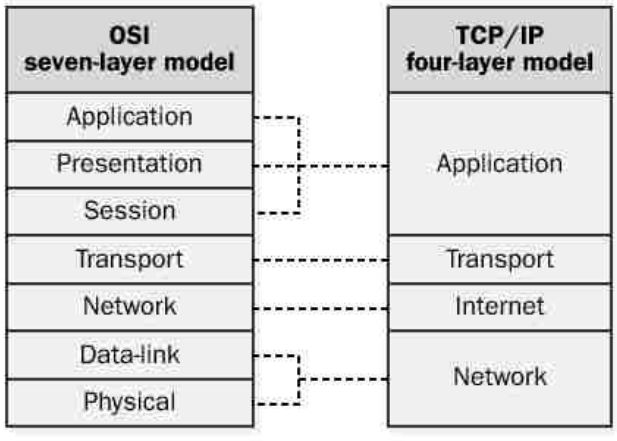


Figure 2.1: TCP/IP protocol stack.

Since TCP is widely used today and the efficient integration of a wireless network with the Internet is paramount wherever possible, it is essential to have mechanisms that can improve TCP's performance in wireless networks. This would enable the seamless operation of application-level protocols such as FTP, SMTP, and HTTP across the integrated wireless networks and the Internet.

TCP is a reliable, end-to-end, connection-oriented transport layer protocol that provides a byte-stream-based service [the stream of bytes from the application layer is split into TCP segments, the length of each segment limited by a maximum segment size (MSS)]. The major responsibilities of TCP include congestion control, flow control and in-order delivery of packets. Congestion control deals with excess traffic in the network which may lead to degradation in the performance of the network, whereas flow control controls the per-flow traffic such that the receiver capacity is not exceeded. TCP regulates the number of packets sent to the network by expanding and shrinking the congestion window. The TCP sender starts the session with a congestion window value of one MSS. It sends out one MSS and waits for the ACK. Once the ACK is received within the retransmission timeout (RTO) period, the congestion window is doubled and two MSSs are originated. The doubling of the congestion window with every successful ACK of all the segments in the current congestion window continues until the congestion window reaches the slow-start threshold (the slow-start threshold has an initial value of 64 KB).

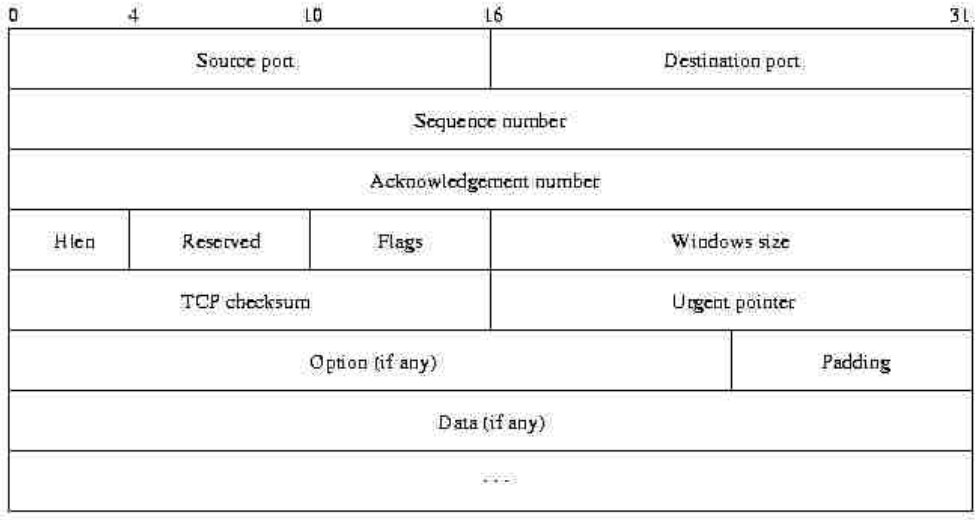


Figure 2.2: TCP header format.

Once it reaches the slow-start threshold, it grows linearly, adding one MSS to the congestion window for every ACK received. This linear growth, which continues until the congestion window reaches the receiver window (which is advertised by the TCP receiver and carries the information about the receiver's buffer size), is called congestion avoidance, as it tries to avoid increasing the congestion window, which will surely worsen the congestion in the network. TCP updates the RTO period with the current round-trip delay calculated

on the arrival of every ACK packet. If the ACK packet does not arrive within the RTO period, then it assumes that the packet is lost. TCP assumes that the packet loss is due to the congestion in the network and it invokes the congestion control mechanism. The TCP sender does the following during congestion control: (i) reduces the slow-start threshold to half the current congestion window or two MSSs whichever is larger, (ii) resets the congestion window size to one MSS, (iii) activates the slow-start algorithm, and (iv) resets the RTO with an exponential back-off value which doubles with every subsequent retransmission. The congestion window is doubled with every successfully acknowledged window and, upon reaching the slow-start threshold, it enters into the congestion avoidance phase.

The TCP sender also assumes a packet loss if it receives three consecutive duplicate ACKs (DUPACKs) which are repeated acknowledgments for the same TCP segment that was successfully received in-order at the receiver. Upon reception of three DUPACKs, the TCP sender retransmits the oldest unacknowledged segment. This is called the fast retransmit scheme. When the TCP receiver receives out-of-order packets, it generates DUPACKs to inform the TCP sender about the sequence number of the last in-order segment received successfully.

Some of the reasons behind the throughput degradation that TCP causes when used in wireless networks are:

- **Misinterpretation of packet loss.** Traditional TCP was designed for wired networks where the packet loss is mainly attributed to network congestion. Network congestion is detected by the sender's packet RTO period. Once a packet loss is detected, the sender node assumes congestion in the network and invokes a congestion control algorithm. Wireless sensor networks experience a much higher packet loss due to factors such as increased collisions due to the presence of hidden terminals, environmental disturbances, human interference, frequent path breaks due to mobility of nodes, and the inherent fading properties of the wireless channel.
- **Misinterpretation of congestion window.** TCP considers the congestion window as a measure of the rate of transmission that is acceptable to the network and the receiver. In wireless sensor networks, the congestion control mechanism is invoked by

TCP when the network gets partitioned or when a path break occurs. This reduces the congestion window and increases the RTO period. When the route is reconfigured, the congestion window may not reflect the transmission rate acceptable to the new route, as the new route may actually accept a much higher transmission rate. Hence, when there are frequent path breaks, the congestion window may not reflect the maximum transmission rate acceptable to the network and the receiver.

- **Multipath routing.** There exists a set of QoS routing and best-effort routing protocols that use multiple paths between a source-destination pair in a wireless sensor network. There are several advantages in using multipath routing. Some of these advantages include the reduction in route computing time, the high resilience to path breaks, high call acceptance ratio, and better security. For TCP, these advantages may add to throughput degradation. These can lead to a significant amount of out-of-order packets, which in turn generates a set of duplicate acknowledgments (DUPACKs) which cause additional power consumption and invocation of congestion control.
- **Power and bandwidth constraints.** Nodes in wireless sensor networks face resource constraints including the two most important resources: (i) power source and (ii) bandwidth. The performance of TCP is significantly affected by these constraints.

It is evident that a solution other than TCP is required for wireless sensor networks. This solution should emphasize more on recovering or reducing from data loss due to various other reasons than only congestion control.

2.2 PSFQ: Pump Slowly, Fetch Quickly

The PSFQ protocol [14] is a transport layer protocol designed to deliver data without loss in wireless sensor networks. It uses a hop-by-hop lost data recovery method (the definition of one hop is given in Section 1.4) with non-acknowledgement of data packets at receiving nodes and in-sequence data packet forwarding at nodes to ensure reliability. In a hop-by-hop data recovery method, intermediate nodes(i.e. nodes between source and destination) take part in lost data recovery. Each node has a data cache to store all the data packets that it receives. A data packet p_i will have the sequence number i and the highest packet sequence

number m sent by the source. This information helps a node x to detect packets missing in its data cache on receiving p_i. The protocol consists of three operations: a pump operation, a fetch operation and a report operation.

In the **pump** operation, the source transmits data packets one by one every T_{min}. Each packet has a sequence number. All other nodes behave as follows:

- When a node x receives a packet p_i not present in its data cache, it stores p_i in the data cache. If there is no missing packet with sequence number lower than p_i and all such packets have been transmitted at least once, then x waits a random period (to avoid collisions) between T_{min} and T_{max} before broadcasting p_i. There is a delay of at least T_{min} between each pair of consecutive packets that x broadcasts. However, this broadcast is suppressed if x finds that four or more of its neighbors have already broadcast p_i, because then the expected additional coverage achieved by x by broadcasting p_i tends to be small. If p_i is already in the data cache, x drops the packet.
- When x receives a packet out-of sequence i.e. p_{i+1} is received where p_i has never been received, p_{i+1} is stored in the data cache if it not present there already but instead of broadcasting it, x requests retransmission of all missing packets with sequence numbers lower than p_{i+1} from its neighbors using a request for retransmission or NACK message indicating the missing packets. The node x will resend the NACK every $T_r < T_{max}$ interval (with slight randomization to avoid collisions between neighbors). As soon as x receives the missing packets, it starts broadcasting the packets in-sequence i.e. p_i before p_{i+1} in the pumping mode.

The decision to broadcast packets only in-sequence has the advantage that loss events do not propagate. To illustrate the advantage of in-sequence transmissions, let us suppose there are no in-sequence transmissions at nodes. Let node x send packets p_i and p_{i+1} to downstream neighbor node y. Packet p_i is lost during transmission. Node y receives p_{i+1} and sends it to downstream node z. Node z on receiving p_{i+1} detects missing p_i and sends a request for it to y which cannot respond since it does not have p_i. This is called propagation of loss and this request from z to y is an unnecessary message which is costly for energy-constrained wireless sensor networks.

The **fetch** operation corresponds to a retransmission request and is triggered by detection of missing packets at nodes. However, a node x will not detect a missing packet p_i unless it receives p_{i+1} or any packet with sequence number higher than i. Hence, if p_{i+1} or any packet with a higher sequence number is not received, the missing p_i will not be detected. PSFQ deals with such cases by having x proactively trigger the fetch operation by sending the request for retransmission or NACK if no new packet is delivered after a period of time $T_{pro} = \alpha * (p_m - p_{i-1}) * T_{max}$ where m is the highest packet sequence number sent by the source and $(i-1)$ is the last successfully received packet and α is dependent on the network channel error rate.

The **report** operation is initiated by the source. The source sends a report request on receipt of which a destination node initiates and sends a report message at a random time between (0, Δ) and each node which receives a report message appends its own report to the received report message and forwards it further. A report message for a node x has its node id and the packets received/missing at x. If x knows that the source has requested a report message but does not receive a report after $T_{report} = T_{max} * TTL + \Delta$ where TTL is proportional to the shortest distance between source and destination, x initiates a report message and forwards it to its neighbors.

However, in-sequence transmission of packets can delay the delivery of packets to destination in comparison to out-of-sequence packets as discussed later Chapter 3. Also, if there are no missing packets at a node, the slow pump operation will delay the transmission of packets unnecessarily.

2.3 RMST: Reliable Multi-Segment Transport

The authors [18] implement reliability methods at different layers of wireless sensor networks namely MAC layer, transport layer and application layers. RMST is a transport layer protocol providing reliability which acts with Directed Diffusion as the routing protocol.

Link layer Automatic Repeat Request (ARQ) is an error control method for data transmission which makes use of acknowledgments and timeouts to achieve reliable data transmission. An acknowledgment is a message sent by the receiver to the transmitter to

indicate that it has correctly received a data frame. Usually, when the transmitter does not receive the acknowledgment before the timeout occurs (i.e. within a reasonable amount of time after sending the data frame), it retransmits the frame until it is either correctly received or the error persists beyond a predetermined number of retransmissions. The following are the different methods for implementing reliability at the MAC layer:

- **No ARQ.** Such transmissions do not employ MAC layer reliability mechanisms such as RTS/CTS and ACK. In this mode, reliability is completely deferred to the transport or application layer. There are several possible benefits to this scheme. Firstly, there is a significant amount of overhead connected with the exchange of RTS/CTS and ACK packets that is avoided. Second, routing protocols like diffusion attempt to select high quality (lower error rate) paths for data transmission. The ARQ reliability mechanisms can make poor paths mistakenly look reliable to higher layers.
- **ARQ always.** This transmission method utilizes RTS/CTS and ACK with retries to bolster reliability. When a node wishes to communicate with multiple neighbors, each neighbor must be sent a unicast packet. The number of ARQ retransmissions attempted before giving up is configurable. This method also has certain benefits for sensor networks. Data that travel on the links identified in route discovery will be delivered with a high degree of reliability
- **Selective ARQ.** This method attempts to combine the benefits of both ARQ and No ARQ. Data and control packets, in unicast transmissions, use ARQ to bolster reliability. Data sent to multiple neighbors have no ARQ. Data used in route-discovery are broadcast to all neighbors without ARQ.

The following are the different methods for implementing reliability at the transport layer:

- **End-to-End Selective Request NACK.** The detection of missing data and the request for the same (NACK) takes place only at the base station. The requests for missing data travel from the base station to the source of the data, where the missing data is retransmitted.
- **Hop-by-Hop Selective Request NACK and Repair from Cache.** Each caching node on a path from the source of the data to the base station caches the fragments

that make up a larger data entity. When such nodes sense a missing fragment, a request (NACK) for them is sent to the next hop on the path toward the source. If the requested fragment is in the local cache, a response is sent. If not, the NACK is forwarded to the next hop toward the source.

The following is a method for implementing reliability at the application layer:

- **End-to-End Positive ACK.** In this approach a base station requests to receive a large data entity, which is fragmented at the source. When all fragments have arrived at the base station, it deletes its request. Sources send the entire set of fragments at precalculated intervals until request is deleted.

RMST was designed to operate with Directed Diffusion [6] as the routing protocol. Reliability in RMST refers to the eventual delivery of all fragments related to a unique RMST entity to the subscribing base station. A unique RMST entity is a data set consisting of one or more fragments coming from the same source. An unfragmented data entity must have an application specific attribute (RmstNo) or set of attributes that serves to distinguish a particular reliable flow of data during transfer from a source to the base station. Each fragment that makes up a fragmented data entity must also contain a sequential fragment id (FragNo). The total number of fragments that make up a data entity must be known (MaxFrag). There is a single control message generated by RMST, the NACK, which must be defined by an attribute.

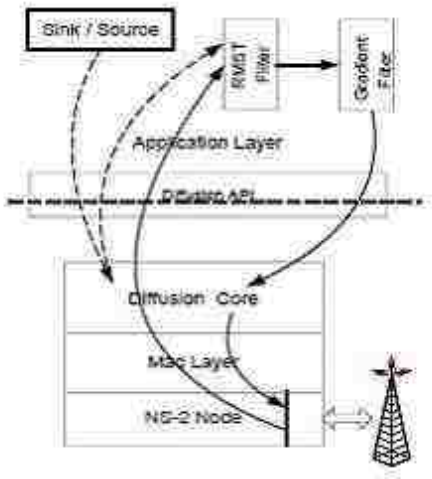


Figure 2.3: RMST protocol.

In RMST, receivers are responsible for detecting whether or not a fragment needs to be sent again. The term receiver here, however, does not necessarily mean the base station. In the non-caching mode, only the base station monitors the integrity of an RMST entity in terms of received fragments. In caching mode, an RMST node collects fragments and is capable of initiating recovery for missing fragments to the next node along the path toward the source. There are two types of loss detected by a receiver:

- A hole in a sequence of fragments. When a hole in a sequence of fragments is detected, the missing fragments should be specifically requested.
- A truncated sequence. The truncation of a sequence is really a special case of a hole, sensed by the receiver via a timeout geared to the expected receipt time of the next fragment.

In non-caching mode, only the base station set timers to detect loss. In caching mode, each caching node on the reinforced path from the source to the base station detects loss. The timer handler inspects the hole map and sends a NACK for any holes that have aged for too long. Multiple hole numbers are aggregated into a single NACK to conserve on control traffic.

The only control message added to normal diffusion by RMST is the NACK. NACKs are unicast in the reverse direction along the reinforced path from the source to the base station. When a node gets a cache hit for a NACKed fragment, it unicasts that fragment to the requesting neighbor. When a node intercepts a NACK, and it cannot find the missing fragment in its local cache (or its not in caching mode), it forwards the NACK on the reinforced path toward the source. In caching mode, the natural progression of traffic from the source to the base station, causes holes to be sensed sooner upstream, thus making NACK forwarding an unlikely event.

In caching mode, a node maintains a local cache of traffic in progress or recently transmitted. In non-caching mode, only the source and the base station maintain a cache. The cache is indexed by the application specific attribute. Each cache entry has an associated fragment map and hole map. The fragment map contains the actual cached data indexed by the fragment id. The hole map is a list that contains missing or overdue fragments

for a particular flow. Hole map entries contain a fragment id, a timestamp indicative of when a NACK for this fragment was sent, and a flag indicating whether or not a NACK is outstanding.

The RMST paper concludes based on simulations that MAC layer ARQ is better than no ARQ. It states that a NACK based transport layer running over a selective-ARQ MAC layer is an appropriate solution. However, RMST does not state how a node will give priority when sending different types of messages it has at a particular time unit nor does it mention the order in which packets are to be forwarded (in-sequence or out-of-sequence) at nodes.

2.4 ESRT: Event-to-Sink Reliable Transport

The ESRT [19] protocol aims to achieve reliable event detection in wireless sensor network minimising energy expenditure in the process. It includes a congestion control component for achieving reliability. It is implemented at the transport layer. The protocol is designed for wireless sensor network applications where sensor nodes gather information from a detected event and send it to the base station and the base station is interested in the collective information obtained from the sensor nodes in the region where the event occured.

Let τ be the duration of a decision interval by which the base station must detect event features. The amount of information received by the base station for the detection and extraction of event features is measured in terms of data packets. Let r_i be the number of data packets received by the base station in the decision interval i and R be the number of data packets required for detection and extraction of event features in a decision interval. If $r_i > R$, then the event features can be reliably detected else appropriate action needs to be taken to achieve the desired reliability R. Packet loss can occur due to link errors, node failures, congestion, etc.

The congestion control component of the protocol involves detection and removal of congestion. The reporting rate f of a sensor node is defined as the number of packets sent out per unit time by that node. A sensor node measures the size of the current contents of its buffer at the end of every reporting period $1/f < \tau$. If it exceeds the maximum buffer size,

the sensor node infers that it is going to experience congestion in the next reporting period. The problem solved by ESRT in a wireless sensor network is to configure the reporting rate f of the sensor nodes so as to achieve the required event detection reliability R at the base station minimising energy consumption and removing congestion, if any.

ESRT achieves this by having the base station handle the task of informing the sensor nodes to adjust their reporting rate f when $r_i < R$ and/or there is congestion. The authors assume that the base station is powerful enough to reach all source nodes by broadcast. It has been already explained before how the base station measures reliability. We now explain how it detects congestion A sensor node on detecting congestion sets the CN (Congestion Notification) bit in the header of the packets it transmits to the base station to notify it of the upcoming congestion condition to be experienced in next reporting period. When the base station receives packets whose CN bit is marked, it infers that congestion is experienced in the last decision interval.

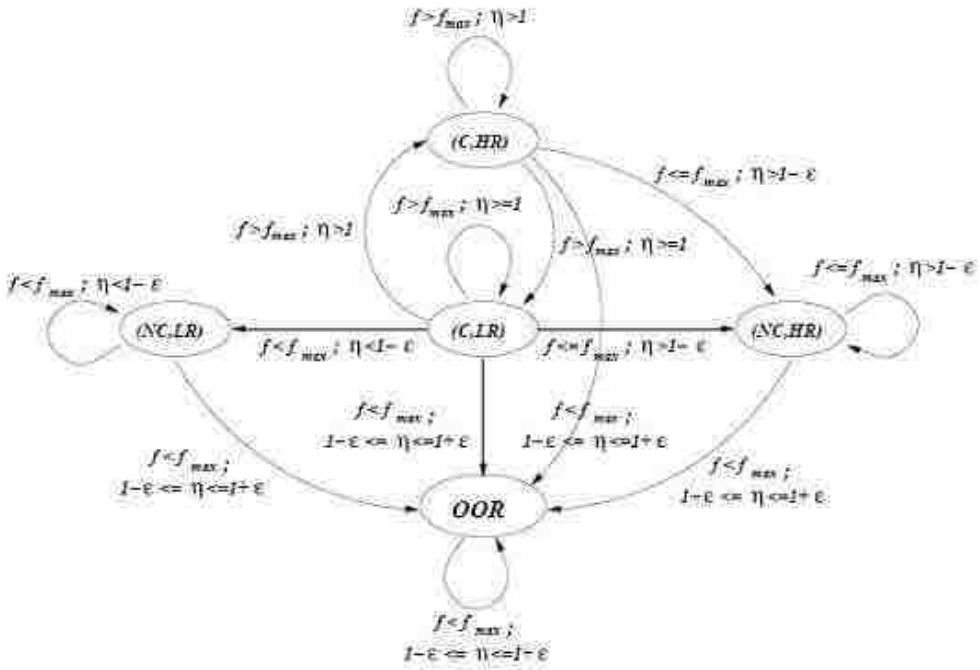


Figure 2.4: ESRT finite state machine.

The authors perform simulations and observe that the reliability achieved shows a linear increase (plotted along the log scale) with reporting rate f until a certain $f = f_{max}$, beyond which the reliability drops. This is because of network congestion. Let η be a measure of the reliability. The aim of the protocol is to have the value of η as close to 1

as possible. Based on their simulations, the authors identify 5 states of the wireless sensor network:

- (NC,LR) : $f < f_{max}$ and $\eta < 1 - \epsilon$ (No Congestion, Low Reliability)
- (NC,HR) : $f \leq f_{max}$ and $\eta > 1 + \epsilon$ (No Congestion, High Reliability)
- (C,HR) : $f > f_{max}$ and $\eta > 1$ (Congestion, High Reliability)
- (C,LR) : $f > f_{max}$ and $\eta \leq 1$ (Congestion, Low Reliability)
- (OOR) : $f < f_{max}$ and $1 - \epsilon \leq \eta \leq 1 + \epsilon$ (Optimal Operating Region)

The desirable state is OOR. The following actions are taken by the protocol when the network is in each of the states 1 to 4.

- (NC,LR) : The base station instructs the sensor nodes to increase the reporting rate using the following formula

  $$f_{i+1} = \frac{f_i}{\eta_i}$$

  where η_i is the reliability observed at the base station at the end of decision interval i.
- (NC,HR) : In order to conserve energy, the base station instructs the sensor nodes to reduce the reporting rate according to the formula:

  $$f_{i+1} = \frac{f_i}{2}\left(1 + \frac{1}{\eta_i}\right)$$

- (C,HR) : The base station instructs the sensor nodes to decrease the reporting rate using the following formula

  $$f_{i+1} = \frac{f_i}{\eta_i}$$

  where η_i is the reliability observed at the base station at the end of decision interval i.
- (C,LR) : This is the worst possible state and more aggressive reduction of the reporting rate is required in order to reduce congestion and improve reliability. The base station instructs the sensor nodes to decrease the reporting rate using the following formula

  $$f_{i+1} = f_i^{(\eta_i/k)}$$

where η_i is the reliability observed at the base station at the end of decision interval i and k denotes the number of successive decision intervals for which the network has remained in state (C,LR) including the current decision interval.

ESRT deals with loss due to congestion only in wireless sensor networks whereas there are additional reasons for data loss in wireless sensor networks. It assumes that the base station is powerful enough to reach all source nodes by broadcast and does not consider data recovery methods in order to achieve reliability. Hence it is unsuitable for applications having zero tolerance for data loss.

2.5 CODA: COngestion Detection and Avoidance

CODA is an energy efficient congestion control scheme for wireless sensor networks that comprises three mechanisms: receiver-based congestion detection, open-loop hop-by-hop backpressure, and closed-loop multi-source regulation.

An application driving the development of wireless sensor networks is the reporting of conditions within a region where the environment abruptly changes due to an observed event, such as in habitat monitoring, target detection, earthquakes, floods, or fires. Here, wireless sensor networks typically operate under light load and then suddenly become active in response to a detected or monitored event. Depending on the application this can result in the generation of large, sudden, and correlated impulses of data that must be delivered to a base station. Although a wireless sensor network may spend only a small fraction of time dealing with impulses, it is during this time that the information it delivers is of greatest importance. The transport of event impulses is likely to lead to varying degrees of congestion in sensor networks. A number of distinct congestion scenarios are likely to arise.

- Densely deployed sensors generating impulse data events will create persistent hotspots proportional to the impulse rate beginning at a location very close to the sources (e.g., within one or two hops).
- Sparsely deployed sensors generating low data rate events will create transient hotspots potentially anywhere in the sensor field but likely farther from the sources, toward the base station.

- Sparsely deployed sensors generating high data-rate events will create both transient and persistent hotspots distributed throughout the sensor field.

CODA provides a solution for all these cases. CODA employs CSMA as a MAC protocol. Though TDMA can strictly control and schedule traffic and flows in the network and thus alleviate the need for congestion control, it has a lot of synchronization and scheduling overhead in wireless sensor networks. Hence, CODA uses CSMA for medium access. CODA also suggests link-layer ARQ for improving reliability. The reason is that CODA does not offer any solution for collisions or data recovery. It only offers reliable congestion detection and control mechanisms.

The methods used by CODA for **congestion detection** are:

- A nearly overflowing queue.
- The base station expects a certain event sampling rate or reporting rate coming from the sources. This rate is highly application-specific. When a base station consistently receives less than desired reporting rate, it can be inferred that packets are being dropped along the path, most probably due to congestion.

The methods used by CODA to **control congestion** are:

- **Open-loop hop-by-hop backpressure.** Once congestion is detected, the receiver will broadcast a suppression message to its upstream neighbors and at the same time make local adjustments to prevent propagating the congestion downstream. A node broadcasts backpressure messages as long as it detects congestion. Backpressure signals are propagated upstream towards the sources of the event. When an upstream node (toward the source) receives a backpressure message, based on its own local network conditions it determines whether or not to further propagate the backpressure signal upstream. For example, a node may simply start to drop its incoming data packets upon receiving a backpressure message, preventing its queue from building up, rather than propagating the backpressure signal further upstream because of an overflowing queue. The term *depth of congestion* is used to indicate the number of hops that the backpressure message has traversed before a non-congested node is encountered. The

depth of congestion can be used by the routing protocol to help balance the energy consumed during congestion across different paths. It is better for a node to use congestion detection mechanisms only when it receives a packet as it will have less overhead.

- **Closed-loop multi-source regulation.** When the source event rate (r) is less than some fraction η of the maximum theoretical throughput (S_{max}) of the channel, the source regulates itself. When this value is exceeded ($r \geq S_{max}$), a source is more likely to contribute to congestion and therefore closed-loop control is triggered. The source only requires feedback from the base station if this threshold is exceeded. At this point, a source requires constant feedback (e.g., ACK) from the base station to maintain its rate (r). A source triggers base station regulation when it detects ($r \geq S_{max}$) by setting a regulate bit in the event packets it forwards toward the base station. Reception of packets with the regulate bit set forces the base station to send ACKs to regulate all sources associated with a particular data event. The reception of ACKs at sources would serve as a self-clocking mechanism allowing the sources to maintain the current event rate (r). When a source sets its regulate bit it expects to receive an ACK from the base station at some predefined rate, or better, a certain number of ACKs over a predefined period allowing for the occasional loss of ACKs due to transient congestion. If a source receives a prescribed number of ACKs during this interval it maintains its rate (r). When congestion builds up, ACKs can be lost forcing sources to drop their event rate (r) according to some rate decrease function (e.g., multiplicative decrease).

CODA like ESRT in Section 2.4 deals with loss due to congestion only in wireless sensor networks whereas there are additional reasons for data loss in wireless sensor networks. It does not consider data recovery methods in order to achieve reliability. Hence it is unsuitable for applications having zero tolerance for data loss.

2.6 DTNLite: Delay Tolerant Networking Lite

This idea [55] assumes the existence of an underlying multihop routing protocol. It is meant for applications which collect data and send it towards a single base station. It is based on a

paradigm called *custody transfer*. Only nodes which are *custodian* nodes will have the entire message to be sent from the source to the destination. A custodian node x will pass the entire message to another custodian node y and will delete the message from its buffer only after receiving an acknowledgement from y that it has received the entire message. The initial custodian node is the source. The custodian of a message sends a query asking for nodes that have a better local metric estimate than itself. The local metric estimate can be energy level remaining at a node, average energy consumption for message delivery, etc. The query packet is forwarded as far as possible and the path traversed is added to the query. Any node willing and eligible to assume custody sends a response to the query using the path in the query. Quality estimates of the traversed links are recorded in the response. The custodian node selects a successor from the candidate nodes based on the local metric estimate and the quality estimate of the path to a candidate. The message transmission from a custodian node to a potential custodian node takes place via a hop-by-hop delivery mechanism based on non-acknowledgement of packets and retransmissions. The idea of custodian nodes is to prevent unnecessary usage of memory space.

The selection of custodian nodes has a considerable overhead and since the wireless sensor network topology is dynamic, this overhead becomes considerable with changes in topology. This idea also does not give a solution to the problem of duplicate custodian nodes because of loss of acknowledgements. Let a current custodian node x send a message to a potential custodian node y. When y receives the entire message, it assumes custody and sends back an acknowledgment to x. If the acknowledgment does not reach x, x does not delete the message from its storage and renounce its custody. It will again try to send it to another potential custodian node which may be different from y as the parameters for evaluating candidate custodian nodes may have changed by then. So we have duplicate custodian nodes and unnecessary usage of memory space. Also in terms of reliable data delivery it does not introduce any new concept different from PSFQ. In fact, it does not mention how the packets are to be fowarded at nodes (in-sequence or out-of-sequence) and the priority order at nodes for sending different types of messages.

2.7 End-to-End Reliable Event Transfer in WSNs

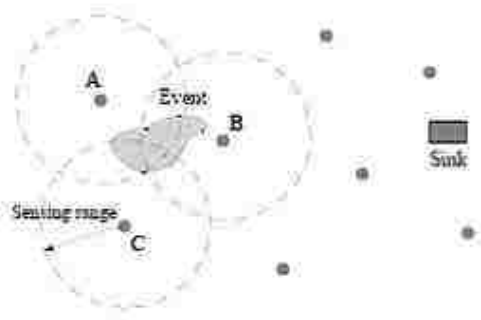


Figure 2.5: Event transfer.

This paper [57] discusses two schemes for implementing reliability in the delivery of events from sensor nodes to the base station.

- **Non-acknowledgement based scheme.** This is implemented in three different ways:
  - **Implicit acknowledgement.** When a upstream node x receives a packet p_i from a downstream neighbor y that it had already sent to y, it is an implicit confrmation that p_i has been successfully delivered. However, if it does not receive p_i, it does not mean that the packet has not been successfully received by y because the transmission of p_i from y to x may have been lost.
  - **Event reporting frequency.** This is the same method used in the ESRT protocol in Section 2.4. As mentioned before, this method deals with reducing congestion only.
  - **Node density.** In sensor networks, there are usually multiple nodes that have overlapping sensing regions. Hence, it is possible that multiple nodes collaborate to detect the same event. The number of nodes that report the same event has an impact on the end-to-end event transfer reliability. Higher end-to-end reliable event transfer rate can be achieved by increasing the number of nodes involved in a sensing task. Event loss probability decreases because more packets are sent for the same event. However, this does not guarantee 100% reliable data delivery to the destination.
- **Acknowledgement based scheme.** This is implemented in three different ways:

- **Selective acknowledgement.** In this method, a sensor node determines whether the data being reported is critical or not. There are various ways to determine this. One approach is using a threshold value. Sensor nodes and the base station can come to an agreement on a threshold value before deployment. The threshold value depends on the application. Then, a sensor node decides whether the measurement is critical or not by using the agreed threshold value. The acknowledgement mechanism is triggered for the critical data packets. Each data packet received by the base station is compared with the threshold value and then categorized as being critical or not. If a critical data packet is received, an acknowledgement packet is forwarded to the source sensor node immediately. If the source sensor node does not receive an acknowledgement packet within a predetermined timeout period, i.e., retransmission time, it retransmits the packet.
- **Enforced acknowledgement.** Here, the idea is almost the same as in the selective acknowledgement case. Not all of the packets but the data packets that carry critical data need to be acknowledged. However, the base station does not determine whether a data packet is critical or not. Instead, the source sensor node decides if a data packet is critical or not and marks the packets that carry critical data. The base station is supposed to acknowledge the marked packets. The source nodes retransmit unacknowledged marked packets after a predetermined retransmission period.
- **Blanket acknowledgement.** Multiple sensor nodes reporting the same event may be acknowledged by a single acknowledgement packet. In blanket acknowledgement, a single acknowledgement packet is broadcast for an event. A sensor node that receives an acknowledgement packet for the event, accepts that the data packet generated for the same event has been received by the base station successfully. Since the reception of the event is more important than the reception of every packet, this is an efficient acknowledgement mechanism for a wireless sensor network.

Use of end-to-end reliability method increases the probability of loss as described in Chapter 3. The acknowledgement based methods cannot be used for applications with zero tolerance

for data loss. The number of acknowledgements would be too huge and it would be a burdern for energy-constrained wireless sensor networks. The non-acknowledgement based methods do not guarantee 100% reliable data delivery here because they do not deal with recovery of data.

2.8 A WSN for Structural Monitoring

Wisden [58] is a wireless sensor network system for structural-response data acquisition. Wisden continuously collects structural response data from a multi-hop network of sensor nodes, and stores the data at a base station. In Wisden, nodes self-organize themselves into a routing tree rooted at the base station. Wisden uses both hop-by-hop and end-to-end lost packet detection and recovery to implement reliable data transfer.

Figure 2.6: Wisden.

Wisden implements a hop-by-hop non-acknowledgement based reliability scheme.

- Each source stores generated vibration data and then transmits the data to its parent in the route tree. Parents keep track of the sequence numbers of the packets that they receive, on a per source basis.
- A gap in the sequence number of sent packets indicates packet loss. Each node maintains a list of missing packets. When a loss is detected, a tuple containing a source ID and sequence number of the lost packet is inserted into this list. Entries in the missing packets list are piggybacked on outgoing transmissions.
- Nodes keep a small cache of recently transmitted packets, from which a child can send packets reported missing by its parent.

Lost packets are often recovered hop-by-hop, however, two factors necessitate end-to-end recovery. First, heavy packet losses can lead to large missing packet lists that might exceed the memory of the sensor nodes. Also, a topology change could cause loss of missing packet list information. For example, when a node selects a new parent, it will no longer respond to requests for missing packets from its previous parent. The end-to-end recovery scheme is essentially implemented in almost the same way as the hop-by-hop scheme. It leverages the fact that the base station has significantly more memory and can keep track of all missing packets. The base station attempts hop-by-hop recovery of a missing packet. When one of its children notices that it has seen a packet from the corresponding source, but does not have a cached copy of that packet, it adds that recovery request to its missing packets list. This request is propagated downward in this manner (using the same mechanisms described for hop-by-hop recovery) until it reaches the source. Since the source maintains the generated packets in its memory, it can repair the missing packet.

The idea here is similar to the idea in PSFQ. However, it does not mention how the packets are to be fowarded at nodes (in-sequence or out-of-sequence) and the priority order at nodes for sending different types of messages.

2.9 STCP: Sensor Transmission Control Protocol

STCP [59] provides a generic, scalable and reliable transport layer paradigm for sensor networks. Majority of STCP functionalities are implemented at the base station. Each node might be the source of multiple data flows with different characteristics such as flow type, transmission rate and required reliability. The authors mention that if the base station can reach all the nodes in a single hop, it would enhance the performance of the solution, but it is not a requirement.

Before transmitting the packets, the sensor nodes establish an association with the base station via a session initiation packet. The session initiation packet informs the base station of the number of flows originating from the node, the type of data flow, transmission rate and required reliability. When the base station receives the session initiation packet, it stores all the information, sets the timers and other parameters for each flow, and acknowl-

edges this packet. It is important for the source sensor node to wait for the acknowledgement to ensure that the association is established. The nodes can now start transmitting data packets to the base station.

Flows are of two types:

- **Continuous flows.** Since the base station knows the rate of transmission from the source, the expected arrival time for the next packet can be found when the base station receives a packet. The base station maintains a timer and sends a negative acknowledgement (NACK) if it does not receive a packet within the expected time. Sensor nodes retransmit packets only on receiving a NACK. No state information is maintained. The transmitted packets are buffered for possible retransmission. To prevent buffer overflow, a buffer timer is maintained, which periodically fires when the buffer size reaches a threshold and the buffer is cleared. If the source node does not receive a NACK, the packet must have reached the base station, unless the NACK is lost. So the base station maintains a record of all packets for which it has sent a NACK. If a packet arrives for which a NACK has been sent, the base station clears the corresponding entry from the record. The base station periodically checks this record and, if it finds an entry, it retransmits a NACK.
- **Event-driven flows.** In event-driven flows, the base station cannot estimate arrival times of data packets. Because of reliability requirement, positive acknowledgements (ACK) are used by the source to know if a packet has reached the base station. The source node buffers each transmitted packet till an ACK is received. When an ACK is received, the corresponding packet is deleted from the buffer. The nodes maintain a buffer timer that fires periodically. When the timer fires, packets in the buffer are assumed to be lost and are retransmitted.

In STCP, sensor nodes specify the required reliability for each flow in the session initiation packet. For continuous flows, the base station calculates a running average of the reliability. Reliability is measured as the fraction of packets successfully received. Even if the base station does not receive a packet within the expected time interval, it will not send a NACK if the current reliability satisfies the required reliability. The base station transmits

NACKs only when the reliability goes below the required level. For event-driven flows, the base station calculates reliability as a ratio of packets received to the highest sequence numbered packet received. The sensor node, before transmitting a packet, calculate the effective reliability assuming that the packet will not reach the base station. If the result is still more than the required reliability, the node does not buffer the packet, thus saving memory space.

Each STCP data packet has a congestion notification bit in its header. Every sensor node maintains two thresholds in its buffer: t_{lower} and t_{higher}. When the buffer reaches t_{lower}, the congestion bit is set with a certain probability. The value of this probability can be determined by an approach similar to that employed in random early detection (RED) by Floyd and Jacobson. When the buffer reaches t_{higher}, the node will set the congestion notification bit in every packet that it forwards. On receiving this packet, the base station informs the source of the congested path by setting the congestion bit in the acknowledgement packet. On receiving the congestion notification, the source will either route successive packets along a different path or slow down the transmission rate. It should be noted that the nodes rely on the network layer algorithm to find alternate routes.

STCP uses end-to-end reliability use of which increases the probability of loss as described in Chapter 3. The first method does not mention how the packets are to be fowarded at nodes (in-sequence or out-of-sequence) and the priority order at nodes for sending different types of messages. The authors clearly mention that PSFQ may be used to complement their solution which implies that they are aware of the incompleteness of their solution. The second method uses too many acknowledgements which would be a burder for energy-constrained wireless sensor networks.

2.10 A Bidirectional Reliable Transport Mechanism for WSNs

This mechanism [13] addresses both sensor-to-base station and base station-to-sensor reliable transport. Bidirectional reliability is achieved by transmitting acknowledgement (ACK) or non-acknowledgement (NACK) messages between the sink and essential sensors that cover

the entire sensing field. Essential nodes (E-nodes) are selected using a weighted-greedy algorithm based on the residual energy of sensors and rotated in time. There is also a distributed congestion control mechanism.

- **Base station-to-Sensor Reliable Query Transfer.** Reliable base station-to-sensor communication is provided using negative acknowledgements. Since the packets are sent in order by the base station, sensors can detect the lost packets by using sequence numbers in the query messages. We assume that each packet has a unique id and a sequence number. An NACK message is sent if a gap is detected, i.e., an out-of-sequence number. NACK is handled only in E-nodes. Other nodes simply ignore the messages. This is because reliability can be achieved by using the essential set. The last packet in a message has a Poll/Final (P/F) bit to indicate that it is the last packet. When an E-node receives a packet whose P/F bit is set, it knows that the last packet is received. It sends a single ACK to the base station indicating the entire message has been successfully received. Base station retransmits the last packet of a message repeatedly until it receives an ACK.
- **Sensor-to-Base Station Reliable Event Transfer.** Each E-node waits for an acknowledgement(ACK) for only the first packet that reports an event which is called event-alarm. When a new event occurs, an E-node decides whether it will report the event or not. If it does, it marks the first packet as an event-alarm by setting the event notification bit. Therefore, the base station sends ACK for packets which are marked as event-alarm. The E-nodes are responsible for waiting for the acknowledgement and retransmitting if necessary. This protocol assumes that congestion can occur because large amounts of data are reported in events. Thus, ACK monitoring of event-alarm messages can be used as an efficient way of congestion detection. Congestion control is handled by the E-nodes based on monitoring the ACK packets of event reports. If an ACK is not received during a timeout period by the E-node, congestion alarm messages are sent to non-essential sensors to reduce their sending rate. When ACK is received, congestion-safe message is announced to resume normal operation of the network.

The reliable data transfer method from a base station to sensor nodes does not mention whether packet broadcasts are done in-sequence or out-of-sequence or the priority order for sending different message types at nodes. The selection of the essential sensors which are used for the data recovery process also involves considerable overhead. This protocol uses end-to-end data recovery as opposed to hop-by-hop recovery. The superiority of hop-by-hop method over end-to-end method is shown in our protocol Chapter 3. Also the ACK-based method used here for data transfer from sensor nodes to a base station is unsuitable for applications with zero-tolerance for data loss as an ACK has to be generated for every data packet sent which will result in too many messages which is unsuitable for energy constrained sensor nodes.

2.11 A Scalable Approach for Reliable Downstream Data Delivery in WSNs

The scheme [15] developed in the GARUDA project addresses a similar problem as PSFQ, namely reliable transfer of data from a base station to sensor nodes. GARUDA uses a NACK based scheme and additionally ensures that the first data packet is delivered to all sensors using Wait-For-First-Packet(WFP) pulse. This has a higher amplitude and lower period compared to normal pulses. This solves the problem in NACK based schemes where a receiver needs to receive at least one packet in order to detect losses of other packets.

GARUDA constructs an approximation to the minimum dominating set of the wireless sensor network topology and the members of this set (called core members) act as recovery servers for downstream core members and neighboring non-core members. Only those nodes whose hop distances from the base station are integral multiple of three can be candidates for the core. All core members know at least one upstream (i.e. closer to the base station) core member from which they request retransmissions of packets.

The reliable delivery of data from the base station proceeds in two steps: first when core nodes recover missing packets from upstream core nodes, and then when the non-core nodes fetch the missing packets from their associated core members. GARUDA is based on out-of-order delivery of packets i.e. a node x can send a packet p_{i+1} before it sends packet

p_i. A core member x requests missing packets from its upstream core member y, but does this only when it knows that the missing packets are indeed available at y. To achieve this, node y includes in every forwarded packet a A-map or bitmap indicating the packets already present at y, and x can use this knowledge to suppress NACKs for packets missing at y. A non-core member associated with x suppresses all retransmission requests until x has all the packets present, indicated by a full bit-map.

It is not clear how the the WFP pulse is better than an acknowledgement for the first packet since there are retransmissions involved in both cases and in addition, WFP pulses propagate downstream leading to periodic WFP pulses from downstream nodes whose upstream neighbors have not yet received the first transmission of the first packet. Also the periodic WFP pulses may collide with data transmissions leading to loss of data. In addition, WFP pulses require twice the transmission power of normal pulses which is unsuitable for energy constrained wireless sensor networks.

If all packets sent by an upstream core node y except packet p_0 are lost (we assume that when y sent p_0, it did not have any other packet), then a downstream core node x will not know what packets y has since the A-maps sent along with the lost packets are also lost and will not request any packets even though y may have some of the packets. This can delay delivery of packets to nodes.

The construction and maintainence the hierarchical core structure adds substantial overhead and affects negatively the feasibility of the solution given the energy limitations of wireless sensor networks. Also the topology of wireless sensor networks is dynamic which would require the core construction process to be repeated from time to time. Since packets can arrive via longer and shorter paths, the placement of core nodes will be far from ideal. This too will lead to the core construction process to be repeated from time to time in a manner similiar to above. The examples below show that it is possible for the core structure to have two core nodes within a distance two hops of each other and an out-of-sequence packet forwarding which does not use the core heirarchy but performs better than GARUDA.

In Figure 2.7, node 0 is the source. The dashed line joining two nodes indicate that the transmission of the first packet p_0 failed along that link. Solid lines indicate successful transmission. Nodes 6, 7 and 9 have been elected as core nodes marked by "C". However,

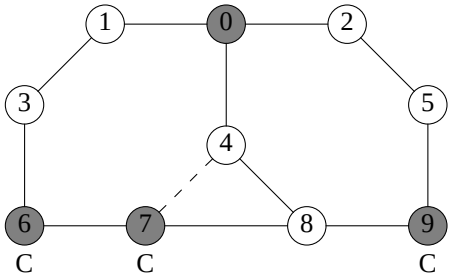


Figure 2.7: A 10-node network for illustrating a deficiency of the GARUDA protocol.

node 7 is not at a distance of 3 from the source. Hence, it is not supposed to be a core node but due to unsuccessful transmission of p_0 along the link joining the nodes 4 and 7, it has been elected as a core node. Thus instead of 2 core nodes, there are 3 core nodes. In this way for a bigger and dense network, unsuccessful transmissions due to various reasons can lead to an undesirable multiplicity of core nodes which results in more energy consumption.

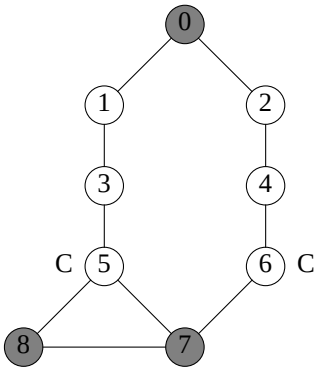


Figure 2.8: A 9-node network for illustrating a deficiency of the GARUDA protocol.

In Figure 2.8, node 0 has to transmit 2 packets p_0 and p_1 to nodes 7 and 8. Nodes 5 and 6 are **core** nodes. Node 5 has both packets while nodes 2, 4, 6 and 7 do not have p_0 but have p_1. Let node 6 be node 7's core node. According to GARUDA, node 7 can request p_0 from node 6 only. Hence, it has to wait for node 6 to receive the missing packet before requesting it. Node 6 will send an unicast message to node 0 to recover p_0. Since nodes 2, 4 do not have it either, so only node 0 can respond to the request. When node 6 finally receives p_0, it will send it to node 7. In out-of-sequence forwarding without the core heirarchy, node 7(a destination node) could have sent a broadcast for p_0 to nodes 5 and 6. Node 5 which has p_0 would have responded and node 7 would have received the complete packet set and executed the instructions faster in comparison to the GARUDA case.

Chapter 3

A New Protocol for Application-Specific Reliable Data Transfer in WSNs*

We propose a reliable data transfer protocol (OSDRMP) for delivery of data from a base station (source: s) to sensor nodes (destination: d) in a wireless sensor network for time-critical applications with zero-tolerance for data loss mentioned in Section 1.4. Each data set will have an unique id called data set id. We are at present considering the delivery of one data set which is divided into n number of packets p_i, $0 \leq i < n$ $(n \geq 1)$. The data set may be instructions for retasking sensors, queries, etc.

3.1 Key Features of the Protocol

- Non-acknowledgement of packets received at a node.
- Hop-by-hop detection and recovery of lost packets.
- Out-of-sequence forwarding of packets at a node.
- A priority order for sending different types of messages at a node.
- Delay in requesting packets missing at a node.

3.1.1 Non-acknowledgement of Packets Received at a Node

The alternative to non-acknowledgement based method is acknowledgement based method. We describe both methods and then the reasons for preferring the former method.

Both the acknowledgment(ACK) and non-acknowledgment($NACK$) based methods use three kinds of messages:

- Transmission of a packet p_i.
- Acknowledgment of a received packet p_i in ACK-based method or Request for a Missing Packet p_i in $NACK$-based method.

*A part of the research detailed in this chapter and the next has already been published. The paper is titled "Reliable and Efficient Data Transfer in Wireless Sensor Networks via Out-of-Sequence Forwarding and Delayed Request for Missing Packets" by Damayanti Datta and Sukhamay Kundu. It appears in the proceedings of ITNG '07. Publication Date: 2-4 April 2007, On page(s): 128-133, ISBN: 0-7695-2776-0, INSPEC Accession Number: 9465309, Digital Object Identifier: 10.1109/ITNG.2007.165.

- ReTransmission of a packet p_i.

We denote them respectively by $T(p_i)$, $ACK(p_i)$, $RMP(p_i)$, and $RT(p_i)$.

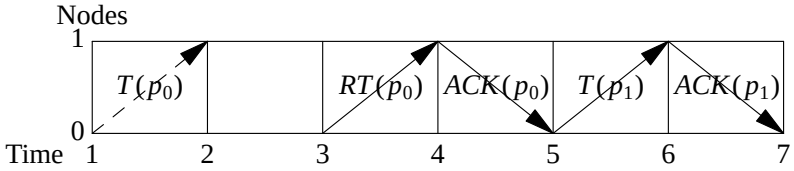
In ***ACK*-based method**, a node waits for an $ACK(p_i)$ from the receiver for a minimum time period t_a after sending a T(p_i). If it does not receive an $ACK(p_i)$, then it sends an $RT(p_i)$ and waits again for time t_a for an $ACK(p_i)$, and the process continues till an $ACK(p_i)$ is received. In this method, nodes should have distinct identification.

In ***NACK*-based method**, with each $T(p_i)$ or $RT(p_i)$, we include the total number of packets n so that a node can detect the missing packets once it receives a packet and send RMPs. A node which has detected a missing packet p_i waits for a time t_r before sending an $RMP(p_i)$. This delay is also kept between two successive RMPs for p_i from the same node. A node sending p_i in this case does not wait for an $ACK(p_i)$ following a $T(p_i)$ or $RT(p_i)$; instead, it sends an $RT(p_i)$ everytime it receives an $RMP(p_i)$. An $RMP(p_i)$ can be regarded as a *negative acknowledgment*($NACK$). Unlike the ACK-based method, this method does not require distinct identification for nodes but it cannot give 100% reliability if the successful transmission of a message between any two nodes seperated by 1 hop has probability less than 1.

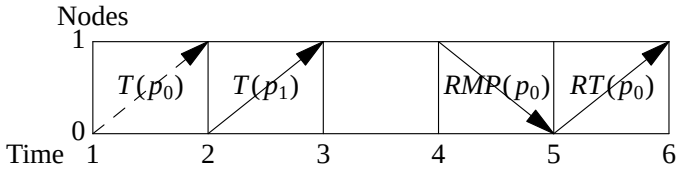
The following **example** shows the superiority of $NACK$-based method over ACK-based method. In Figure 3.2(a)-(b), the timing diagrams are shown for the transmission of $n = 2$ packets $\{p_0, p_1\}$ from node 0 to node 1 in the network in Figure 3.1 using ACK and $NACK$-based methods. We assume that only the first $T(p_0)$ is lost in both cases and all other messages are successfully sent in the first attempt. The dashed lines in Figure 3.2 show the lost transmissions. In Figure 3.2(b), the receiving node 1 does not know that p_0 is missing until it receives p_1. The ACK-based method takes 7 time units and 5 messages to deliver the two packets from node 0 to node 1 while the $NACK$-based method takes 6 time units and 4 messages. This example shows that the $NACK$-based method is better than the ACK-based method in terms of both the total delivery time and the total number of messages.

We do an analysis of the message efficiency for $NACK$-based vs. ACK-based methods in the network in Figure 3.1. Assume as in the above, that of the $n = 2$ packets $\{p_0, p_1\}$,

Figure 3.1: A 2-node network.



(a) *ACK*-based method with $t_a = 1$.



(b) *NACK*-based method with $t_r = 1$.

Figure 3.2: Comparison of $NACK$-based and ACK-based methods for delivery of 2 packets from nodes 0 to 1 in the network in Figure 3.1

p_1 is successfully delivered from node 0 to node 1 in one attempt in the network in Figure 3.1. Let P be the probability of successful transmission of a message between any 2 nodes seperated by 1 hop. Hence, the probability that k transmissions are required for one message to be successfully sent from node 0 to node 1 is

$$P(1-P)^{k-1}$$

and the expected number of transmissions is

$$= P + 2P(1-P) + 3P(1-P)^2 + \cdots = \frac{1}{P}$$

Let T_S and T_F denote respectively a successful and a failed transmission of a packet p_i; we use similar abbreviations for successful and failed RTs, RMPs and ACKs.

We compute the **number of messages sent** in $NACK$-based vs. ACK-based methods for $n = \mathbf{2}$ $\{p_0,p_1\}$ packets. In the ACK-based method, the following are the possible message sequences for the delivery of p_0 and $ACK(p_0)$:

- $T_S(p_0)$ from node 0 to 1 followed by an $ACK_S(p_0)$ or an $ACK_F(p_0)$ from node 1 to

node 0.

- $T_F(p_0)$ from node 0 to 1 (with no following $ACK(p_0)$).

When $T_S(p_0)$ is followed by $ACK_S(p_0)$, there will be no more messages. But there will be additional $RT(p_0)$s and $ACK(p_0)$s for the cases of $T_S(p_0)$ followed by $ACK_F(p_0)$ and $T_F(p_0)$. Let $E(p_0)$ be the expected number of messages required for the delivery of p_0 and $ACK(p_0)$. Then,

$$E(p_0) = P^2.2 + P(1-P)[2 + E(p_0)] + (1-P)[1 + E(p_0)]$$

and hence,

$$E(p_0) = \frac{1+P}{P^2} \tag{3.1}$$

Since we assume node 0 delivers p_1 in one attempt, the expected number of messages $E_{ACK}(p_1)$ for the delivery of $ACK(p_1)$ is

$$E_{ACK}(p_1) = P.1 + (1-P)[1 + \frac{1}{P} + E_{ACK}(p_1)]$$

and hence,

$$E_{ACK}(p_1) = \frac{1}{P^2} \tag{3.2}$$

Therefore,

$$E(p_1) = \frac{1+P^2}{P^2} \tag{3.3}$$

So the expected number of messages for delivery of $\{p_0, p_1\}$ and their acknowledgments is

$$E(p_0) + E(p_1) = \frac{2+P+P^2}{P^2} \tag{3.4}$$

In the *NACK*-based method, after $T_F(p_0)$ and $T_S(p_1)$ from node 0 to node 1, node 1 detects that p_0 is missing and sends an $RMP(p_0)$. The computation of the expected number of messages $E'(p_0)$ for the delivery of p_0 is slightly more complex now. With probability P, p_0 is delivered via the initial $T_S(p_0)$. If the first $T(p_0)$ fails, then this will be followed by an average $\frac{1}{P}$ many $RMP(p_0)$s from node 1 to node 0 till the latter receives the request for p_0 and from that point on there will be an additional $E'(p_0)$ many messages for the delivery of

p_0. Thus,

$$E'(p_0) = P.1 + (1-P)[1 + \frac{1}{P} + E'(p_0)]$$

and hence,

$$E'(p_0) = \frac{1}{P^2} \tag{3.5}$$

Since the expected number of messages for the delivery of p_1 ($E'(p_1)$) is 1, the expected number of messages for delivery of $\{p_0, p_1\}$ here is

$$E'(p_0) + E'(p_1) = \frac{1+P^2}{P^2} \tag{3.6}$$

The nodes in the ACK-based method therefore send on an average $\frac{1+P}{P^2}$ extra messages for $n = 2$ packets.

We compute the **number of messages received** in the $NACK$-based vs. ACK-based methods for $n = \mathbf{2}$ packets. We do a similar analysis to determine the average number of messages received by nodes 0 and 1. For a particular message sequence, the number of messages sent and received will vary. For example, in the ACK-based method,

- $T_S(p_0)$ from node 0 to node 1 is followed by an $ACK_S(p_0)$ or an $ACK_F(p_0)$ from node 1 to node 0. When $T_S(p_0)$ is followed by $ACK_F(p_0)$, two messages are sent but only one is received.
- $T_F(p_0)$ from node 0 to node 1 (with no following $ACK(p_0)$). Here one message is sent but none is received.

Let $E_R(p_0)$ be the expected number of messages received for delivery of p_0 and $ACK(p_0)$ in the ACK-based method. Then,

$$E_R(p_0) = P^2.2 + P(1-P)[1 + E_R(p_0)] + (1-P)[0 + E_R(p_0)]$$

and hence,

$$E_R(p_0) = \frac{1+P}{P} \tag{3.7}$$

Since we assume that node 0 delivers p_1 in one attempt, the expected number of messages

received for the delivery of $ACK(p_1)$ is

$$E_{R_{ACK}}(p_1) = P.1 + (1-P)[0 + 1 + E_{R_{ACK}}(p_1)]$$

and hence,

$$E_{R_{ACK}}(p_1) = \frac{1}{P} \tag{3.8}$$

Therefore,

$$E_R(p_1) = \frac{1+P}{P} \tag{3.9}$$

Hence, the expected number of messages received by nodes 0 and 1 is

$$E_R(p_0) + E_R(p_1) = \frac{2(1+P)}{P} \tag{3.10}$$

Let ${E'}_R(p_0)$ be the expected number of messages received for delivery of p_0 in the $NACK$-based method. Then,

$${E'}_R(p_0) = P.1 + (1-P)[0 + 1 + {E'}_R(p_0)]$$

and hence,

$${E'}_R(p_0) = \frac{1}{P} \tag{3.11}$$

Since we assume node 0 delivers p_1 in one attempt, the expected number of messages received for the delivery of p_1 is

$${E'}_R(p_1) = 1 \tag{3.12}$$

Hence, the expected number of messages received by nodes 0 and 1 is

$${E'}_R(p_0) + {E'}_R(p_1) = \frac{1+P}{P} \tag{3.13}$$

Thus, the nodes in the ACK-based method receive on an average $\frac{1+P}{P}$ extra messages for delivering $n = 2$ packets.

Finally we analyze the **number of messages sent** in the $NACK$-based vs. ACK-

based methods for $n \geq \mathbf{2}$ packets. In the case of $n \geq 2$ packets, the probability that at least one packet has been successfully delivered in the first attempt is

$$Q_n = 1 - (1 - P)^n$$

and assuming that this is the case the probability that $k \geq 1$ packets are delivered in the first attempt is

$$p(k) = \frac{1}{Q_n} \binom{n}{k} P^k (1 - P)^{n-k} \tag{3.14}$$

The expected number of messages sent in the ACK-based method for the delivery of all n packets and their acknowledgments from node 0 to 1 is

$$\sum_{k \geq 1} p(k)[k.E(p_1) + (n - k)(1 + E(p_0))]$$

The expected number of messages sent in the $NACK$-based method for the delivery of all n packets is

$$\sum_{k \geq 1} p(k)[k + (n - k)(1 + \frac{1}{P} + E'(p_0))]$$

This shows that the nodes in the ACK-based method send on an average $\frac{n}{PQ_n}$ many extra messages, which can be very large when P is small; it is close to n when P is close to 1.

3.1.2 Hop-by-hop Detection and Recovery of Lost Packets

The protocol implements detection and recovery of lost packets at intermediate nodes between a source and destinations including the destinations themselves i.e. hop-by-hop detection and recovery of lost packets, instead of detecting and recovering lost packets at destination nodes i.e. end-to-end detection and recovery of lost packets. We describe both methods and then the reasons for preferring the former method.

In the **end-to-end detection and recovery** method, only the destination nodes detect and recover packets lost during transmissions provided they have received at least one packet. If the probability of successful transmission of a message between any two nodes seperated by 1 hop is P, then the probability of the message reaching a destination from the

source where the message travels along a path of length a hops is P^a which is less than P.

In the **hop-by-hop detection and recovery** method, each intermediate node x between source and destinations (including the destination nodes themselves) which have received at least one packet detect and recover packets missing at x. In this case, x must maintain a data cache to store the packets received so that it can detect and request missing packets.

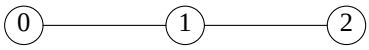
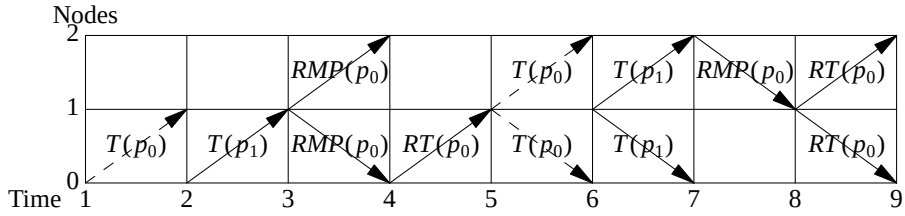


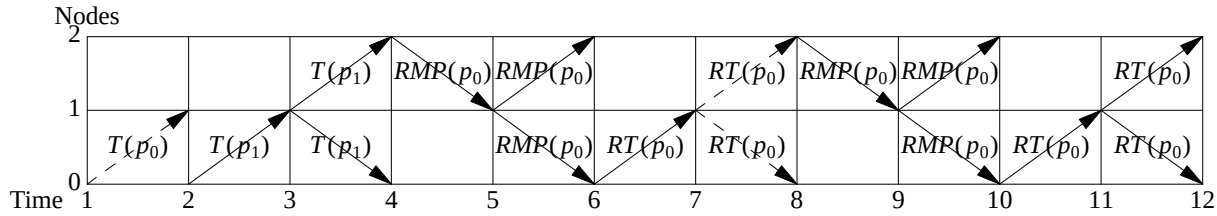
Figure 3.3: A 3-node linear network.

In the end-to-end method, detection and recovery of lost packets take longer time as detection and recovery take place only at destination nodes in comparison to hop-by-hop method where detection and recovery take place at all nodes between the source and destinations including the destinations themselves. Also the probability of a message reaching a destination from source is less in comparison to a message reaching a node from its 1-hop neighbor ($P^a < P$). This will lead to more messages in terms of requests and retransmissions and hence, longer delivery time for all n packets sent by the source in the end-to-end detection and recovery method. In hop-by-hop detection and recovery method, data caches have to be maintained at intermediate nodes which requires extra resources but intermediate nodes will be required to have data caches anyway since they can function as destination nodes a a later point in time for different data requirements.

The example in Figure 3.4 shows the time taken by both methods with exactly 3 lost transmissions of a packet p_0 occuring in the network in Figure 3.3. We assume that $T(p_1)$ is delivered in one attempt. The delivery time and number of messages required for detection and recovery of p_0 in hop-by-hop method is 9 time units and 12 messages while that in the end-to-end method is 12 time units and 16 messages. It is evident that here the hop-by-hop detection and recovery method performs better than the end-to-end detection and recovery method.



(a) Hop-by-hop recovery of lost packets.



(b) End-to-end recovery of lost packets.

Figure 3.4: Comparison of end-to-end vs. hop-by-hop detection and recovery of lost packets for delivery of 2 packets from nodes 0 to 2 via non-acknowledgement based method ($t_r = 0$) in the network in Figure 3.3.

3.1.3 Out-of-Sequence (OS) Forwarding of Packets at a Node

The alternative to out-of-sequence (OS) forwarding of packets at nodes is in-sequence (IS) forwarding of packets. We describe both methods and the reasons for choosing the former below.

In **IS-forwarding**, a node x can send a packet p_j only if it has previously sent each packet p_i, $i < j$, at least once. This tends to delay the delivery of p_j to a node and increase the total delivery time.

In **OS-forwarding**, a node x can send a packet p_j before sending one or more p_i, $i < j$. In particular, a node y can have the complete set of n packets although none of its neighbors have the complete set; this is not possible in IS-forwarding. This increases the possibility of y getting the packets quicker than in IS-forwarding.

3.1.4 A Priority Order for Sending Different Types of Messages at a Node

In OS, the preferred priority order for sending different types of messages at a node is $T > RT > RMP$. Since a node x can send a packet p_j without any constraints on j, it should send the packets as soon as possible so that a destination node y can get different p_j from different neighbors at the earliest. The preferred priority order in IS-forwarding [as in

PSFQ] is $RMP > RT > T$; here, it is better for a node x to request a missing packet p_i because it cannot transmit p_j, $j > i$, before transmitting p_i at least once. Unlike PSFQ, a node x does not delay the sending of Ts in OS-forwarding.

The following **example** illustrates the superiority of OS($T > RT > RMP$) over IS($RMP > RT > T$). Figure 3.6 shows the minimum time and number of messages required for the delivery of $n = 2$ packets $\{p_0, p_1\}$ from node 0 to node 3 in the network in Figure 3.5 using OS($T > RT > RMP$) and IS($RMP > RT > T$) respectively. We assume that in both cases the only messages lost are $T(p_0)$ from node 0 to node 2 at time 1 and $T(p_1)$ from node 0 to 1 at time 2, as indicated by the dashed lines. Here, we use $t_r = 0$ for simplicity. It shows how OS can achieve a better performance than IS. The sequence of events in Figure

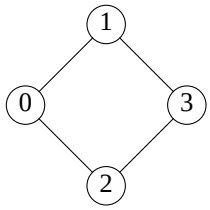
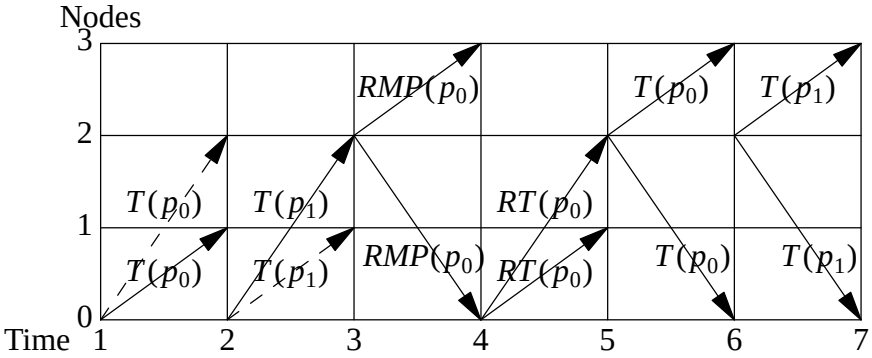


Figure 3.5: A 4-node non-linear network.

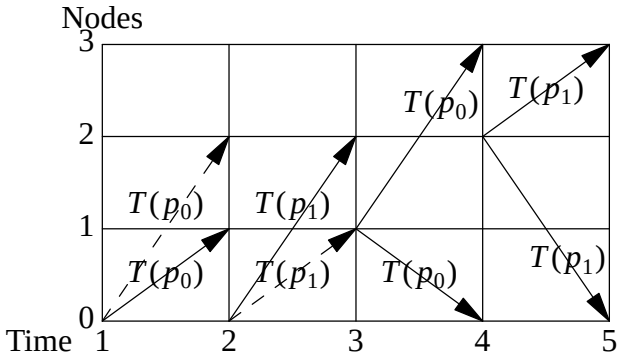
3.6 is repeated in Figure 3.7 in a different manner for better understanding of the OS vs IS forwarding.

However, the priority order for sending different types of messages present at different nodes may be vary. The following points illustrate this issue.

- It is desirable for destination nodes to receive the entire set of data packets as early as possible and say, for example, execute the instructions in the data set. Hence, if destination nodes have the priority order as $RMP > RT > T$, it may enable them to request and hence, receive the packets earlier than when their priority order is $T > RT > RMP$.
- The priority order at nodes may be also determined probabilistically. While it is better for nodes nearer to the source to send whatever packets they have as fast as possible so that nodes downstream can get different data packets from different neighbors at the earliest, it is possible that some missing packets at the nodes are not requested for a long time due to the less priority put on RMPs and nodes nearer to the destination



(a) IS($RMP > RT > T$).



(b) OS($T > RT > RMP$).

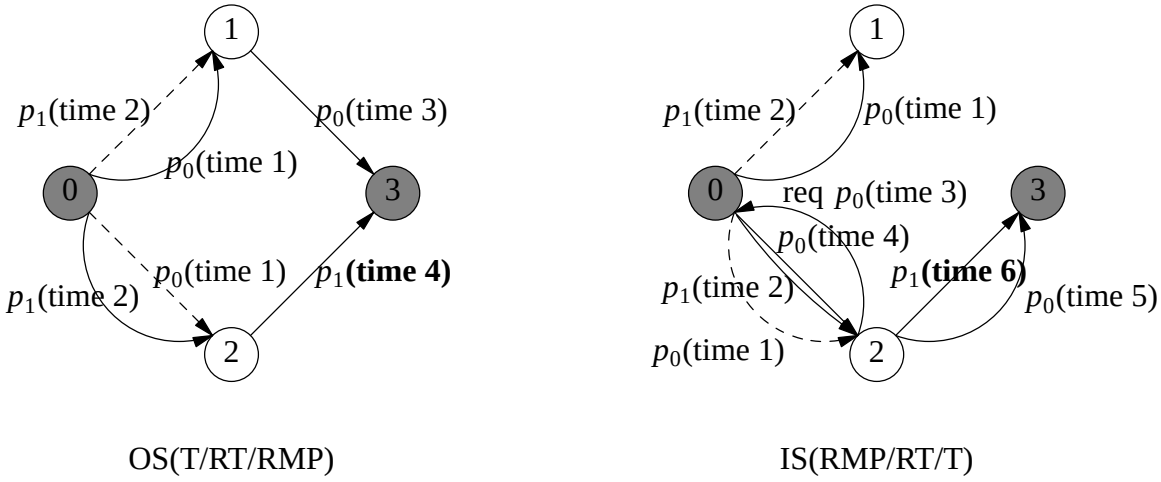
Figure 3.6: Comparison of OS($T > RT > RMP$) and IS($RMP > RT > T$) for the delivery of 2 packets from nodes 0 to 3 in the network in Figure 3.5 (timing diagram).

nodes including destination nodes do not receive these packets from any neighbors. In that case, it is better if nodes nearer to the destination nodes including them put priority on RMPs. This can probably alleviate the problem. Hence, the probability of a node sending RMPs is determined by its distance from the source which is *the current known distance of the node from the source* divided by *the shortest distance between the source and a destination*. It increases as distance from the source increases.

3.1.5 Delay in Requesting Packets Missing at a Node

A node x may have a missing packet p_i for either of the following two reasons:

- p_i has not been sent to x.
- Each transmission of p_i to x from its neighbors has failed to reach x.



OS(T/RT/RMP) IS(RMP/RT/T)

Figure 3.7: Comparison of OS($T > RT > RMP$) and IS($RMP > RT > T$) for the delivery of 2 packets from nodes 0 to 3 in the network in Figure 3.5.

We use a minimum delay of t_r time units from the time p_i is detected missing at x till x sends the first RMP for p_i to its neighbors; this allows enough time for p_i to reach x from one or more of its neighbors. The same delay t_r is used between two successive RMPs from x for a given p_i. This prevents too many unnecessary RMPs for p_i in case none of the neighbors of x currently has p_i.

Optimal estimation of the delay is an area of further research. We have experimented with different methods of calculating the delay and present them below.

- Delay = Degree of a node × RMPDelayFactor.
- Delay = Effective degree of a node × RMPDelayFactor.

Delay = Degree of a node × RMPDelayFactor. A node x should give a chance to all of its neighbors to send a packet p_i to it if they have p_i. At a time, only one neighbor can send p_i to x in order to prevent collisions. Hence, it is better for x to delay its $RMP(p_i)$ for a time equal to degree(x). This gives a chance to all of its neighbors to send p_i. However, it is possible that when x sends an $RMP(p_i)$, none of its neighbors have p_i. So x must wait a certain amount of time and then give a chance to all its neighbors to send p_i. Hence, we multiply the degree with a constant RMPDelayFactor. It is an integer constant which can be varied starting from 1. The effect of RMPDelayFactor will depend on the type of network and P.

Delay = Effective degree of a node × RMPDelayFactor. It is more likely that a node receives most packets from its upstream neighbors (neighbors nearer to the source). So, all neighbors are not equally responsible for sending packets. Hence, a node x will prefer not to delay sending RMPs by considering in its delay computation those nodes which do not contribute or contribute less to the sending of packets. So, instead of considering the degree(x) in the delay computation, we consider effective degree of x. Let number of packets sent from y (a neighbor of x) by time t be $n_{y \to x}$ (we consider only the packets which y has not sent before). Let total number of such packets from all neighbors of x by time t be $n_{ally \to x}$. Ideally, we expect each neighbor to equally contribute in sending the total n packets. Hence effective degree of x is computed as follows for each neighbor y of x at time t: if $\frac{n_{y \to x}}{n_{ally \to x}} \geq \frac{1}{degree(x)}$, effective degree($x$) = effective degree($x$) + 1.

3.2 Components of Each Message Type

Each message sent by a node x to its neighbors includes a data set id, nodeTTL(x) − 1, n, the message type (T, RT, or RMP), and one of the following:

- The packet sequence number i and the data for p_i, for T and RT messages.
- The sequence number i of a missing packet p_i, for RMP messages.

We refer to the component (nodeTTL(x) − 1) in a message as packetTTL(p_i). The data set id is the same for all n packets of data sent by the source and is the same for all RMPs and RTs for any of the n packets.

TimeToLive(TTL): nodeTTL and packetTTL. TimeToLive (TTL) or packetTTL is a counter associated with a packet p_i as it travels away from the source. When p_i is transmitted by the source, its TTL is equal to h where h is the minimum distance (in hops) between the source s and a destination d. As it travels one hop, its TTL decreases by one. So, when it reaches a destination node, its TTL equals zero. This helps a node identify itself as a destination node and know that it should not transmit p_i further. We incorporate some changes in the manner in which TTL is maintained in order to ensure that intermediate nodes do not identify themselves as destination nodes due to packets reaching

them via longer paths and thus prevent packets from reaching the actual destination nodes at a minimum distance h from s.

Each node x which has received at least one message maintains a nodeTTL, which equals the maximum packetTTL (packet Time To Live) associated with all messages received by x. For the source node s, nodeTTL(s) is initialized to h and it does not change with time. The packetTTL of any packet p_i sent by s is (h - 1). As a packet p_i travels away from the s along various paths, the packetTTL(p_i) associated with the $T(p_i)$ and $RT(p_i)$ messages for p_i typically goes down by one with each step; it can also occasionally go up when p_i reaches a node that has previously received other messages along shorter paths. A node x with nodeTTL(x) = 0 is considered a destination node, except that if at some later time nodeTTL(x) becomes greater than zero then from that point onwards x remains permanently labeled as an intermediate node. An RMP message from x to y has a packetTTL associated with it in order to inform the recipient node y of the current shortest path between s and the sending node x which may help y to update its path information.

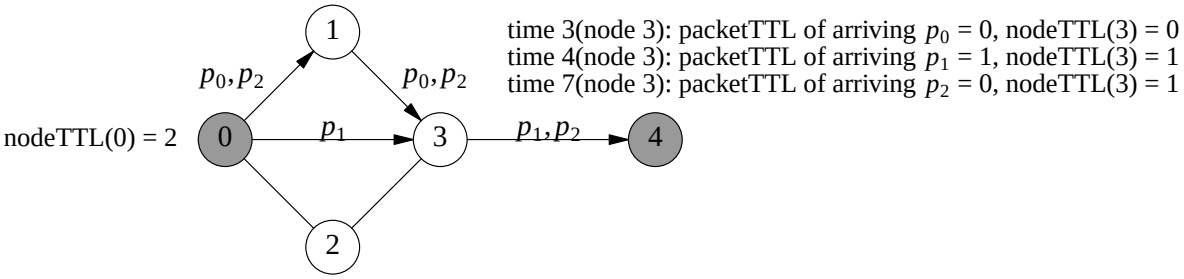


Figure 3.8: Update of nodeTTL of a node based on packetTTL.

The following **example** illustrates the necessity of maintaining the nodeTTL at the nodes. In Figure 3.8, node 0 is the source and it has to send 3 packets {p_0,p_1,p_2} to all nodes 2 hops away. We assume that only one node can transmit at each time unit. The following illustrates the sequence of events that occur in the network in Figure 3.8:

- **Time 1:** nodeTTL(0) = 2; packetTTL(p_0) = packetTTL(p_1) = packetTTL(p_2) = 2. Node 0 transmits p_0. Transmission of p_0 to node 1 is successful but the transmissions to nodes 2 and 3 are not.
- **Time 2:** packetTTL(p_0) received at node 1 = 1. nodeTTL(1) is initialized to 1. Node 1 transmits p_0. Only the transmission to node 3 is successful.

- **Time 3:** packetTTL(p_0) received at node 3 = 0. nodeTTL(3) is initialized to 0. Hence, node 3 assumes that it is a destination node and hence, does not transmit p_0. Node 0 transmits p_1. The transmission of p_1 to node 3 is successful but the transmissions to nodes 0 and 2 are not.
- **Time 4:** packetTTL(p_1) received at node 3 = 1. nodeTTL(3) is updated to 1. Node 3 realizes that it is not a destination node and transmits p_1. Only the transmission to node 4 is successful.
- **Time 5:** packetTTL(p_1) received at node 4 = 0. nodeTTL(4) is initialized to 0. Node 4 assumes that it is a destination node and hence, does not transmit p_1. Node 0 transmits p_2. The transmission of p_2 to node 1 is successful but the transmissions to nodes 2 and 3 are not.
- **Time 6:** packetTTL(p_2) received at node 1 = 1. nodeTTL(1) is not updated as nodeTTL(1) = packetTTL(p_2). Node 1 transmits p_2 and only the transmission to node 3 is successful.
- **Time 7:** packetTTL(p_2) received at node 3 = 0. nodeTTL(3) is not updated as nodeTTL(3) > packetTTL(p_2). Though packetTTL of p_2 is 0, the value of its nodeTTL helps node 3 realize that it is not a destination node and it transmits p_2. Only the transmission to node 4 is successful.

In the above example, if the nodeTTL(3) was not maintained, node 3 on receiving p_2 would have assumed itself to be a destination node and not transmitted p_2 which would result in node 4 sending a request for p_2, thus increasing the number of messages required for delivering the packets which is undesirable in energy-constrained wireless sensor networks.

3.3 Assumptions about the Local Data at a Node

Each node x with nodeTTL(x) ≥ 0 has a Data Cache DC(x) = $\{p_i$: p_i received by $x\}$, a Transmission Queue (TQ(x)), a ReTransmission Queue (RTQ(x)) and a Request-for-Missing-Packet Queue (RMPQ(x)). At each node x, the following properties hold:

- DC(x) and RMPQ(x) are disjoint.

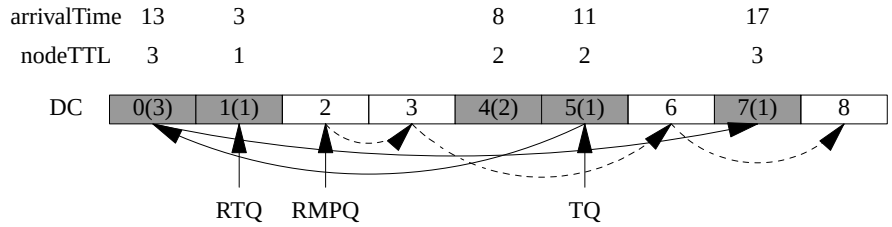


Figure 3.9: Illustration of DC, TQ, RTQ, RMPQ, and updating of nodeTTL at a node.

- $|\text{DC}(x) \cup \text{RMPQ}(x)| = n$.
- TQ(x) and RTQ(x) are disjoint subsets of DC(x).
- None of DC(x), TQ(x), RTQ(x) and RMPQ(x) contains any duplicate item.
- The packets p_i in TQ(x) are ordered in the order of their arrival via Ts or RTs, and those in RTQ(x) are ordered in the order of their arrival via RMPs. RMPQ(x) is maintained as a circular list, ordered by the packet sequence numbers.

The following **example** illustrates the different queues of a node, their contents and update of nodeTTL of the node. In Figure 3.9, the DC(x) of a node x for $n = 9$ and the *arrivalTime* of each p_i in DC are shown; in particular, the packets that arrived in chronological order are p_1, p_4, p_5, p_0, and p_7. The packetTTL(p_i) is shown in parentheses next to each p_i in DC. Each of TQ, RTQ and RMPQ points to the first packet in the corresponding queue and the arrows from one packet to another show the sequence of packets in the queue. We also show the updates to nodeTTL(x) as each $p_i \in$ DC is received.

3.4 Description of the New Protocol (OSDRMP)

3.4.1 Message Processing at a Node

In one time unit, a node x can do one or more of the following:

- Receive a message from one of its neighbors and process it.
- Send a message to each of its neighbors.
- Remain idle.

3.4.2 Nodes Selected for Message Transmission

The set of candidate nodes that are eligible to send messages at time t is given by

$$S'(t) = \{x : TQ(x) \cup RTQ(x) \neq \emptyset \text{ or } x \text{ can send } RMP(p_i) \text{ at time } t \text{ for } p_i \in RMPQ(x)\} \tag{3.15}$$

$S(t)$ is the set of nodes selected from $S'(t)$ for sending messages at time t. Any protocol that prevents message collision may be used to decide the group of nodes $S(t) \subseteq S'(t)$ that send messages at time t. At time $t = 1$, $S'(1) = S(1) = s$.

We can exclude a node x with nodeTTL(x) = 0 from sending T and RT messages; this tends to reduce the number of messages slightly without significantly affecting the total delivery time in spite of the fact two destination nodes may be adjacent to each other and that a nodeTTL(x) may become positive at a future time. The advantage of having destination nodes sending T and RT messages is that a destination node x may be successful in sending packets to another destination node y which is yet to receive packets from any other node. This helps in ensuring that all possible destination nodes receive the data at the earliest.

A node in $S(t)$ transmits a message from one of its TQ, RTQ or RMPQ in that priority order. A node x sends $RMP(p_i)$ for the first $p_i \in$ RMPQ(x) which satisfies t_r delay requirement. For each p_i, x sends $T(p_i)$ only once but it sends $RMP(p_i)$ and $RT(p_i)$ multiple times.

Two different methods of requesting RMPs can be implemented in our protocol. A node x can have a missing packet p_i because none of its neighbors have p_i or because it has been lost in transmission to x. Since the sequence of forwarding packets is OS which means there are no restrictions on the order in which x transmits packets, either one of the above reasons can be responsible for any missing p_i at x. It is not clear whether requesting all missing packets including those with sequence numbers higher than m_x will be beneficial or requesting only those with sequence numbers lower than m_x will be beneficial where m_x is the highest packet sequence number in DC(x). Hence, either way of requesting RMPs can be implemented depending on network type and P.

In **restrictedRMP(rRMP)**, x sends an $RMP(p_j)$ at time t only if $j < m_x$; the

other restrictions that $p_j \in \text{RMPQ}(x)$ and x satisfies the delay requirement t_r still apply. If there is no $p_j \in \text{RMPQ}(x)$ with $j < m_x$ and $\text{RMPQ}(x) \neq \emptyset$(i.e., $m_x < n-1$), then x sends $RMP(p_j)$ for $j = m_x + 1$ repeatedly till it receives $p_i \geq p_j$.

In **unrestrictedRMP(urRMP)**, a node x sends RMPs from the RMPQ in a circular fashion while satisfying the the delay requirement t_r.

3.4.3 Processing of Input

In addition to the update of nodeTTL(x), the DC and various queues at a node x are updated as follows when it receives an input. It processes an input message if and only if it is the first message to x with packetTTL ≥ 0 or nodeTTL(x) ≥ 0.

- If the first message with packetTTL ≥ 0 is:
  - a $T(p_i)$ or an $RT(p_i)$, then x initializes $\text{DC}(x) = \text{TQ}(x) = \{p_i\}$, $\text{RMPQ}(x) = \{p_j$: $j \neq i$ and $j < n\}$, and $\text{RTQ}(x) = \emptyset$.
  - an $\text{RMP}(p_i)$, then x initializes $\text{RMPQ}(x) = \{p_0\}$ and $\text{DC}(x) = \text{TQ}(x) = \text{RTQ}(x) = \emptyset$.
- When nodeTTL(x) ≥ 0, if the message is:
  - a $\text{T}(p_i)$ or an $\text{RT}(p_i)$ and $p_i \notin \text{DC}(x)$, then p_i is added to both $\text{DC}(x)$ and $\text{TQ}(x)$ and is removed from $\text{RMPQ}(x)$, if necessary; $\text{RMPQ}(x)$ is updated with all p_j where $p_j \notin \text{DC}(x) \cup \text{RMPQ}(x)$. If $p_i \in \text{DC}(x)$ then no updates take place.
  - an $\text{RMP}(p_i)$, then p_i is added to $\text{RTQ}(x)$ if $p_i \in \text{DC}(x)$ and $p_i \notin \text{TQ}(x) \cup \text{RTQ}(x)$. If $p_i \notin \text{DC}(x)$ or if $p_i \in \text{DC}(x)$ and $p_i \in \text{TQ}(x) \cup \text{RTQ}(x)$, then no updates take place.

3.5 A Method for Ensuring At Least One Packet Delivery to a Node

In non-acknowledgement based methods, at least one packet needs to reach a node in order for the node to request missing packets. Though we assume in our protocol that a message will have enough packets for at least one packet to reach a node, we present a method that

ensures at least one packet delivery to a node at the cost of some extra energy in terms of messages.

In this method, when a node x puts a packet in its TQ for the first time, it creates an array ACK_ARRAY whose length is equal to the degree of x. For each neighbor y of x, there is an entry in the array which consists of the neighbor y's id and a bit which is initially set to zero. The value of this bit is set to 1 when x knows that y has received at least one packet. Whenever x receives a T or an RT message or an acknowledgement (ACK) message from y for the first time, the corresponding bit in ACK_ARRAY is set to 1. We assume that each message will have it's sending node's id.

When x sends its first T which may be for any packet p_i, it sets the $ACKREQ$ bit in the packet to 1 and destination in the same to all y which have not yet sent a T or an RT or an ACK message. It expects an ACK from each neighbor y of x within a certain time interval t_a where the corresponding bit for y in ACK_ARRAY is not set to 1. If it does not receive a T or an RT or an ACK message within the t_a time interval from a neighbor y, it sends a packet from its TQ or RTQ and if they are empty, the packet p_l having the lowest sequence number in its data cache with the $ACKREQ$ bit in the packet set to 1 and destination in the same set to all y which have not sent a T or an RT or an ACK message. This retransmission is called $RT_{ACK}(p_l)$ and will have priority over all other message types except ACK messages for transmission purposes. It sends $RT_{ACK}(p_l)$ after every t_a time intervals as long as it does not receive a T or an RT or an ACK message from y. If x receives a T or an RT or an ACK message from y when the corresponding bit in ACK_ARRAY is zero, then its sets the bit to 1. This indicates that y has received at least one packet. The node x expects only one ACK from each neighbor y. To reduce the number of messages, node ids for all nodes which have not sent a T or an RT or an ACK message are placed together in the destination section of a packet p_i whose $ACKREQ$ bit is set to 1. If x is a destination node, it will not wait for ACK messages.

When a node y receives a packet which may be any packet p_i with the $ACKREQ$ bit set to 1 and it's id present in the destination section of p_i, it sends an ACK. However, packetTTL(p_i) or nodeTTL(y) must be greater than zero.

The ACK message of a node y consists of the data set id, the node id of y, nodeTTL(y)

-1 and the message type which is ACK here. The sending of ACK message type will have priority over other message types at the nodes. The ACK is broadcast in order to ensure all neighbors of x know that it received at least one packet and hence, they will not wait for an ACK message from it when they send a packet which may reduce the number of messages.

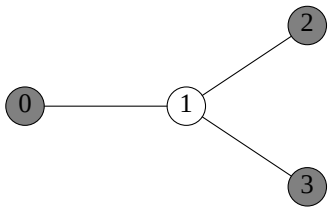


Figure 3.10: A 4-node non-linear network.

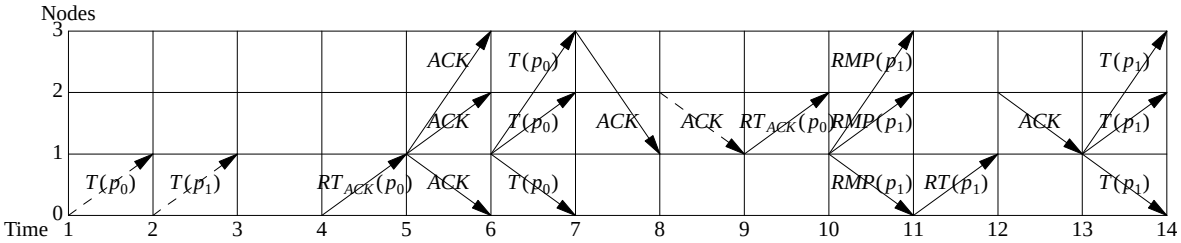


Figure 3.11: Ensuring at least one packet delivery in a non-acknowledgement based method for delivery of 2 packets from node 0 to nodes 2,3 in the network in Figure 3.10.

In Figure 3.11, $n = 2$ packets $\{p_0, p_1\}$ are delivered from node 0 to nodes 2 and 3 in the network in Figure 3.10. Dashed lines show lost messages. The time interval t_a is set to 1 for simplicity. The delay in sending RMPs at each node x is equal to the product of degree(x) and RMPDelayFactor which is set to 1. At each time unit, only one node can send a message. We observe that at node 1, sending $RT_{ACK}(p_0)$ and ACK has priority over other message types.

The messages $T(p_0)$ and $T(p_1)$ from node 0 to node 1 are lost. Node 0 waits a time interval t_a before sending $RT_{ACK}(p_0)$ to node 1. Node 1 now sends an ACK which reaches node 0. Then, node 1 sends $T(p_0)$. All transmissions are successful. In response, both neighbors of node 1 i.e. nodes 2 and 3, send ACKs. Note that node 1 does not require an ACK from node 0 as it already knows that node 0 has at least one packet. But the ACK from node 2 is lost. Here, at both nodes 2 and 3, sending an ACK has priority over sending $T(p_0)$. Node 1 sends a $RT_{ACK}(p_0)$ to node 2 and then, sends $RMP(p_1)$. Note that sending $RT_{ACK}(p_0)$ has priority over sending $RMP(p_1)$ here. Also, node 2 would have sent an ACK

but did not have access to the medium here. In reponse to node 1's $RMP(p_1)$, node 0 sends $RT(p_1)$. Then, node 2 gets chance to send a message and transmits an ACK. Finally node 1 sends $T(p_1)$ which reaches all nodes.

3.6 A Reporting Method Indicating Delivery of All Packets to At Least One Destination Node

Sometimes, the base station may need to know if the entire data has been delivered to at least one destination node. We do not say all destination nodes here since a base station cannot estimate the number of destination nodes. So, it will not know how many reports to wait for if it wants reports from all destination nodes. If it receives a report from one destination node, then it knows that all n packets have successfully traveled the specified distance and are likely to have reached other potential destination nodes. We present a new reporting method for delivering a report back to the base station that at least one destination node has received the entire data.

The base station station indicates that it requires a report by setting a *REPREQ* bit to 1 in all packets that it sends out. All nodes with nodeTTL ≥ 0 receiving messages with the *REPREQ* bit set to 1 sets the *REPREQ* bit to 1 in all T, RT and RMP messages that they send to their neighbors. Once a destination node gets the entire data and a message with the *REPREQ* bit set to 1 , it activates a *REPORT* bit by setting it to 1 in all T and RT messages that it sends to its neighbors. If it has no message to send, its sends a report (REP) message to its neighbors. The REP message has the data set id and the type of message which is REP in this case. Both the *REPORT* bit and REP message indicate that all n data packets have been received.

Each node which gets a message with the *REPORT* bit set to 1 in a T, an RT, an RMP or a REP message activates the *REPORT* bit in any T, RT or RMP message it sends to its neighbors. If it does not have a message to send, it sends a REP message to its neighbors. A node will send a message with the $REPREQ$ bit or the report bit set to 1 or a REP message only for a pre-determined maximum number of times. This can be equal to the degree of the node. This is done to reduce too many request and report messages since

only one message needs to reach the base station. At each node, sending of T, RT and RMP messages will have priority over sending of REP messages. Only nodes with nodeTTL $\geq$ 0 will send REP messages or messages with $REPORT$ bit set to 1. When the base station receives a message with the $REPORT$ bit set to 1 or a REP message it knows that at least once destination node has received the entire data.

Chapter 4

Simulation Environment, Software and Results

We simulate the OSDRMP(rRMP and urRMP) and PSFQ-based protocols in the network in Figure 4.1 for the following source(s)-destination(d) pairs at 3 different probabilities of successful transmission (P = 0.3, 0.6 and 0.9) where the source is the base station and the destinations are the sensor nodes.

- $s = 40$, $h = 5$ i.e. all d which are at a distance of 5 hops from s.
- $s = 0$, $h = 14$ i.e. all d which are at a distance of 14 hops from s.

In one set of simulations, we vary the RMPdelayFactor keeping the number of packets constant at $n = 10$ and in the other set of simulations we vary the number of packets (n) keeping the RMPdelayFactor constant at RMPdelayFactor = 3. When n is constant, we choose the value of the constant n as 10 in order to better analyze the results. A higher number would make it more difficult to analyze the results. When the RMPdelayFactor is constant, we choose the value of the constant RMPdelayFactor as 3 because the simulation results in the previous case show that both protocols perform well at approximately this value of RMPdelayFactor for all s-d pairs and all probabilities.

We compare the performances of the protocols based on the total delivery time (delivTime) and the total number of messages sent and received (numMess) to deliver all the n packets from source to destinations. We analyze the time taken by different nodes in the network in Figure 4.1 to fill their DC when OSDRMP(rRMP) protocol is executed under the conditions mentioned before and the reasons behind it. We also modify the OSDRMP(rRMP) protocol by varying the method by which the delay in sending RMPs is calculated, the priority order of sending messages of different types at different nodes and the way in which the nodes respond to RMPs, then simulate the protocol with modifications in the network in Figure 4.1 under similar conditions mentioned before and compare the performances of the modified versions with the unaltered OSDRMP(rRMP) protocol. The reason for choosing OSDRMP(rRMP) protocol is that though both versions of the OSDRMP protocol (rRMP and urRMP) perform better than the PSFQ-based protocol, the

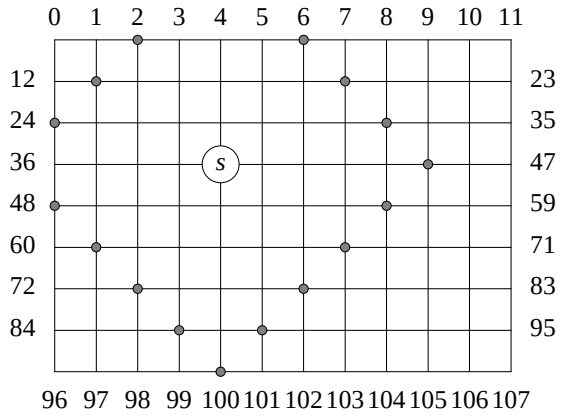


Figure 4.1: A 12x9 grid network; the intended destination nodes are marked for s = 40 and $h = 5$.

OSDRMP(urRMP) protocol use more messages in comparison to the the PSFQ-based protocol at low probabilities. Hence we choose the OSDRMP(rRMP) protocol for our purposes.

4.1 Simulation Environment and Software

We have developed a network simulator in C in the Unix environment. The flowchart of the simulation program for the OSDRMP protocol is given in Figure 4.2 followed by more details.

The following are provided to the simulator via an input file:

- The network in the form of a graph adjacency list.
- s.
- h
- P.
- RMPDelayFactor.
- n

All of s, d, h and n have been defined in Section 1.4 and in the beginning of Chapter 3. The simulator has a function newProtocol() which executes the protocol for the input network. The function new Protocol executes till the terminating condition is met or none of the nodes in the network have any messages to send. If the terminating condition is met, the run is successful, otherwise it is an unsuccessful run.

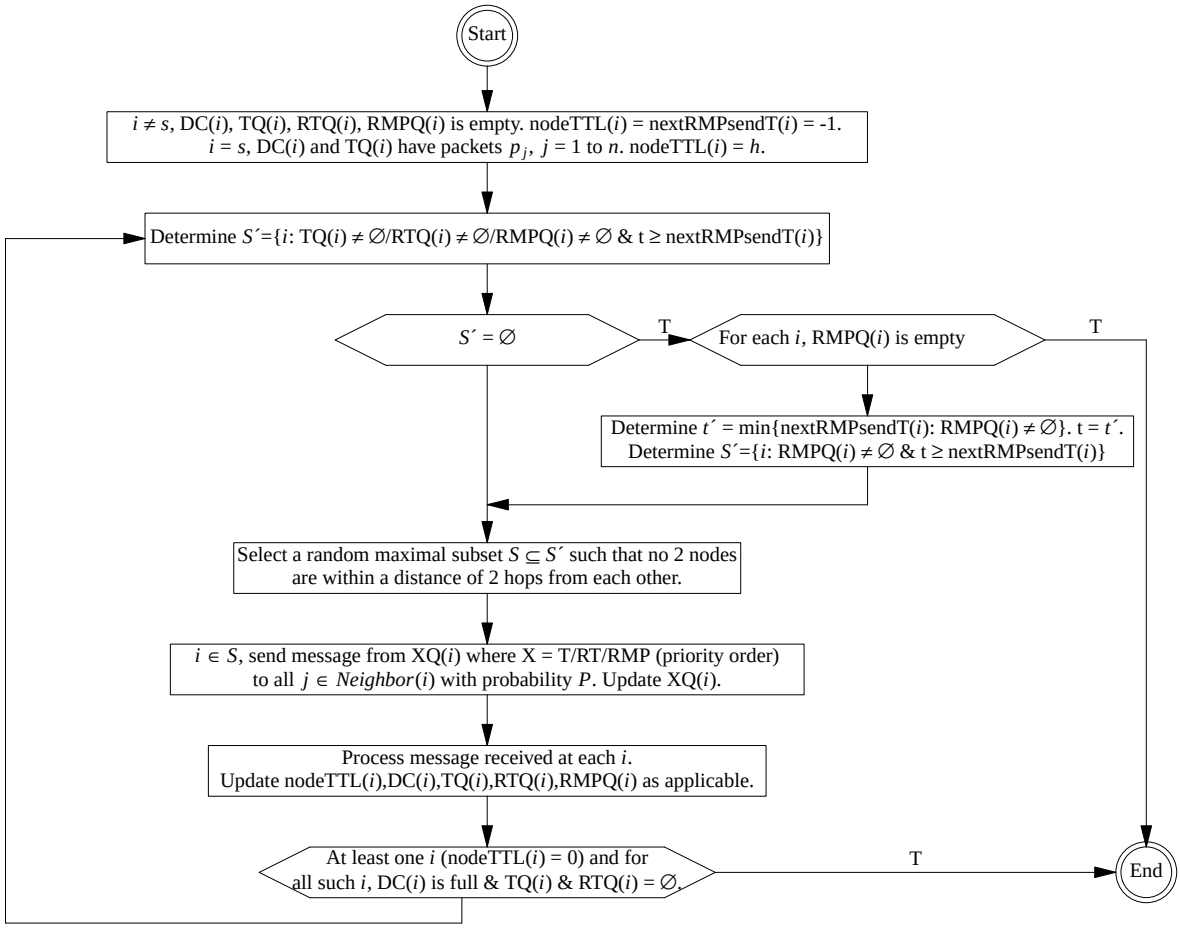


Figure 4.2: Flowchart for the network simulator for OSDRMP protocol.

The terminating condition is that at least one destination node x (i.e., nodeTTL(x) = 0) is found and each node x with nodeTTL(x) = 0 has | DC(x) | = n and TQ(x) = RTQ(x) = $\emptyset$; the condition TQ(x) = RTQ(x) = $\emptyset$ reduces the probability that there are other potential destinations nodes that have not received any p_i yet.

The function newProtocol() is given below.

1. A function determinesTransmittingNodes() is executed to determine the nodes which will send messages during a time unit. If the output of determinesTransmittingNodes() is zero, all nodes are checked to see if there are nodes which have RMPs but are unable to send any because it is not yet time for any of them to send an RMP. In that case, the current time is advanced by a minimum time t' so that at least one node with a non-empty RMPQ meets the delay requirement and becomes ready to send an RMP at $t + t'$. The period $t' - t$ is idleTime which is a time unit when no node in the network has any T or RT to send and it is not time for them to send RMPs due to

delay requirements. The function determinesTransmittingNodes() is executed again. If the output is still zero, the run terminates and is considered to be an unsuccessful run.

2. If the output of determinesTransmittingNodes() is greater than zero, the nodes selected for transmission send their messages in the priority order $T > RT > RMP$. For each such node x, the probability P determines which neighbors of x receive a message.
3. For each node with an input, the input message is processed according to the algorithm in Section 3.4.3.
4. The terminating condition is checked to see whether it is true or not. If it is not true, then steps 1 to 4 are executed again else the function terminates and the run is considered to be a successful run.

The reason for executing two seperate loops for processing the inputs of nodes and sending messages from nodes respectively is that this is a serial execution environment and if both occured in the same loop, x may end up sending a message to another node y at time t which we may encounter later during execution of the loop and process it, also at time t which is wrong because the output message is supposed to reach y at time $t + 1$.

The function determinesTransmittingNodes() is given below.

1. For each node x, all nodes at a distance ≤ 2 from x are calculated and stored in $adjDist2Nodes[x]$. This is done only in the first call of determinesTransmittingNodes().
2. All nodes in $S'(t)$ in Eqn. 3.15 in Section 3.4.2 are stored in the array *candidateTransmittingNodes*.
3. A node y is randomly selected and added to *nodesSelectForTrans* which represents the set $S(t)$ in Section 3.4.2.
4. The node y and all nodes in $adjDist2Nodes[y]$ are removed from *candidateTransmittingNodes*.
5. Steps 3 to 4 are repeated till *candidateTransmittingNodes* is empty.

The following example shows the execution of determinesTransmittingNodes() for the 10-node linear network in Figure 4.3. The input to the function determinesTransmittingN-

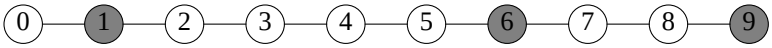


Figure 4.3: A 10-node linear network.

odes() is the network shown below.

```
numNodes = 10
nodeNum (degree) adjNodes:
0 (1)  1
1 (2)  0 2
2 (2)  1 3
3 (2)  2 4
4 (2)  3 5
5 (2)  4 6
6 (2)  5 7
7 (2)  6 8
8 (2)  7 9
9 (1)  8
```

Step 1. For each node i, we compute the nodes which are at a distance ≤ 2 from i. The output below shows the *adjDist2Nodes* array.

```
adjDist2Nodes[0]:  1 2
adjDist2Nodes[1]:  0 2 3
adjDist2Nodes[2]:  0 1 3 4
adjDist2Nodes[3]:  1 2 4 5
adjDist2Nodes[4]:  2 3 5 6
adjDist2Nodes[5]:  3 4 6 7
adjDist2Nodes[6]:  4 5 7 8
adjDist2Nodes[7]:  5 6 8 9
adjDist2Nodes[8]:  6 7 9
adjDist2Nodes[9]:  7 8
```

Step 2. In this example, we randomly choose the nodes belonging to $S'(t)$ i.e. *candidateTransmittingNodes* in the network above. The array *candidateTransmittingNodes* is initialized.

```
candidateTransmittingNodes:  0 1 2 5 6 7 8 9
```

Iteration 1. Step 3. Node selected is 9.

```
nodesSelectForTrans:  9
```

Iteration 1. Step 4. Node 9 and all nodes in *adjDist2Nodes*[9] (7,8) are removed from *candidateTransmittingNodes*.

```
candidateTransmittingNodes:  0 1 2 5 6
```

Iteration 1. Step 5. The array *candidateTransmittingNodes* is non-empty.

Iteration 2. Step 3. Node selected is 1.

```
nodesSelectForTrans:  9 1
```

Iteration 2. Step 4. Node 1 and all nodes in *adjDist2Nodes*[1] (0,2) are removed from *candidateTransmittingNodes*.

```
candidateTransmittingNodes:  5 6
```

Iteration 2. Step 5. The array *candidateTransmittingNodes* is non-empty.

Iteration 3. Step 3. Node selected is 6.

```
nodesSelectForTrans:  9 1 6
```

Iteration 3. Step 4. Node 6 and all nodes in *adjDist2Nodes*[6] (5) are removed from *candidateTransmittingNodes*.

```
candidateTransmittingNodes:
```

Iteration 3. Step 5. The array *candidateTransmittingNodes* is empty. Hence the iterations stop.

The simulator runs for R number of runs and averages the results over successful runs. The simulator requests R from the user during runtime. The following are the outputs from the simulator:

- delivTime averaged over all successful runs.
- numMess averaged over all successful runs. When a node x sends a message (T, RT, or RMP) to its neighbors, we count this as one message in numMess irrespective of the number of neighbors of x. The reason is that the energy spent by a node in sending a message is the same irrespective of the number of neighbors. Each message received by x is also counted as one message in numMess because energy is also spent by a node in receiving and processing a message.
- idleTime averaged over all successful runs.
- The number of successful and unsuccessful Ts, RTs and RMPs also averaged over all successful runs.
- A file containing details of each node which has atleast one packet in its queues or DC at each time unit during each run. The file also contains the delivTime, numMess and

idleTime for each run as well as a table giving the details of the number of messages of each type (T, RT, RMP) sent by each node to each of its neighbors during each run.

In Figures 4.4 and 4.5, we show outputs from the simulator after execution of OSDRMP(rRMP) protocol in the network in Figure 4.1.

In the example in Figure 4.4, at time 17 node 29 sends RMPs to nodes 17, 28, 30 and 41 for retransmission of packet 2. All RMPs are successfully sent. Nodes 17 and 30 do not have packet 2 in their DC. Node 28 has packet 2 in its DC but not in its TQ or RTQ and hence puts packet 2 in its RTQ. Node 41 has packet 2 in its DC and TQ and hence takes no action. At the same time, node 66 sends Ts of packet 0 to nodes 54, 65, 67 and 78. Only the T to node 65 is successful. Node 65 does not have packet 0 in its DC. Hence, it adds packet 0 to its DC and TQ and updates its RMPQ to request packet 2 instead of packet 0. At time 17, node 26 also sends Ts of packet 2 to nodes 14, 25, 27 and 38. All Ts are successful. Node 27 already has packet 2 in its DC and hence takes no action. Nodes 14, 25 and 38 do not have packet 2 in their DC and hence add packet 2 to their DC and TQ. The updated contents of all nodes are shown at time 18.

In the example in Figure 4.5, at time 123, no node has any Ts and RTs to send and it is not time for any node to send an RMP. So the software updates the time by the minimum time after which atleast one node can send an RMP which is time 129. So the time from 123 to 128 is called *idleTime*. At time 129, node 89 sends RMPs for packet 7 to nodes 77, 88, 90 and 101. Only the transmission to node 90 is unsuccessful.

We simulate PSFQ protocol by creating a protocol with the key characteristics of PSFQ i.e. IS-forwarding of packets and the priority order for sending different types of messages at nodes as $RMP > RT > T$. The methods by which this PSFQ-based protocol simulation processes its input, chooses the set of nodes for transmission and sends messages at each time unit are the same as in the OSDRMP protocol simulation.

4.2 Simulation Results

We implement both methods of requesting RMPs (rRMP and urRMP) described in Section 3.4.2 in our simulation of the OSDRMP protocol. The PSFQ-based protocol has only

```
Time Node    TQ                  RTQ                 pacForRRT           DC
 ================================================================================================

After processing nodes with non-empty IQs:
 17    2[D] (1)                                      (0)                 (1)
 17    3    (1)                                      (0)                 (1)
 17    4                                             (0)                 (1)
 17    5                                             (0)                 (1)
 17    6[D] (0)                                      (2)                 (0,1)
 17   14                                             (0)                 (1)
 17   15    (2)                                      (0)                 (1,2)
 17   16                                             (0)                 (1)
 17   17                                             (2)                 (0,1)
 17   18    (1)                                      (2)                 (0,1)
 17   19[D]                                          (2)                 (0,1)
 17   24[D]                                          (0)                 (1)
 17   25                                             (0)                 (1)
 17   26    (2)                                      (0)                 (1,2)
 17   27                                             (0)                 (1,2)
 17   28    (3,5)                                    (4)                 (0,1,2,3,5)
 17   29                                             (2)                 (0,1)
 17   30                                             (2)                 (0,1)
 17   31                                             (2)                 (0,1)
 17   32[D] (0)                                      (2)                 (0,1)
 17   37[D]                                          (0)                 (1)
 17   38                                             (0)                 (1)
 17   39    (1,4,2)                                  (0)                 (1,2,4)
 17   40[S] (6,7,8,9)                                                    (0,1,2,3,4,5,6,7,8,9)
 17   41    (2,0,3,4)                                (5)                 (0,1,2,3,4)
 17   42                                             (2)                 (0,1)
 17   43                                             (2)                 (0,1)
 17   48[D]                                          (0)                 (1)
 17   49                                             (0)                 (1)
 17   50                                             (0)                 (1)
 17   51    (4)                                      (0)                 (1,4)
 17   52                                             (0)                 (1,2,4)
 17   53    (2,4)                                    (0)                 (2,4)
 17   54                                             (2)                 (0,1)
 17   55    (1)                                      (2)                 (0,1)
 17   56[D]                                          (1)                 (0)
 17   61[D]                                          (0)                 (1)
 17   62                                             (0)                 (1)
 17   63                                             (0)                 (1)
 17   64    (2,4)                                    (0)                 (1,2,4)
 17   65                                             (0)                 (1)
 17   66    (0)                                      (2)                 (0,1)
 17   67[D]                                          (1)                 (0)
 17   75                                             (0)                 (1)
 17   76[D]                                          (0)                 (1)
 17   77                                             (0)                 (1)
 17   89[D]                                          (0)                 (1)

     Sending Receiving              Packet
Time  Node#    Node#s               & TTL  T_RT_RMP
*********************************************************
 17     29     +17,+28,+30,+41      2(3)   RMP
 17     52     -40,+51,+53,+64      0(4)   RMP
 17     32     -20,-31,+33,+44      0(0)   T
 17      3     +2,+4,+15            1(1)   T
 17      6     +5,+7,-18            0(0)   T
 17     66     -54,+65,-67,-78      0(1)   T
 17     26     +14,+25,+27,+38      2(2)   T

Time Node    TQ                  RTQ                 pacForRMP           DC
 ================================================================================================

After processing nodes with non-empty IQs:
 18    2[D] (1)                                      (0)                 (1)
 18    3                                             (0)                 (1)
 18    4                                             (0)                 (1)
 18    5    (0)                                      (2)                 (0,1)
 18    6[D]                                          (2)                 (0,1)
 18   14    (2)                                      (0)                 (1,2)
 18   15    (2)                                      (0)                 (1,2)
 18   16                                             (0)                 (1)
 18   17                                             (2)                 (0,1)
 18   18    (1)                                      (2)                 (0,1)
 18   19[D]                                          (2)                 (0,1)
 18   24[D]                                          (0)                 (1)
 18   25    (2)                                      (0)                 (1,2)
 18   26                                             (0)                 (1,2)
 18   27                                             (0)                 (1,2)
 18   28    (3,5)              (2)                   (4)                 (0,1,2,3,5)
 18   29                                             (2)                 (0,1)
 18   30                                             (2)                 (0,1)
 18   31                                             (2)                 (0,1)
 18   32[D]                                          (2)                 (0,1)
 18   37[D]                                          (0)                 (1)
 18   38    (2)                                      (0)                 (1,2)
 18   39    (1,4,2)                                  (0)                 (1,2,4)
 18   40[S] (6,7,8,9)                                                    (0,1,2,3,4,5,6,7,8,9)
 18   41    (2,0,3,4)                                (5)                 (0,1,2,3,4)
 18   42                                             (2)                 (0,1)
 18   43                                             (2)                 (0,1)
 18   48[D]                                          (0)                 (1)
 18   49                                             (0)                 (1)
 18   50                                             (0)                 (1)
 18   51    (4)                                      (0)                 (1,4)
 18   52                                             (3)                 (1,2,4)
 18   53    (2,4)                                    (0)                 (2,4)
 18   54                                             (2)                 (0,1)
 18   55    (1)                                      (2)                 (0,1)
 18   56[D]                                          (1)                 (0)
 18   61[D]                                          (0)                 (1)
 18   62                                             (0)                 (1)
 18   63                                             (0)                 (1)
 18   64    (2,4)                                    (0)                 (1,2,4)
 18   65    (0)                                      (2)                 (0,1)
 18   66                                             (2)                 (0,1)
 18   67[D]                                          (1)                 (0)
 18   75                                             (0)                 (1)
 18   76[D]                                          (0)                 (1)
 18   77                                             (0)                 (1)
 18   89[D]                                          (0)                 (1)
```

Figure 4.4: Simulation output from time 17 to time 18 in the network in Figure 4.1.

```
Time Node    TQ                  RTQ                 pacForRMP           DC
 ============================================================================================

After processing nodes with non-empty IQs:
123    2[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123    3                                                                 (0,1,2,3,4,5,6,7,8,9)
123    4                                                                 (0,1,2,3,4,5,6,7,8,9)
123    5                                                                 (0,1,2,3,4,5,6,7,8,9)
123    6[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   13[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   14                                                                 (0,1,2,3,4,5,6,7,8,9)
123   15                                                                 (0,1,2,3,4,5,6,7,8,9)
123   16                                                                 (0,1,2,3,4,5,6,7,8,9)
123   17                                                                 (0,1,2,3,4,5,6,7,8,9)
123   18                                                                 (0,1,2,3,4,5,6,7,8,9)
123   19[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   24[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   25                                                                 (0,1,2,3,4,5,6,7,8,9)
123   26                                                                 (0,1,2,3,4,5,6,7,8,9)
123   27                                                                 (0,1,2,3,4,5,6,7,8,9)
123   28                                                                 (0,1,2,3,4,5,6,7,8,9)
123   29                                                                 (0,1,2,3,4,5,6,7,8,9)
123   30                                                                 (0,1,2,3,4,5,6,7,8,9)
123   31                                                                 (0,1,2,3,4,5,6,7,8,9)
123   32[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   36                                                                 (0,1,2,3,4,5,6,7,8,9)
123   37                                                                 (0,1,2,3,4,5,6,7,8,9)
123   38                                                                 (0,1,2,3,4,5,6,7,8,9)
123   39                                                                 (0,1,2,3,4,5,6,7,8,9)
123   40[S]                                                              (0,1,2,3,4,5,6,7,8,9)
123   41                                                                 (0,1,2,3,4,5,6,7,8,9)
123   42                                                                 (0,1,2,3,4,5,6,7,8,9)
123   43                                                                 (0,1,2,3,4,5,6,7,8,9)
123   44                                                                 (0,1,2,3,4,5,6,7,8,9)
123   45[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   48[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   49                                                                 (0,1,2,3,4,5,6,7,8,9)
123   50                                                                 (0,1,2,3,4,5,6,7,8,9)
123   51                                                                 (0,1,2,3,4,5,6,7,8,9)
123   52                                                                 (0,1,2,3,4,5,6,7,8,9)
123   53                                                                 (0,1,2,3,4,5,6,7,8,9)
123   54                                                                 (0,1,2,3,4,5,6,7,8,9)
123   55                                                                 (0,1,2,3,4,5,6,7,8,9)
123   56[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   61[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   62                                                                 (0,1,2,3,4,5,6,7,8,9)
123   63                                                                 (0,1,2,3,4,5,6,7,8,9)
123   64                                                                 (0,1,2,3,4,5,6,7,8,9)
123   65                                                                 (0,1,2,3,4,5,6,7,8,9)
123   66                                                                 (0,1,2,3,4,5,6,7,8,9)
123   67[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   74[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   75                                                                 (0,1,2,3,4,5,6,7,8,9)
123   76                                                                 (0,1,2,3,4,5,6,7,8,9)
123   77                                                                 (0,1,2,3,4,5,6,7,8,9)
123   78[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   87[D]                                                              (0,1,2,3,4,5,6,7,8,9)
123   88                                                                 (0,1,2,3,4,5,6,7,8,9)
123   89[D]                                       (7)                    (0,1,2,3,4,5,6,8,9)
123  100[D]                                                              (0,1,2,3,4,5,6,7,8,9)
     Sending Receiving             Packet
Time  Node#    Node#s              & TTL  T_RT_RMP
**********************************************************
129     89     +77,+88,-90,+101     7(0)   RMP
Time Node    TQ                  RTQ                 pacForRMP           DC
 ============================================================================================

After processing nodes with non-empty IQs:
130    2[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130    3                                                                 (0,1,2,3,4,5,6,7,8,9)
130    4                                                                 (0,1,2,3,4,5,6,7,8,9)
130    5                                                                 (0,1,2,3,4,5,6,7,8,9)
130    6[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   13[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   14                                                                 (0,1,2,3,4,5,6,7,8,9)
130   15                                                                 (0,1,2,3,4,5,6,7,8,9)
130   16                                                                 (0,1,2,3,4,5,6,7,8,9)
130   17                                                                 (0,1,2,3,4,5,6,7,8,9)
130   18                                                                 (0,1,2,3,4,5,6,7,8,9)
130   19[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   24[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   25                                                                 (0,1,2,3,4,5,6,7,8,9)
130   26                                                                 (0,1,2,3,4,5,6,7,8,9)
130   27                                                                 (0,1,2,3,4,5,6,7,8,9)
130   28                                                                 (0,1,2,3,4,5,6,7,8,9)
130   29                                                                 (0,1,2,3,4,5,6,7,8,9)
130   30                                                                 (0,1,2,3,4,5,6,7,8,9)
130   31                                                                 (0,1,2,3,4,5,6,7,8,9)
130   32[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   36                                                                 (0,1,2,3,4,5,6,7,8,9)
130   37                                                                 (0,1,2,3,4,5,6,7,8,9)
130   38                                                                 (0,1,2,3,4,5,6,7,8,9)
130   39                                                                 (0,1,2,3,4,5,6,7,8,9)
130   40[S]                                                              (0,1,2,3,4,5,6,7,8,9)
130   41                                                                 (0,1,2,3,4,5,6,7,8,9)
130   42                                                                 (0,1,2,3,4,5,6,7,8,9)
130   43                                                                 (0,1,2,3,4,5,6,7,8,9)
130   44                                                                 (0,1,2,3,4,5,6,7,8,9)
130   43                                                                 (0,1,2,3,4,5,6,7,8,9)
130   44                                                                 (0,1,2,3,4,5,6,7,8,9)
130   45[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   48[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   49                                                                 (0,1,2,3,4,5,6,7,8,9)
130   50                                                                 (0,1,2,3,4,5,6,7,8,9)
130   51                                                                 (0,1,2,3,4,5,6,7,8,9)
130   52                                                                 (0,1,2,3,4,5,6,7,8,9)
130   53                                                                 (0,1,2,3,4,5,6,7,8,9)
130   54                                                                 (0,1,2,3,4,5,6,7,8,9)
130   55                                                                 (0,1,2,3,4,5,6,7,8,9)
130   56[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   61[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   62                                                                 (0,1,2,3,4,5,6,7,8,9)
130   63                                                                 (0,1,2,3,4,5,6,7,8,9)
130   64                                                                 (0,1,2,3,4,5,6,7,8,9)
130   65                                                                 (0,1,2,3,4,5,6,7,8,9)
130   66                                                                 (0,1,2,3,4,5,6,7,8,9)
130   67[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   74[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   75                                                                 (0,1,2,3,4,5,6,7,8,9)
130   76                                                                 (0,1,2,3,4,5,6,7,8,9)
130   77                          (7)                                    (0,1,2,3,4,5,6,7,8,9)
130   78[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   87[D]                                                              (0,1,2,3,4,5,6,7,8,9)
130   88                          (7)                                    (0,1,2,3,4,5,6,7,8,9)
130   89[D]                                       (7)                    (0,1,2,3,4,5,6,8,9)
130  100[D]                                                              (0,1,2,3,4,5,6,7,8,9)
```

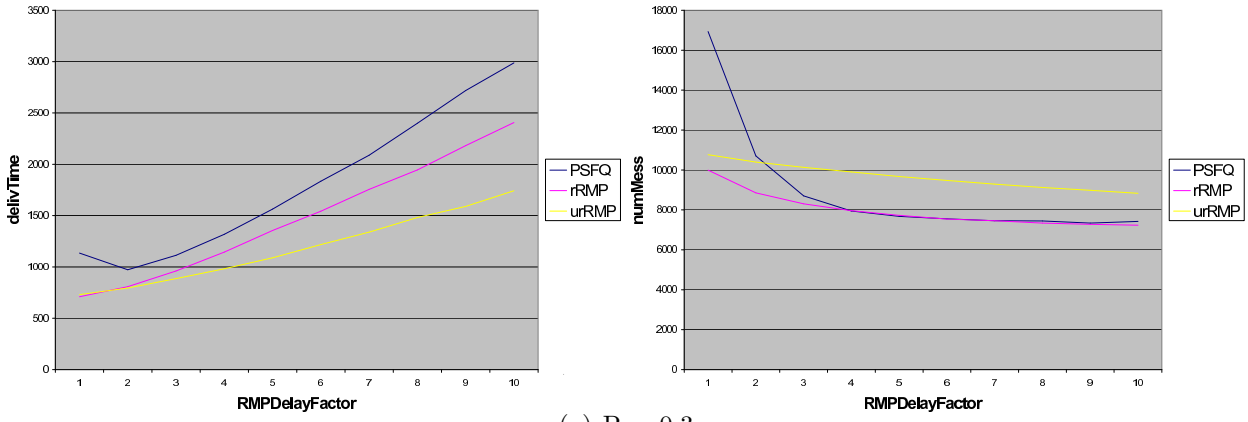
Figure 4.5: Simulation output from time 123 to time 130 in the network in Figure 4.1.

urRMP. We compute the minimum delay t_r for a node x according to t_r = degree(x) $\times$ RMPdelayFactor, where degree(x) is the number of neighbors of x and RMPdelayFactor is a constant. A time point t is considered an *idleTime* if $S'(t) = \emptyset$.

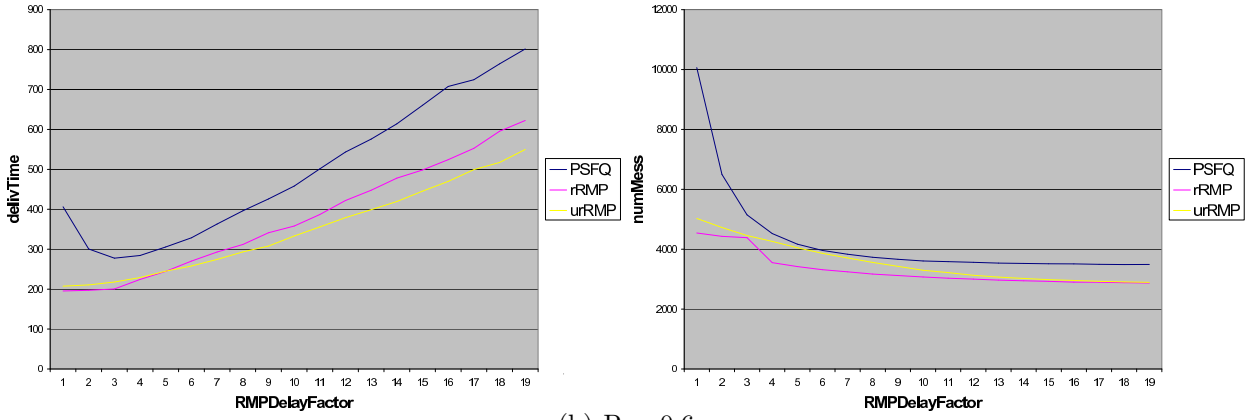
4.2.1 Simulation Results of OSDRMP vs. PSFQ-based Protocols for $s = 40$, $h = 5$ and $n = 10$ Varying RMPdelayFactor and P

Figure 4.6 shows the variation of delivTime and numMess of the OSDRMP(rRMP and urRMP) and the PSFQ-based protocols w.r.t. RMPdelayFactor at $s = 40$, $h = 5$, $n = 10$, and $P = 0.3$, 0.6 and 0.9. For higher P, we use a larger range of RMPdelayFactor to better represent the trend of variation of delivTime and numMess. Both versions of the OSDRMP protocol perform better than the PSFQ-based protocol in terms of both delivTime and numMess when $P = 0.6$ and 0.9. For $P = 0.3$, numMess for the PSFQ-based protocol are initially higher than that for both versions of the OSDRMP protocol but as the RMPdelayFactor increases, the PSFQ-based protocol performs better than the OSDRMP(urRMP) protocol. However, this comes only with a very large increase in delivTime.

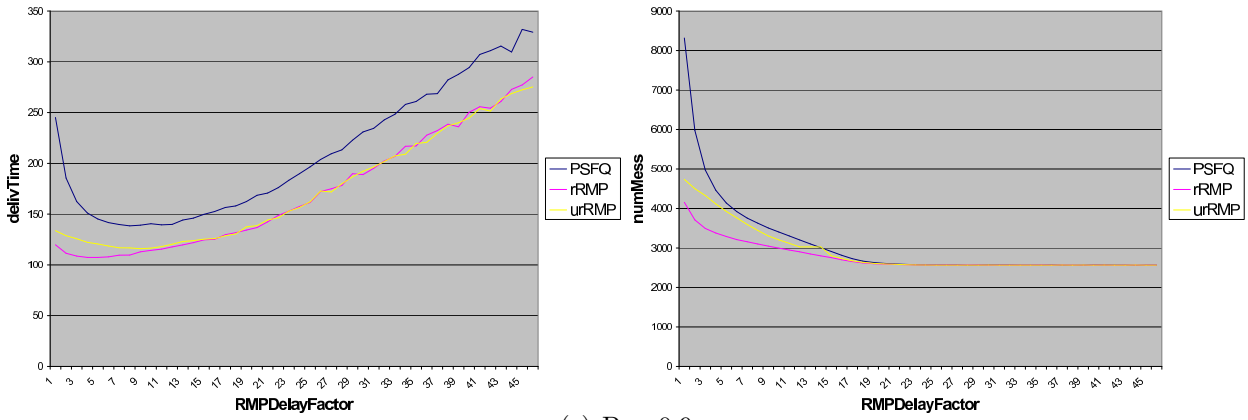
Analysis: delivTime vs. RMPdelayFactor. Consider delivTime in different graphs in Figure 4.6 for different Ps. In each case, delivTime of PSFQ-based protocol first decreases and then increases as we increase the RMPDelayFactor. When RMPDelayFactor is small, the higher priority of RMP in the PSFQ-based protocol does not give enough opportunities for a node x to send Ts and RTs to its neighbors, which can result in more RMPs for some of the neighbors of x. Even if RMPQ(x) = $\emptyset$, frequent competition for sending RMPs from the neighbors of x decreases chances of x being selected for sending Ts or RTs, and this too can lead to more RMPs from the neighbors. This again delays the delivery of Ts and RTs to nodes. The result is higher delivTime. When we increase the RMPDelayFactor, it gives nodes more opportunity to send Ts and RTs reducing number of RMPs and hence, delivTime. On the other hand, when RMPDelayFactor is increased beyond a certain value it results in increasing idleTime because all Ts and RTs have been sent and nodes cannot send RMPs due to delay requirements, and this increases the delivTime. For a given RMPDelayFactor, the increase in P causes fewer RMPs and this decreases the



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

Figure 4.6: Comparison of OSDRMP(rRMP and urRMP) and PSFQ-based protocols for s=40, h=5, n=10 varying RMPdelayFactor and P.

delivTime. Note that the minimum delivTime occurs at a higher RMPDelayFactor as P increases. This is because at higher P, increase in RMPDelayFactor benefits propagation of successful Ts and RTs (which are more at higher P) throughout the network.

In the OSDRMP protocol, T and RT have priority. So for low RMPDelayFactor, the delay of Ts and RTs due to frequent transmission of RMPs is less than that in case of the PSFQ-based protocol and hence, initial increase in RMPDelayFactor benefits the OSDRMP protocol's delivTime much less than that of the PSFQ-based protocol. At $P =$ 0.3, number of Ts and RTs is low and hence, priority of Ts and RTs at low RMPDelay-Factors is sufficient to ensure their propagation in the network. So at P =0.3, there is no decrease in delivTime with increase in RMPDelayFactor. Like the PSFQ-based protocol, at higher P, increase in RMPDelayFactor benefits propagation of successful Ts and RTs through the network by ensuring nodes with Ts and RTs compete less with RMPs for sending messages in the network. In both versions of the OSDRMP protocol, increase in idleTime contributes to increase in delivTime at higher RMPDelayFactor (there are more Ts and RTs than in case of lower P) in comparison to that at lower P. Initially OS-DRMP(rRMP) performs better than OSDRMP(urRMP) but as RMPDelayFactor increases performance of OSDRMP(urRMP) betters that of OSDRMP(rRMP). The reason is that at low RMPDelayFactor, OSDRMP(urRMP) has more RMPs for different packets in compar-ison to OSDRMP(rRMP) and this leads to more competition for nodes with Ts and RTs in case of the former protocol. However, as RMPDelayFactor increases, Ts and RTs get more chances to propagate and OSDRMP(urRMP) reduces the possibility of idleTime by sending more RMPs which also increases the chances of nodes getting packets faster and thus reducing delivTime at lower P. But as P increases, packets reach nodes successfully and hence, the number of packets for which RMPs are to be sent at nodes reduce and thus OSDRMP(urRMP) is not able to reduce idleTime as effectively as in case of lower P. Also, OSDRMP(rRMP) allows successful Ts and RTs to propagate at higher P and hence, the difference in delivTime between OSDRMP(urRMP) and OSDRMP(rRMP) decreases as P increases. It is important to note that minimum delivTime does not indicate minimum numMess.

Analysis: numMess vs. RMPdelayFactor. In Figure 4.6, for a low RMPDelay-

Factor, highest priority to RMP and frequent transmission of RMPs in the PSFQ-based protocol with delayed Ts and RTs leading to more RMPs, add up to a high numMess at all P. With increase in RMPDelayFactor, Ts and RTs have enough time to reach nodes thus reducing the number of RMPs and the numMess for the PSFQ-based protocol. For low RMPDelayFactor, both versions of the OSDRMP protocol perform better than the PSFQ-based protocol as T and RT have higher priority in the former. For $P = 0.3$, unsuccessful transmissions coupled with urRMP and nodes requesting packets which none of their neighbors have (due to OS) contribute to high numMess for the OSDRMP(urRMP) protocol compared to that for the PSFQ-based protocol. The rRMP in the OSDRMP(rRMP) protocol eliminates any potential disadvantage of OS and thus, the numMess becomes almost equal to that of the PSFQ-based protocol. For $P = 0.6$ and 0.9, there are more and more successful Ts and RTs and OSDRMP(urRMP) does not have as many RMPs to send as in the case of $P = 0.3$. Its behavior approaches that of OSDRMP(rRMP) and hence, the numMess required for both versions of the OSDRMP protocol with increasing RMPDelayFactor is almost equal and at times better than that required for the PSFQ-based protocol. We do not obtain a clear benefit in terms of numMess unlike delivTime because the disadvantages of OS forwarding counterbalance the disadvantages of IS forwarding by sending too many RMPs even when neighbors of a node do not have the packets. There is a RMPDelayFactor at each P for each protocol after which numMess do not change with increase in RMPDelayFactor. Increasing RMPDelayFactor does not benefit this situation and only contributes to idleTime. The RMPDelayFactor after which numMess becomes constant increases with increase in P for both the OSDRMP(rRMP) protocol and the PSFQ-based protocol. This is because there are more successful Ts and RTs at higher P and increase in RMPDelayFactor helps in propagation of successful transmissions. For the OSDRMP(urRMP) protocol, the RMPDelayFactor after which numMess become constant is very high at $P = 0.3$.

4.2.2 Simulation Results of OSDRMP vs. PSFQ-based Protocols for $s = 40$, $h = 5$ and RMPdelayFactor = 3 Varying Number of Packets (n) and P

Figure 4.7 shows the variation of delivTime and numMess of the OSDRMP(rRMP and urRMP) and the PSFQ-based protocols w.r.t. n at $s = 40$, $h = 5$, RMPdelayFactor = 3, and $P = 0.3$, 0.6 and 0.9. Both versions of the OSDRMP protocol perform better than the PSFQ-based protocol in terms of both delivTime and numMess when $P = 0.3$, 0.6 and 0.9.

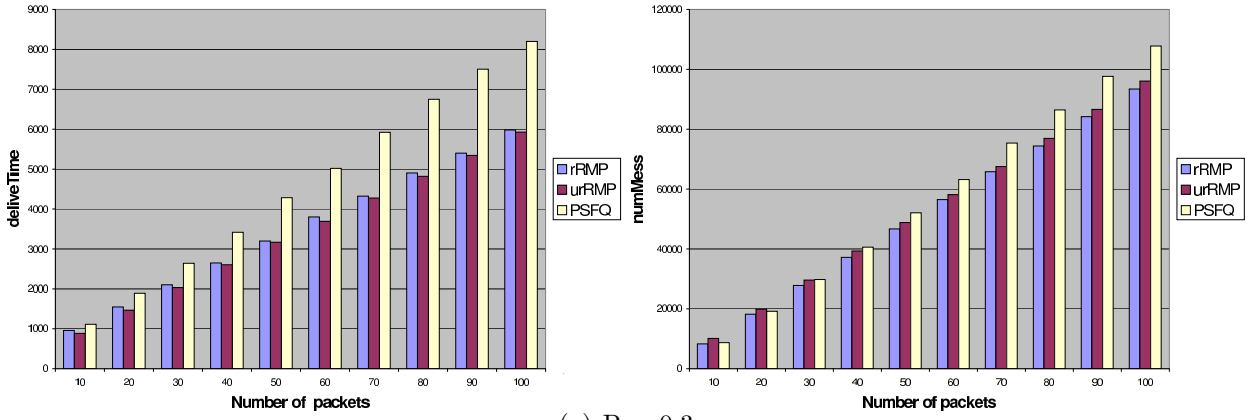
Analysis: delivTime vs. number of packets. Consider the different graphs in Figure 4.7 for different Ps. In each case, delivTime of the PSFQ-based protocol and both versions of the OSDRMP protocol increase with increase in number of packets. This is expected because the increase in the number of packets to be delivered increase the number of messages exchanged in the process which results in more delivTime.

The delivTime of the PSFQ-based protocol is always more than the delivTime of both versions of the OSDRMP protocol and this difference increases with increase in the number of packets because the PSFQ-based protocol takes more messages to deliver the packets in comparison to the OSDRMP protocol due to the disadvantages of IS-forwarding ($RMP > RT > T$) in comparison to OS-forwarding ($T > RT > RMP$) of packets at nodes.

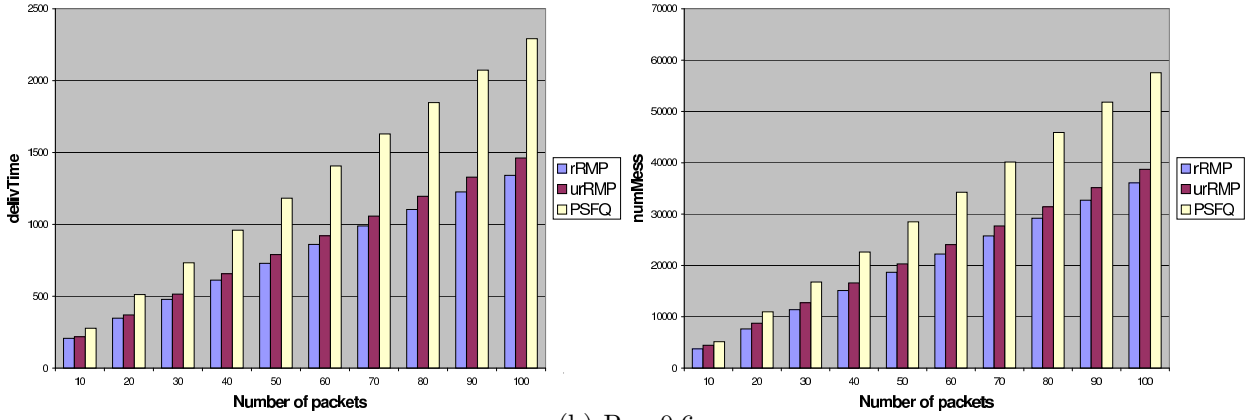
The delivTime of both versions of OSDRMP protocol are close to each other even with increase in the number of packets. The reasons for this have been mentioned in Section 4.2.1.

Analysis: numMess vs. number of packets. In the different graphs in Figure 4.7 for different Ps, the numMess for PSFQ-based protocol and both versions of the OSDRMP protocol increase with increase in the number of packets. This is expected as the increase in number of packets to be delivered increase the number of messages exchanged in the process.

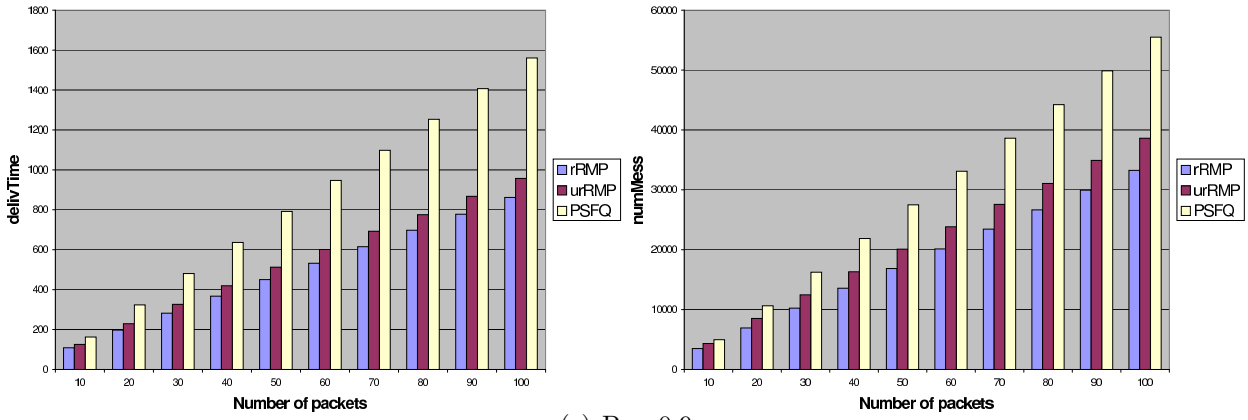
The numMess for the PSFQ-based protocol is always more than the numMess for both versions of the OSDRMP protocol and this difference increases with increase in number of packets. This is due to the benefits of OS-forwarding ($T > RT > RMP$) over IS-forwarding ($RMP > RT > T$) of packets at nodes. When number of packets is low i.e. $n = 10$ and 20 at $P = 0.3$, OSDRMP(urRMP) takes more number of messages in comparison to PSFQ-based protocol because unsuccessful transmissions coupled with urRMP and nodes



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

Figure 4.7: Comparison of OSDRMP(rRMP and urRMP) and PSFQ-based protocols for s=40, h=5, RMPdelayFactor=3 varying number of packets (n) and P.

requesting packets which none of their neighbors have (due to OS) contribute to the high numMess. However, as number of packets increases, the advantages (in terms of numMess) of OS-forwarding overcome the disadvantages of urRMP.

The numMess of both versions of OSDRMP protocol are close to each other even with increase in number of packets with the reduced number of RMPs in the rRMP version resulting in slightly lower numMess w.r.t to the urRMP version.

4.2.3 Simulation Results of OSDRMP vs. PSFQ-based Protocols for $s = 0$, $h = 14$ and $n = 10$ Varying RMPdelayFactor and P

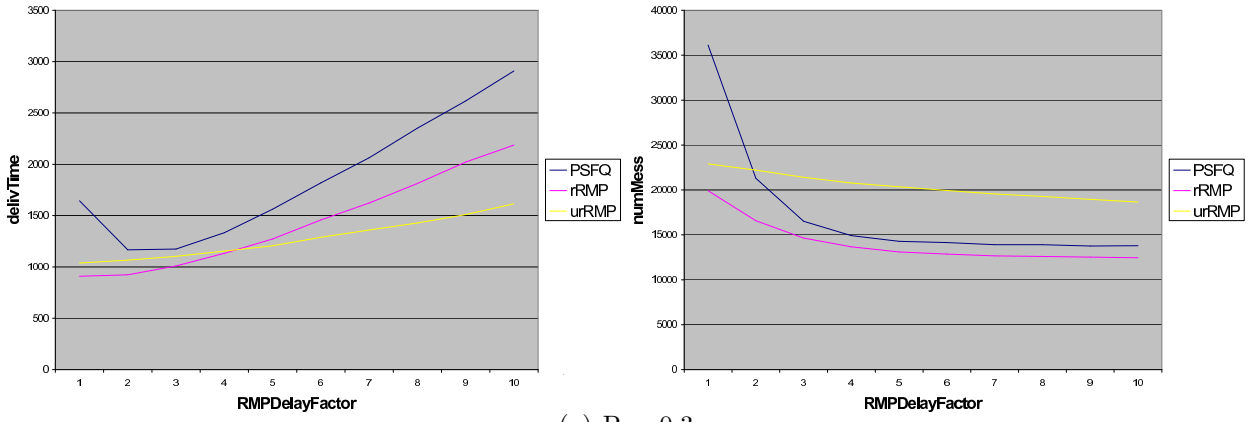
Figure 4.8 shows the variations of delivTime and numMess w.r.t. RMPDelayFactor for the OSDRMP protocol(rRMP and urRMP) and the PSFQ-based protocol for $P = 0.3$, 0.6 and 0.9. Here, the same conclusions as those in Figure 4.6 hold, except that there are more messages in Figure 4.8 (because of larger h) which also causes the increase in idleTime to start at a higher RMPDelayFactor.

4.2.4 Simulation Results of OSDRMP vs. PSFQ-based Protocols for $s = 0$, $h = 14$ and RMPdelayFactor = 3 Varying Number of Packets (n) and P

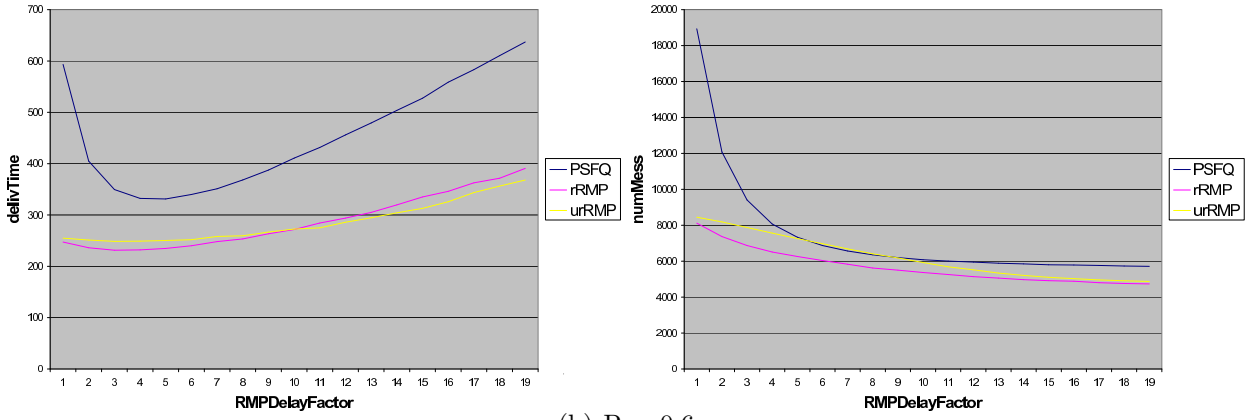
Figure 4.9 shows the variations of delivTime and numMess w.r.t. n for the OSDRMP protocol(rRMP and urRMP) and the PSFQ-based protocol for $P = 0.3$, 0.6 and 0.9. Here, the same conclusions as those in Figure 4.7 hold, except that there are more messages in Figure 4.9 because of larger h.

4.2.5 Simulation Results Demonstrating the Time Taken to Fill the DC of Nodes during Execution of OSDRMP(rRMP) Protocol.

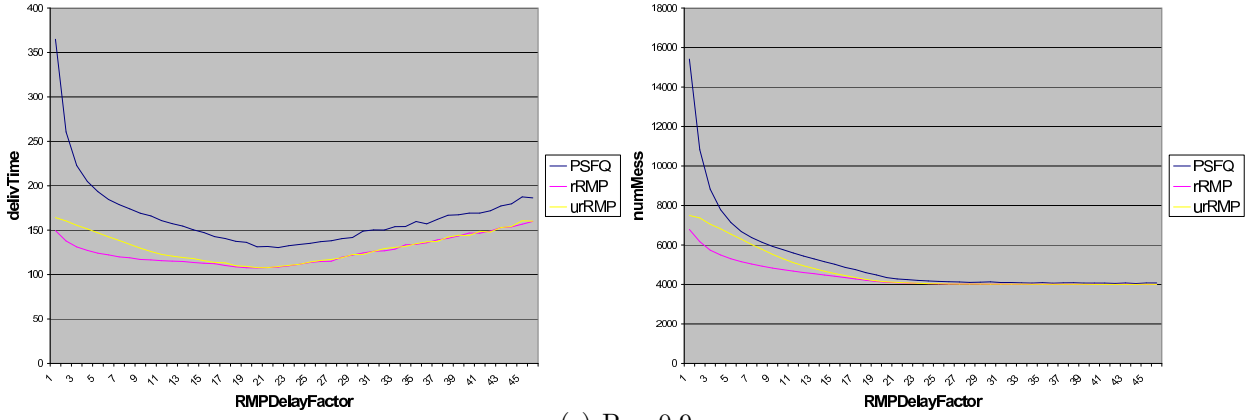
We run simulations using OSDRMP(rRMP) with the the source-destination pairs (i)$s = 40$, $h = 5$ and (ii)$s = 0$, $h = 14$ at $n = 10$ and 3 different probabilities of successful transmission ($P = 0.3$, 0.6 and 0.9) in the 12x9 grid network in Figure 4.1 and observe the time taken by the different nodes in the network between the source and destinations to fill their DC. The results are plotted in the Tables 4.1 till 4.6.



(a) P = 0.3

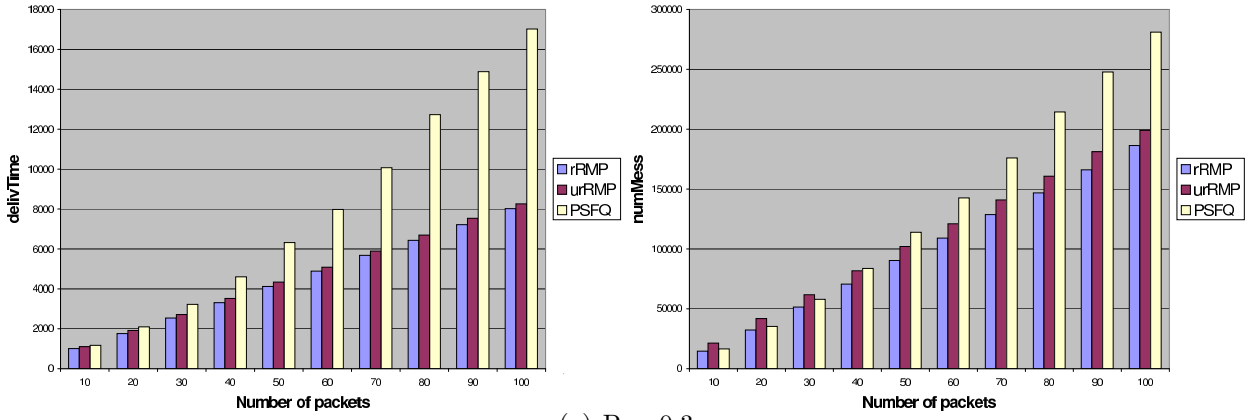


(b) P = 0.6

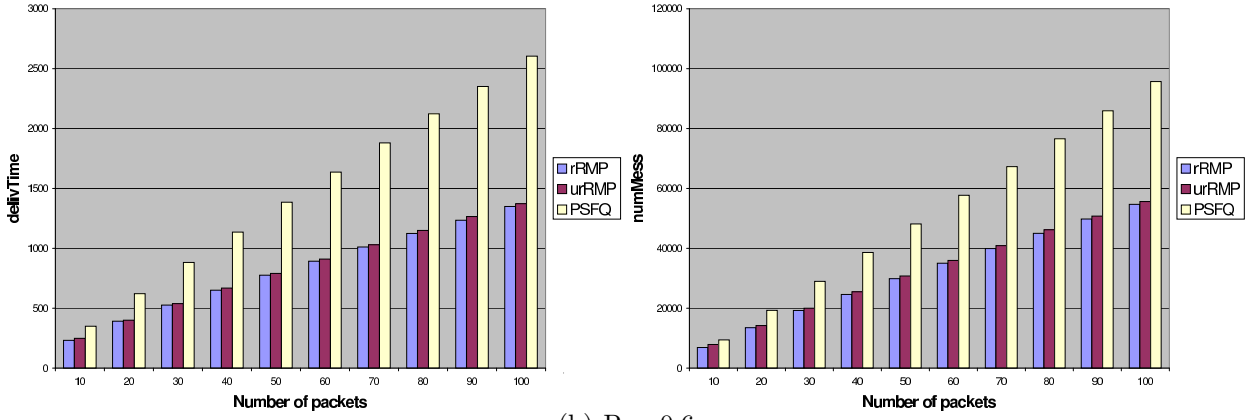


(c) P = 0.9

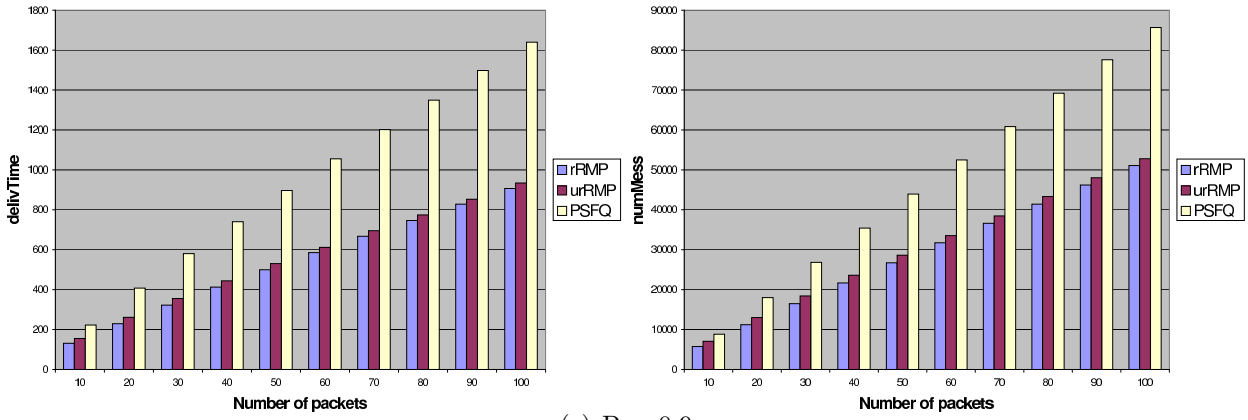
Figure 4.8: Comparison of OSDRMP(rRMP and urRMP) and PSFQ-based protocols for s=0, h=14, n=10 varying RMPdelayFactor and P.



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

Figure 4.9: Comparison of OSDRMP(rRMP and urRMP) and PSFQ-based protocols for s=0, h=14, RMPdelayFactor=3 varying number of packets (n) and P.

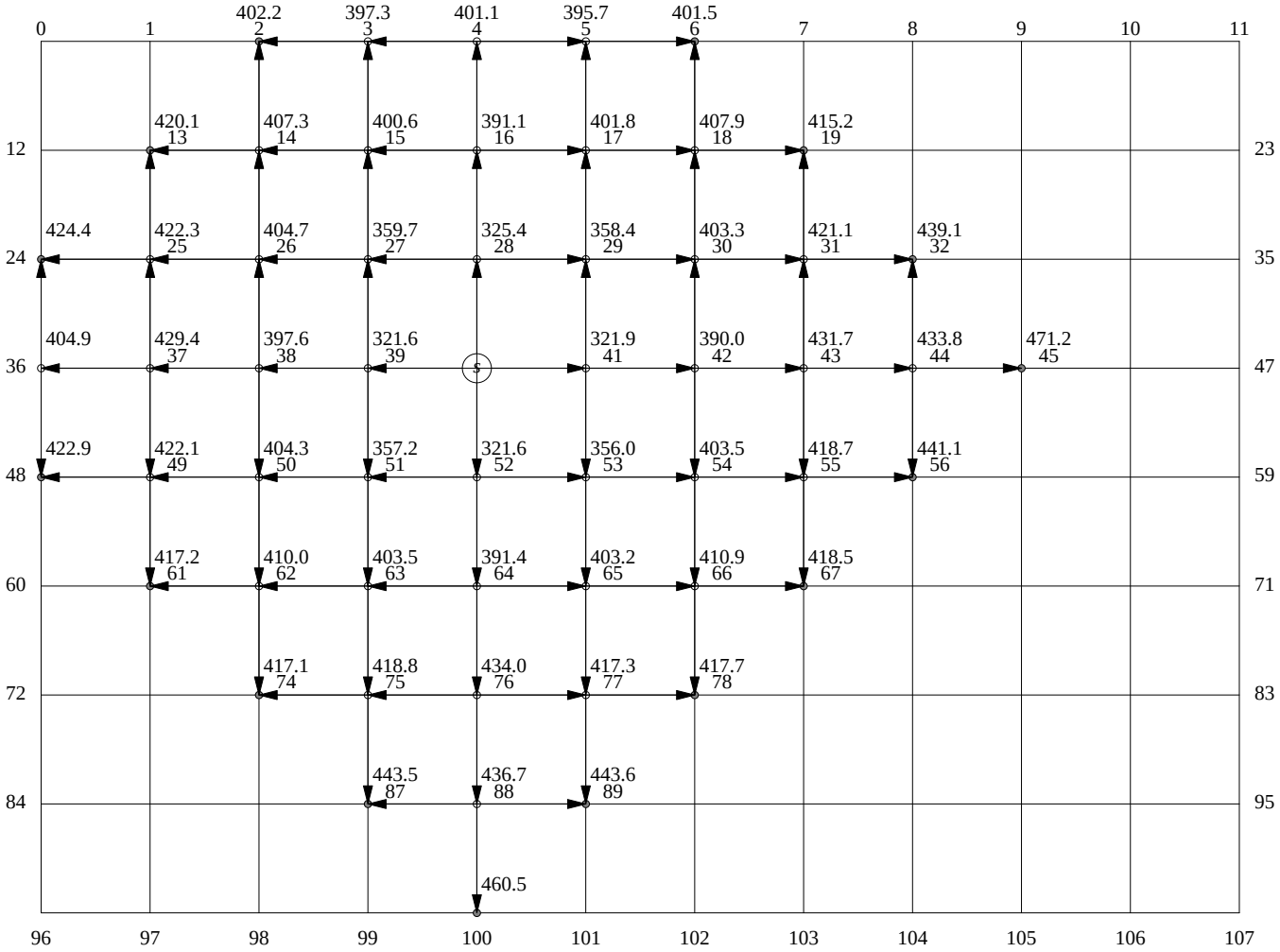


Figure 4.10: Time taken to fill the DC of nodes for $s = 40$, $h = 5$, $n = 10$, RMPDelayFactor = 1 and $P = 0.3$.

Figure 4.10 shows the time taken by each node to fill its DC when $s = 40$, $h = 5$, $n = 10$, RMPDelayFactor = 1 and $P = 0.3$. From the figure we see that the time taken to fill a node x's DC has dependencies on the following:

- Number of shortest paths between the base station (s) and x. It is given by the following equation:

$$numPaths = \binom{m+n}{m} \tag{4.1}$$

  where (m,n) are coordinates of x and the coordinates of s is (0,0) and each distance of 1 hop in either direction increases the corresponding coordinates by 1.

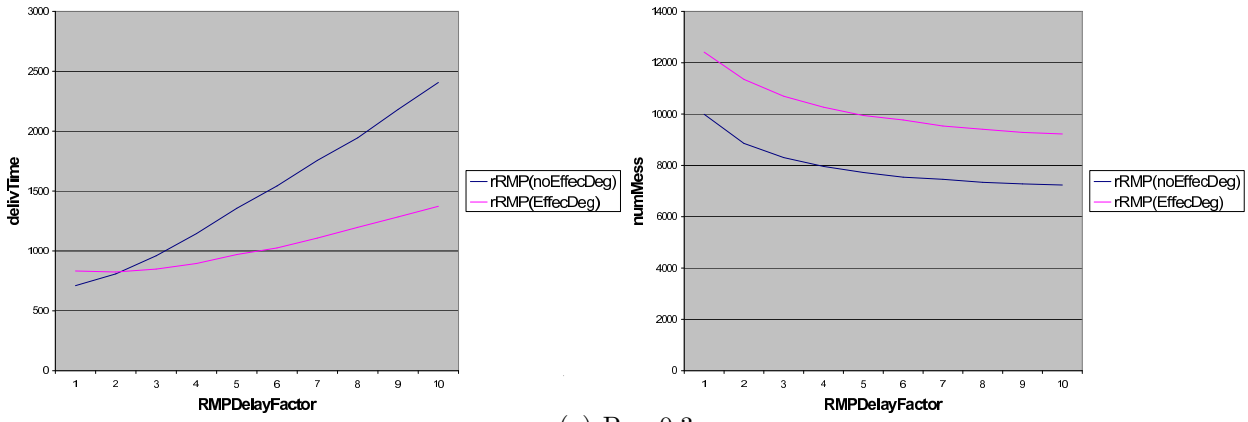
- Number of upstream neighbors of x via shortest paths from s.

- Degree of x.

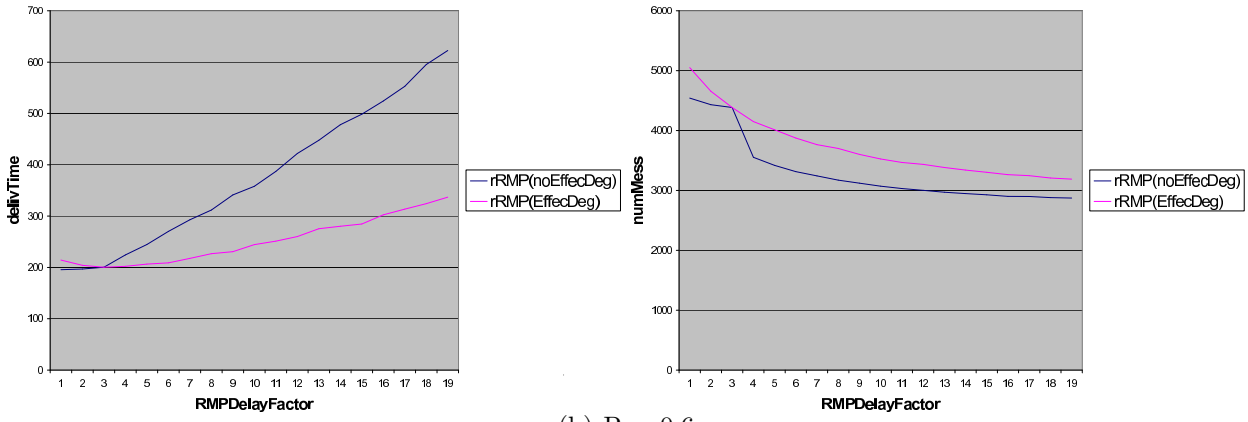
4.2.6 Simulation Results of Varying the Methods of Calculation of Delay in Sending RMPs in OSDRMP(rRMP) Protocol

We run our simulations for the 12x9 grid in Figure 4.1 and observe the effects of OSDRMP(rRMP) protocol with and without effective degree calculations on delivTime and numMess w.r.t RMPDelayFactor for (i) $s = 40$, $h = 5$ and (ii) $s = 0$, $h = 14$ at $P = 0.3$, 0.6 and 0.9 and $n = 10$. The reasons for choosing the OSDRMP(rRMP) protocol is given at the beginning of this chapter. In Figures 4.11 and 4.12, OSDRMP(rRMP) without effective degree and with effective degree are referred to as rRMP(noEffecDeg) and rRMP(EffecDeg) respectively.

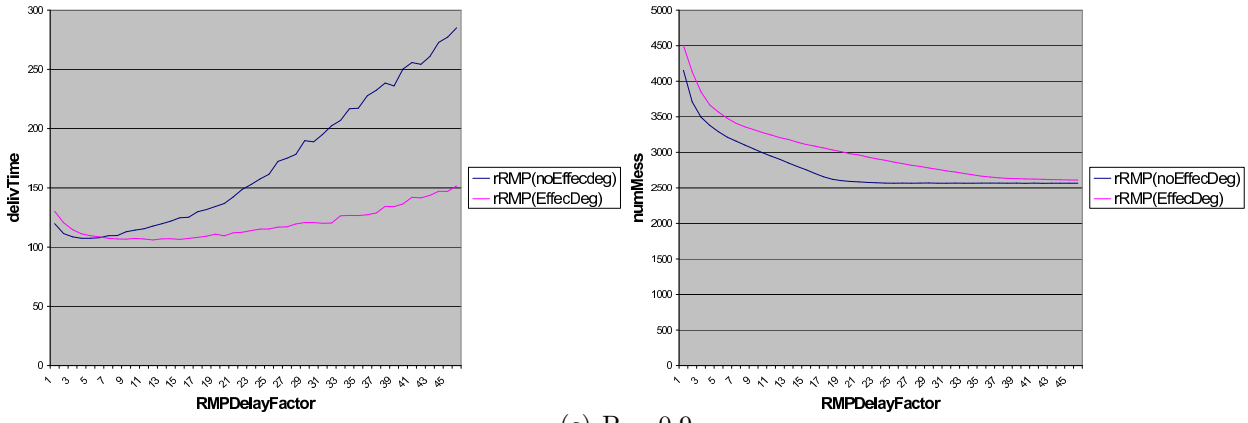
The simulations in the Figures 4.11 and 4.12 show that the OSDRMP(rRMP) protocol with effective degree initially takes more delivTime in comparison to the OSDRMP(rRMP) protocol with no effective degree and then, with increase in RMPDelayFactor its performance becomes better than the latter for all P but always takes more number of messages. Initially, the OSDRMP(rRMP) protocol with effective degree takes more delivery time with increase in RMPDelayFactor. This is because when RMPDelayFactor is small too many RMPs decrease the chances of nodes with Ts and RTs from being selected. However when RMPDelayFactor is increased beyond a certain point, the idleTime observed in the OSDRMP(rRMP) protocol with no effective degree is effectively utilized by the OSDRMP(rRMP) protocol with effective degree by sending more RMPs based on effective degree and not waiting unnecessarily. Effective degree helps in reduction of the unnecessary waiting time to send RMPs by not taking into account nodes from where there are less possibilities of new packets and hence, the OSDRMP(rRMP) protocol with effective degree uses more RMPs. This effectively reduces the idleTime to make requests and receive packets and thus lowers the delivTime. However, it increases the number of RMPs and hence, there are more number of messages in comparison to the OSDRMP(rRMP) protocol with no effective degree. This results in nodes receiving the necessary RTs and thus, in destination nodes receiving the total packets faster in comparison to other version without effective degree.



(a) P = 0.3

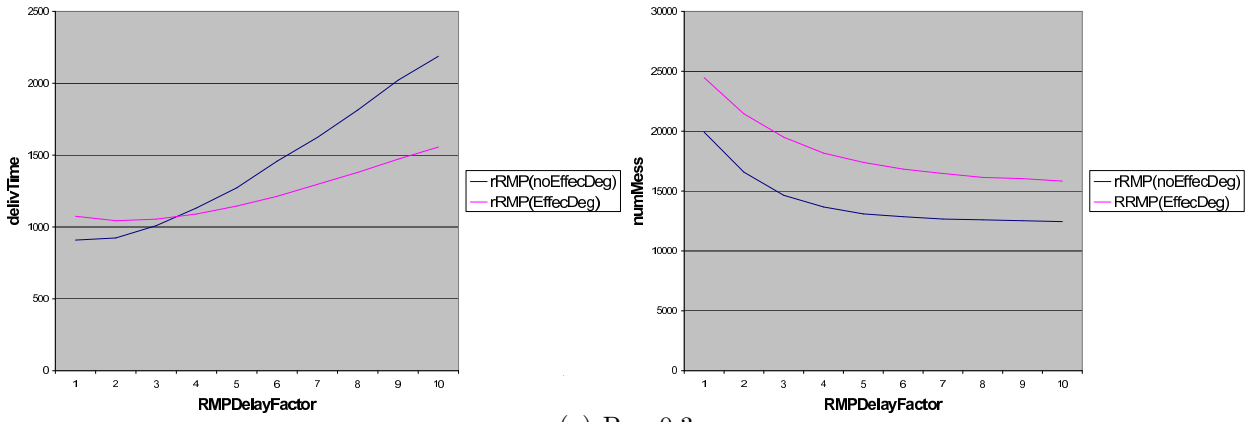


(b) P = 0.6

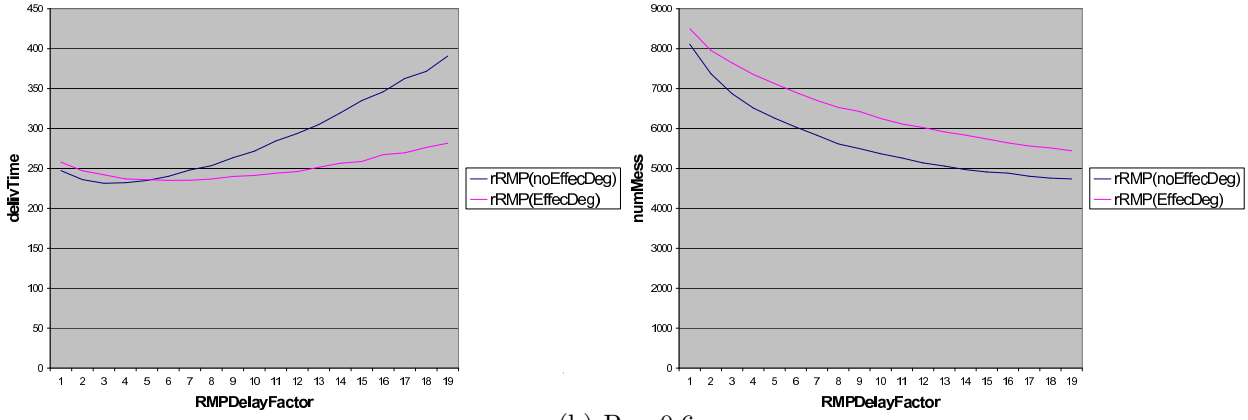


(c) P = 0.9

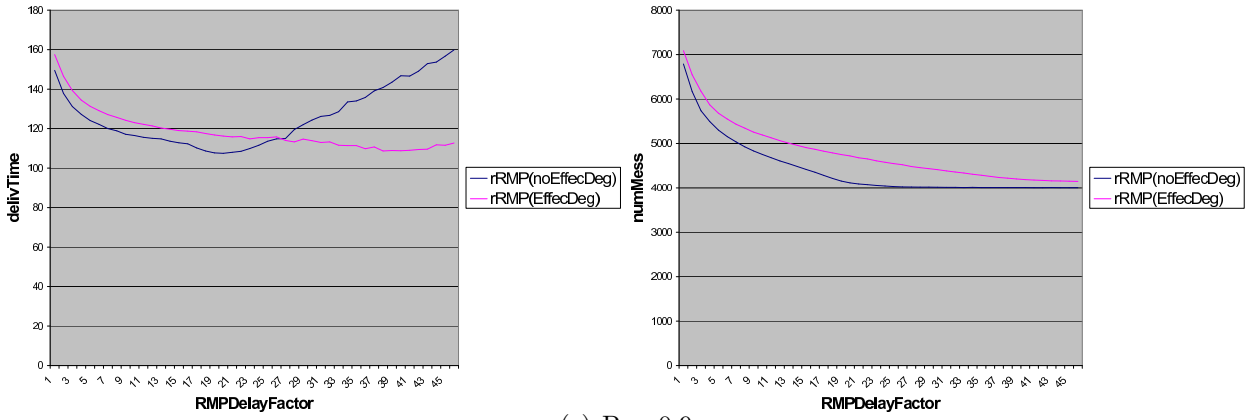
Figure 4.11: Performance of OSDRMP(rRMP) for two different methods of calculating the delay in sending RMPs (t_r) and s=40, h=5, n=10.



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

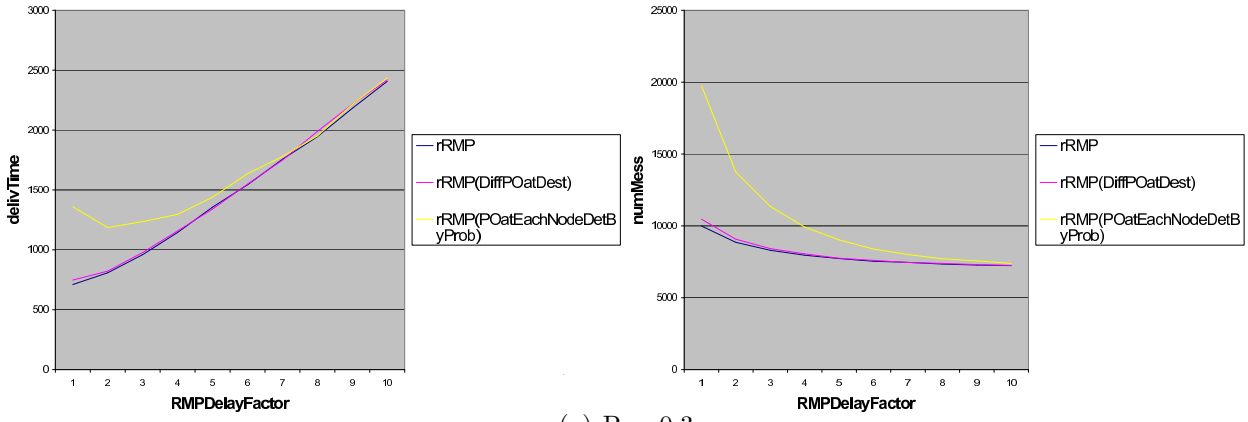
Figure 4.12: Performance of OSDRMP(rRMP) for two different methods of calculating the delay in sending RMPs (t_r) and s=0, h=14, n=10.

4.2.7 Simulation Results of Varying the Priority Orders for Sending Different Types of Messages at Nodes in OSDRMP(rRMP) Protocol

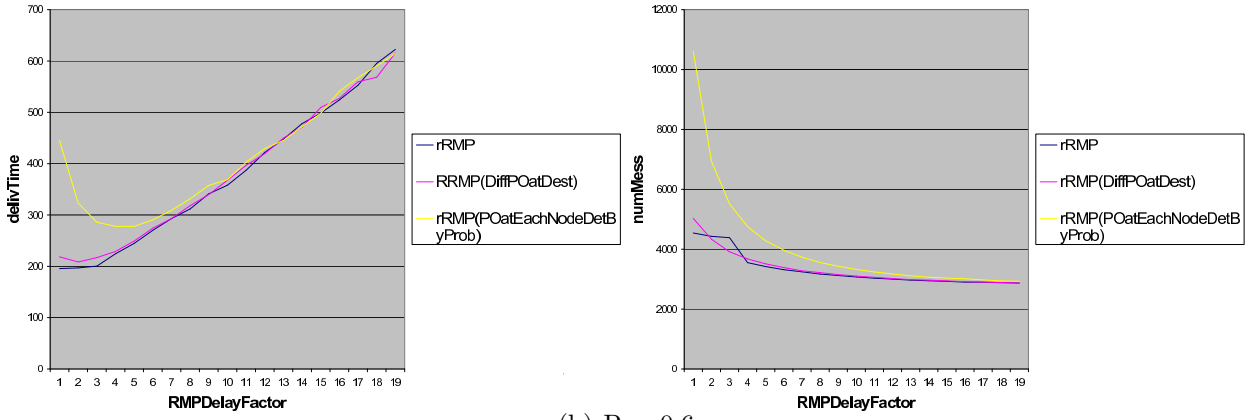
For reasons mentioned in the beginning of this chapter, we choose to implement the OSDRMP(rRMP) protocol and then vary the message priority orders at the nodes in the network in Figure 4.1. The source-destination pairs, number of packets and probabilities remain the same as in the previous section. Both delivTime and numMess are plotted against RMPDelayFactor for the different cases of priority orders (mentioned in Section 3.1.4) as given below:

- Message priority order at all nodes is $T > RT > RMP$. In the graph, it is denoted by rRMP.
- Message priority order at all nodes except destination nodes is $T > RT > RMP$. At destination nodes, the order is $RMP > RT > T$. In the graph, it is denoted by rRMP(DiffPOatDest). We will refer to it as OSDRMP(rRMP) with DiffPOatDest.
- Message priority order at all nodes is determined probabilistically. The probability that each node x (selected for transmission) with eligibility to send an RMP can send one is given by $1 - \frac{nodeTTL(x)}{h}$ where h is the minimum distance between the source and a destination. If x is not selected for RMP, it sends a T if TQ(x) $\neq \emptyset$, else it sends an RT. In the graph, it is denoted by rRMP(POatEachNodeDetByProb). We will refer to it as OSDRMP(rRMP) with POatEachNodeDetByProb.

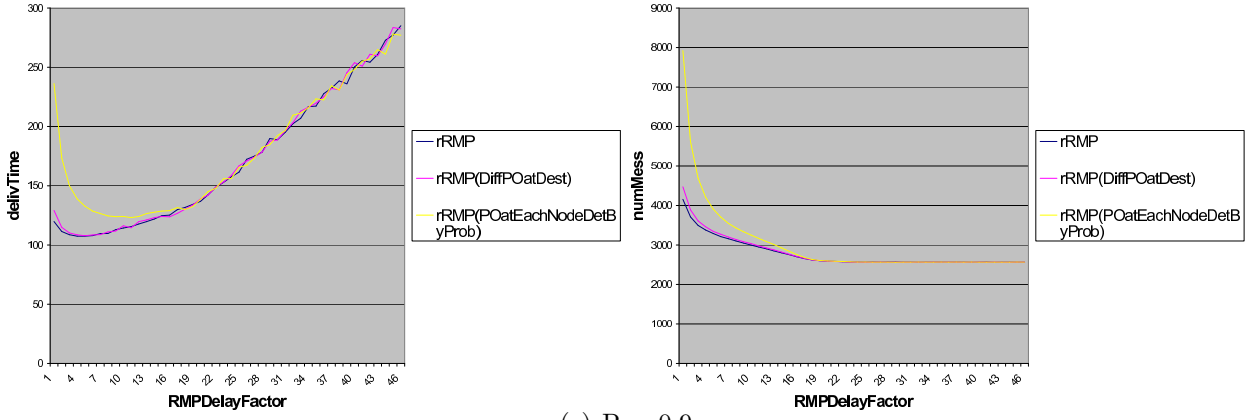
We see from the Figures 4.13 and 4.14 that there is no real gain in terms of delivTime and numMess by implementing OSDRMP(rRMP) protocols with DiffPOatDest and with POatEachNodeDetByProb respectively. While OSDRMP(rRMP) protocol with DiffPOatDest aims to save time by ensuring that destination nodes request and get packets before sending Ts and RTs, it does not differ from OSDRMP(rRMP) w.r.t delivTime because priority on Ts and RTs at the destination nodes in the latter help neighbors get packets faster and distribute them to other potential destination nodes which in turn saves unnecessary RMPs and RTs and hence, delivTime. Even if destination nodes get all packets a little faster, the net impact on delivTime is less as there are no downstream nodes for destination



(a) P = 0.3

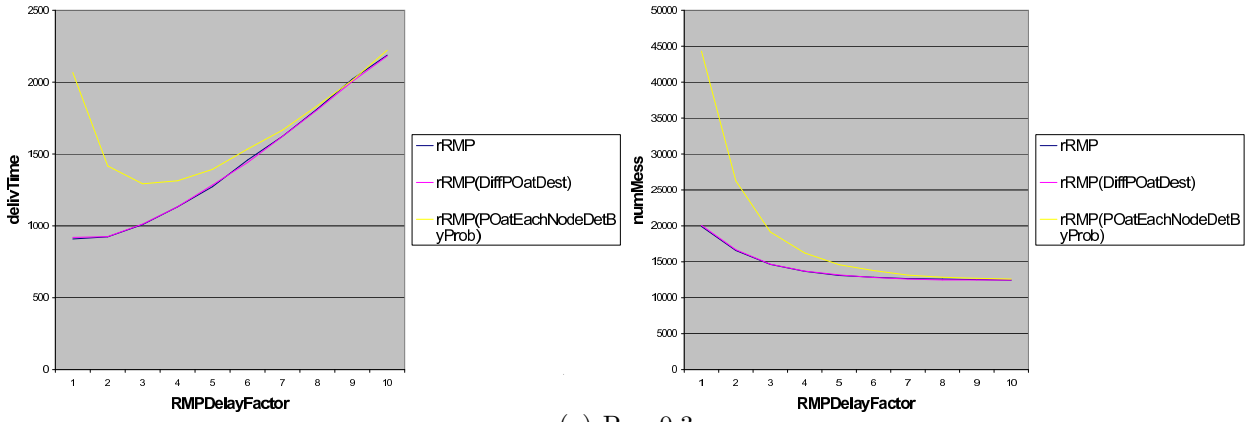


(b) P = 0.6

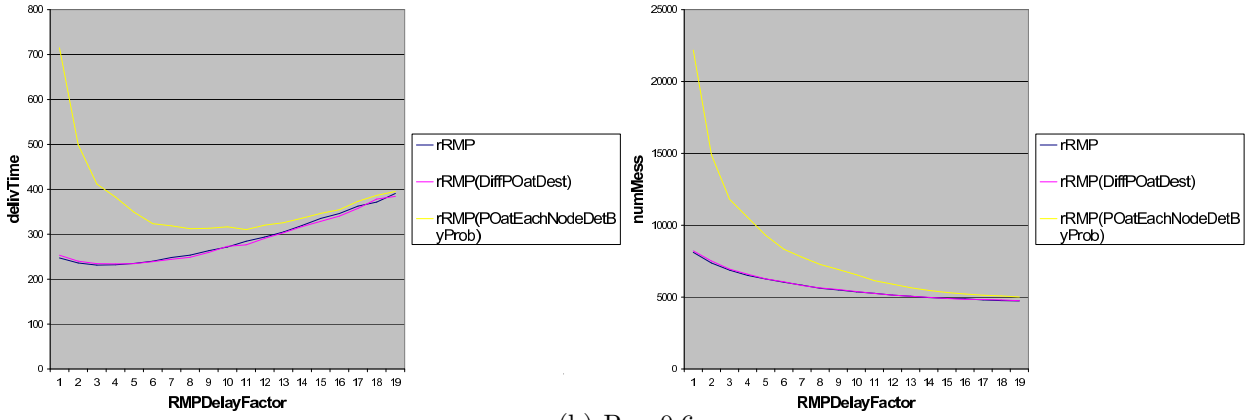


(c) P = 0.9

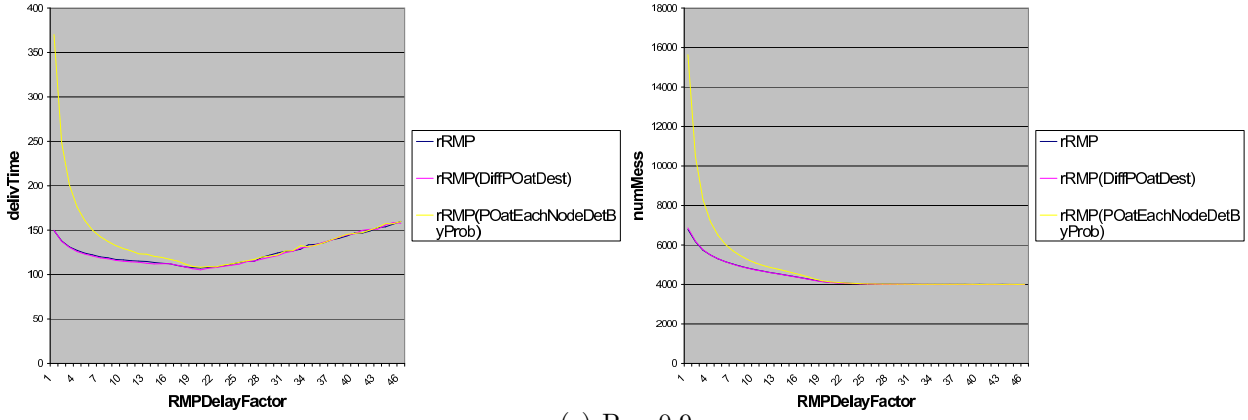
Figure 4.13: Performance of OSDRMP(rRMP) for three different methods of determining the priority order for sending different types of messages at nodes and s=40, h=5, n=10.



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

Figure 4.14: Performance of OSDRMP(rRMP) for three different methods of determining the priority order for sending different types of messages at nodes and s=0, h=14, n=10.

nodes in which case propagation would have been faster and benefitted the delivTime better.The numMess too, is the same for both cases as in both destination nodes have to send all of the Ts, RTs and RMPs. However, OSDRMP(rRMP) protocol with POatEachNodeDetByProb performs worse than both OSDRMP(rRMP) protocol with DiffPOatDest and OSDRMP(rRMP) protocol because the possibility of putting priority on sending RMPs on nodes counteracts the benefits of OS forwarding by delaying the sending of Ts and RTs to nodes which increases delivTime in which the destination nodes receive the packets and also, results in the increase of RMPs and RTs. However, as RMPDelayFactor increases, Ts and RTs have more chances to propagate and this along with OS forwarding makes the behavior of this protocol similar to that of the other two protocols.

4.2.8 Simulation Results Demonstrating the Number of New Packets Sent by Upstream Neighbors of Nodes during Execution of OSDRMP(rRMP) Protocol

Each node x receives data packets via Ts and RTs from its neighbors. A portion of the RTs may be in response to the RMPs sent by x to its neighbors while the rest may be in response to RMPs sent by other nodes. Some of the packets received at x are new packets which it has not received before while the rest are duplicates.

Normally, we expect the upstream neighbors of x to receive new packets before x does. However, it is possible that a packet p_i arrives at x faster than it arrives at y, an upstream neighbor of x because of link failures. Hence, by this time x, a downstream neighbor of y can transmit p_i to y. This example demonstrates that downstreams neighbors of a node can also send new packets to it. We run simulations with $s = 40$, $h = 5$ and $n = 100$ at 3 different probabilities of successful transmission ($P = 0.3$, 0.6 and 0.9) and at 2 different RMPDelayFactors (= 1 and 3) in the 12x9 grid network in Figure 4.1 to observe the number of new packets sent by upstream neighbors of each node. The intention is to see whether upstream nodes are responsible for most of the new packets at a node. This can be useful in methods for reduction of RMPs. If a node x receives considerable more new packets from its upstream neighbors than its downstream ones, then x can ignore RMPs from any upstream neighbor y. Because y can receive its missing packets from its upstream neighbors.

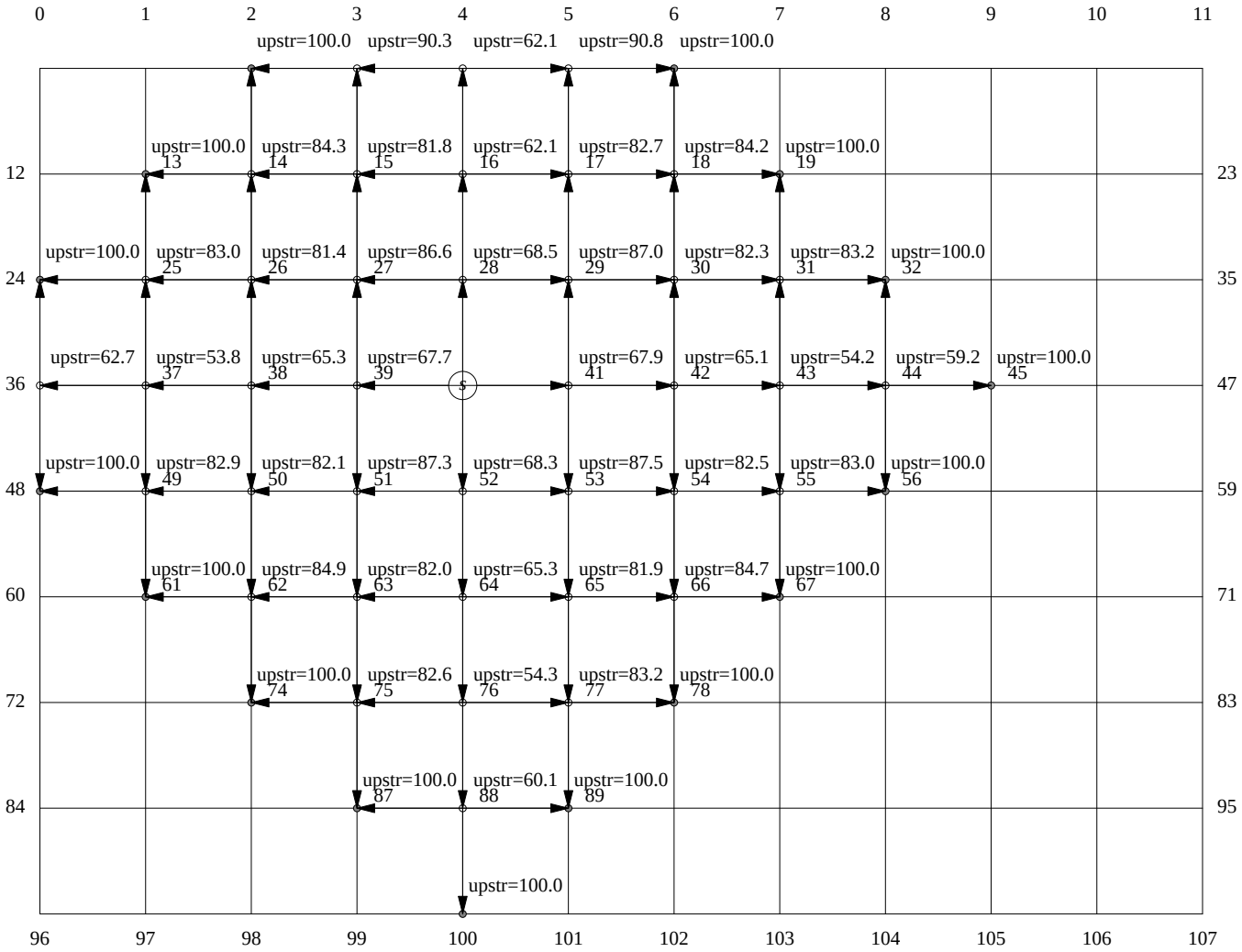


Figure 4.15: Number of packets arriving via upstream nodes for $s = 40$, $h = 5$, $n = 100$, RMPdelayFactor = 3.0 and $P = 0.3$.

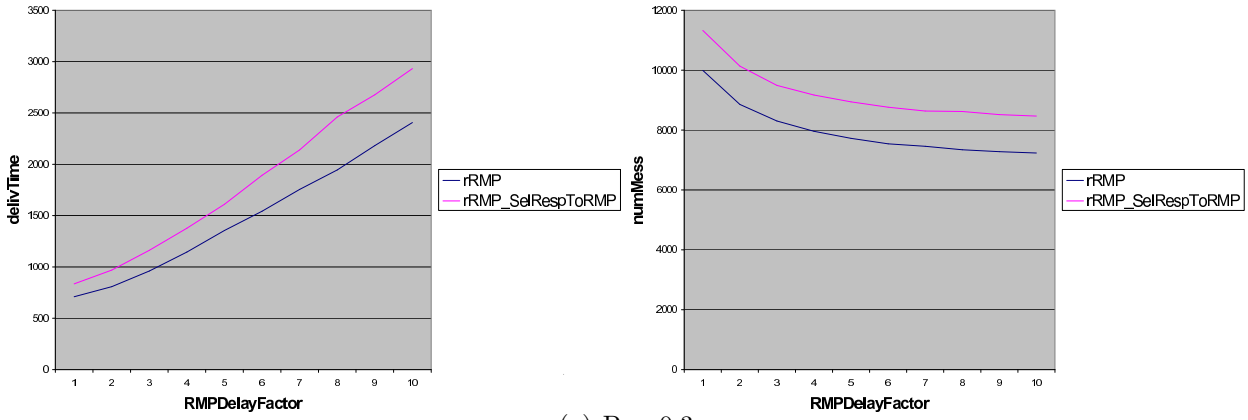
The results of the simulations are plotted in Tables 4.7 and 4.8. The term "packetTTL $\geq$ nodeTTL" implies that a node x has received packets with packetTTL greater than or equal to nodeTTL(x) i.e. from upstream neighbors of x and the term "packetTTL $<$ nodeTTL" implies that x has received packets from its downstream neighbors. However, nodeTTL(x) may change over time. So a packet may arrive at x via an upstream path which can become a downstream path for x later on. The data in the tables are collected under these circumstances and validate our conclusions. We also collect separate data to check how many packets actually arrive via the shortest path between the source(base station) and each node for the different cases mentioned above. The results are almost the same as those in Tables 4.7 and 4.8. The results for $s = 40$, $h = 5$, $n = 100$, RMPdelayFactor = 3.0 and $P = 0.3$ are presented in the Figure 4.15.

In Figure 4.15, nodes 28, 39, 41 and 52 at distance of 1 hop from the source receive approximately the same number of packets via upstream neighbors. They each have only 1 upstream neighbor and 3 downstream ones. Of nodes whose shortest distance from source is 2 hops, nodes 27, 29, 51 and 53 have exactly 2 upstream neighbors and 2 downstream ones and get approximately the same number of packets from upstream neighbors. The other set of nodes at a distance 2 i.e. 16, 38, 42 and 64 have only 1 upstream neighbor and 3 downstream ones, and receive approximately the same number of packets which are less that received by the other set of nodes at a distance 2 probably because they have only one upstream neighbor. This pattern is repeated throughout the network with the nodes having the same number of upstream and downstream neighbors at the same distance from the source receiving approximately the same number of packets from upstream nodes. All destination nodes receive all their packets from upstream neighbors as they have no downstream paths to themselves.

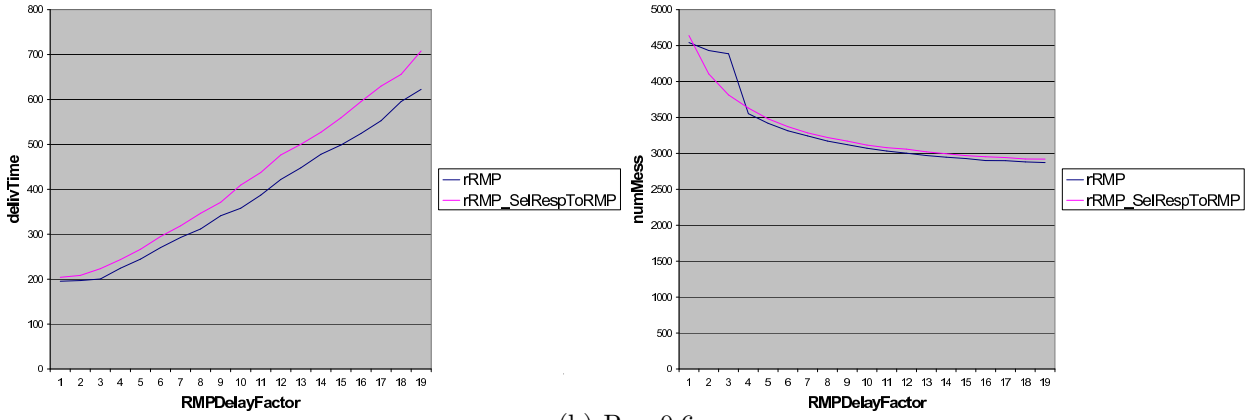
The results in the tables and the figure show that most of the new packets at x are received from its upstream neighbors. Hence, it is feasible to modify the protocol so that the node responds to RMPs from its downstream neighbors only. We do not consider the number of duplicate packets sent by upstream neighbors vs. the number of duplicate packets sent by downstream neighbors because they are not important in the reduction of RMPs.

4.2.9 Simulation Results Demonstrating the Effect of a Node Selectively Responding to RMPs in OSDRMP(rRMP) Protocol

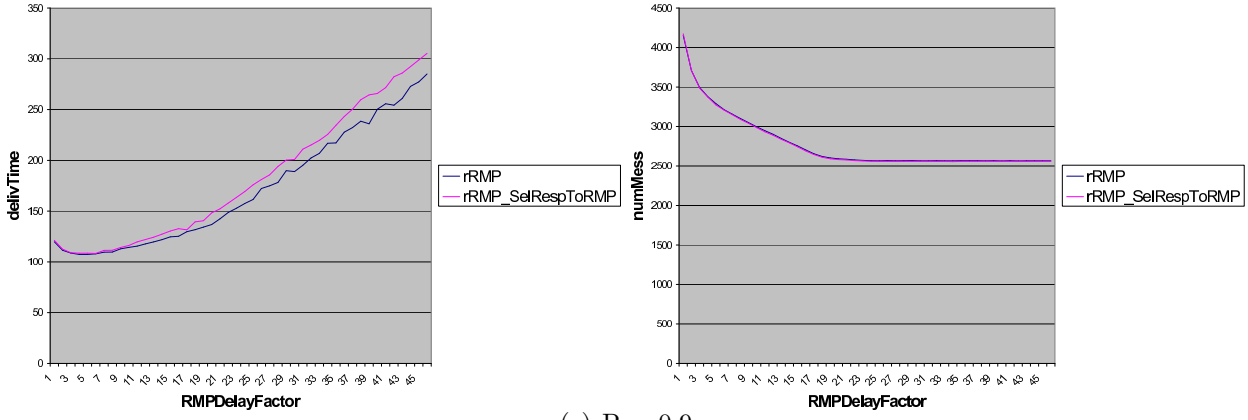
Based on the results in the previous section, we modify the OSDRMP(rRMP) protocol such that a node x responds only to RMPs sent to it by its downstream neighbors. We call this modified protocol OSDRMP(rRMP) with SelResp; in the graph, it is denoted by rRMP_SelRespToRMP. We run simulations with the source-destination pairs (i)$s = 40$, $h = 5$ and (ii)$s = 0$, $h = 14$ at 3 different probabilities of successful transmission ($P = 0.3$, 0.6 and 0.9) and $n = 10$ in the 12x9 grid network in Figure 4.1 and observe the performance of the modified protocol OSDRMP(rRMP) vs. OSDRMP(rRMP)with SelResp. The results are shown in the Figure 4.16 and Figure 4.17 .



(a) P = 0.3

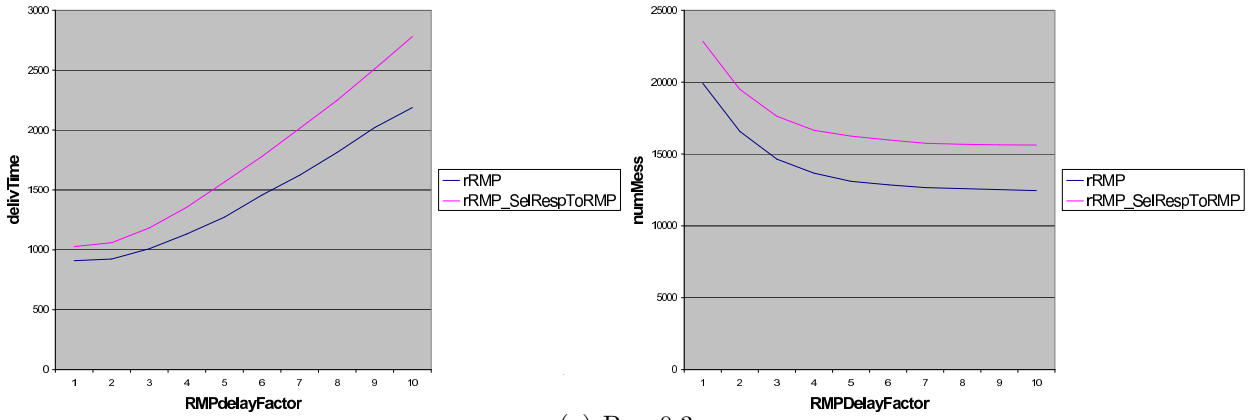


(b) P = 0.6

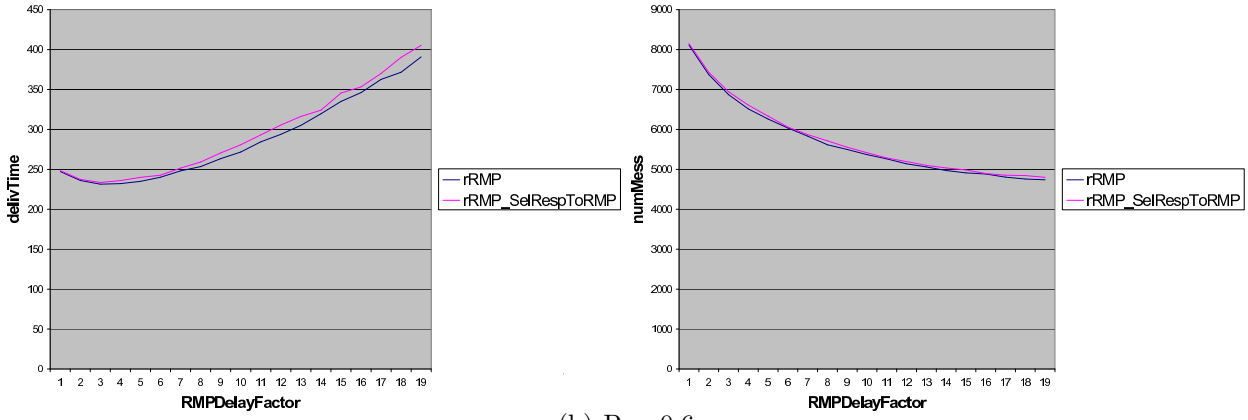


(c) P = 0.9

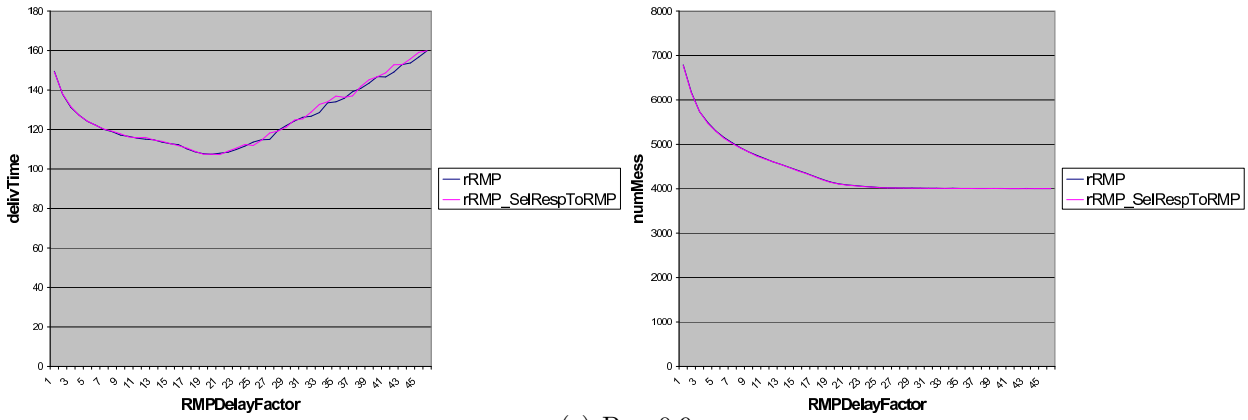
Figure 4.16: Performance of OSDRMP(rRMP) for two different methods of responding to RMPs by nodes and s=40, h=5, n=10.



(a) P = 0.3



(b) P = 0.6



(c) P = 0.9

Figure 4.17: Performance of OSDRMP(rRMP) for two different methods of responding to RMPs by nodes and s=0, h=14, n=10.

The results show that the OSDRMP(rRMP) protocol with SelResp performs worse in terms of delivTime and numMess with respect to the OSDRMP(rRMP) protocol. The difference is more pronounced when $P = 0.3$ and becomes reduced as P increases. The reason for this is as follows. As number of responses to RMPs are reduced in the OSDRMP(rRMP) protocol with SelResp, the probability of p_i reaching the neighbors of a node x also reduces. Selective response to RMP by x may prevent a packet p_i from reaching a downstream node y faster in comparison to non-selective response. In the OSDRMP(rRMP) protocol, p_i which reached y would have been sent from y to nodes further downstream and reached a destination node z faster in comparison to the OSDRMP(rRMP) protocol with SelResp. As mentioned before, as number of responses to RMPs are reduced in the OSDRMP(rRMP) protocol with SelResp, the probability of p_i reaching the neighbors of x also reduces which may lead to more RMPs. This possibility is more at lower P (=0.3) than at higher P (=0.6 and 0.9). Hence the difference in the performances of the OSDRMP(rRMP) protocol with SelResp vs. the OSDRMP(rRMP) protocol reduce at higher P.

Table 4.1: Time taken to fill DC for s=40, h=5, P=0.3, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 2 becomes full:	402.2	408.3	431.5	474.4	520.8
Average time when DC of 3 becomes full:	397.3	395.1	408.0	435.8	470.6
Average time when DC of 4 becomes full:	401.1	394.2	409.7	432.8	464.7
Average time when DC of 5 becomes full:	395.7	394.3	413.0	436.6	466.5
Average time when DC of 6 becomes full:	401.5	401.6	431.8	471.4	522.2
Average time when DC of 13 becomes full:	420.1	428.4	460.3	498.9	556.8
Average time when DC of 14 becomes full:	407.3	400.7	415.2	434.0	468.2
Average time when DC of 15 becomes full:	400.6	392.7	405.6	416.0	441.8
Average time when DC of 16 becomes full:	391.1	385.1	393.8	403.2	429.1
Average time when DC of 17 becomes full:	401.8	393.2	399.6	413.5	440.4
Average time when DC of 18 becomes full:	407.9	400.5	415.7	435.9	464.0
Average time when DC of 19 becomes full:	415.2	426.9	452.2	509.8	548.5
Average time when DC of 24 becomes full:	424.4	425.0	455.4	497.4	552.9
Average time when DC of 25 becomes full:	422.3	413.4	430.9	453.7	481.9
Average time when DC of 26 becomes full:	404.7	395.7	409.1	417.7	438.1
Average time when DC of 27 becomes full:	359.7	356.0	365.0	371.8	390.2
Average time when DC of 28 becomes full:	325.4	320.0	324.3	331.4	343.9
Average time when DC of 29 becomes full:	358.4	354.8	361.4	376.1	389.8
Average time when DC of 30 becomes full:	403.3	396.8	402.2	420.9	437.1
Average time when DC of 31 becomes full:	421.1	414.8	425.2	458.6	481.5
Average time when DC of 32 becomes full:	439.1	446.5	481.0	540.2	587.9
Average time when DC of 36 becomes full:	404.9	405.2	425.5	464.5	512.0
Average time when DC of 37 becomes full:	429.4	416.3	429.7	453.0	483.7
Average time when DC of 38 becomes full:	397.6	390.1	398.2	412.4	425.4
Average time when DC of 39 becomes full:	321.6	319.3	323.3	331.3	338.1
Average time when DC of 41 becomes full:	321.9	317.3	323.1	330.6	339.4
Average time when DC of 42 becomes full:	390.0	391.6	391.5	413.4	426.9
Average time when DC of 43 becomes full:	431.7	421.8	429.4	453.9	477.7
Average time when DC of 44 becomes full:	433.8	428.2	453.7	504.4	549.3
Average time when DC of 45 becomes full:	471.2	551.6	661.1	809.6	961.6
Average time when DC of 48 becomes full:	422.9	422.8	453.6	499.7	554.0
Average time when DC of 49 becomes full:	422.1	411.8	424.8	456.6	490.5
Average time when DC of 50 becomes full:	404.3	396.8	403.7	422.1	444.4
Average time when DC of 51 becomes full:	357.2	353.5	362.8	378.6	387.4
Average time when DC of 52 becomes full:	321.6	319.2	321.0	331.6	338.0
Average time when DC of 53 becomes full:	356.0	356.1	363.0	374.8	388.6
Average time when DC of 54 becomes full:	403.5	394.8	401.9	423.8	443.6
Average time when DC of 55 becomes full:	418.7	412.8	426.7	451.8	483.9
Average time when DC of 56 becomes full:	441.1	441.9	483.4	531.1	588.7
Average time when DC of 61 becomes full:	417.2	423.8	459.7	503.9	562.7
Average time when DC of 62 becomes full:	410.0	401.4	413.7	444.1	468.3
Average time when DC of 63 becomes full:	403.5	394.6	402.5	421.6	440.5
Average time when DC of 64 becomes full:	391.4	386.7	393.1	408.0	429.0
Average time when DC of 65 becomes full:	403.2	395.8	400.9	420.4	443.4
Average time when DC of 66 becomes full:	410.9	404.9	410.4	439.5	471.9
Average time when DC of 67 becomes full:	418.5	430.0	454.8	501.3	556.5
Average time when DC of 74 becomes full:	417.1	423.9	457.9	503.4	558.6
Average time when DC of 75 becomes full:	418.8	409.3	423.4	452.4	486.5
Average time when DC of 76 becomes full:	434.0	419.0	427.6	453.9	480.6
Average time when DC of 77 becomes full:	417.3	413.3	426.8	454.4	489.8

Table 4.2: Time taken to fill DC for s=40, h=5, P=0.6, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 2 becomes full:	126.0	120.0	116.3	115.2	113.1
Average time when DC of 3 becomes full:	123.1	113.7	110.0	106.8	104.5
Average time when DC of 4 becomes full:	123.9	114.0	111.0	107.8	105.2
Average time when DC of 5 becomes full:	122.5	112.9	109.9	107.1	104.8
Average time when DC of 6 becomes full:	127.1	117.9	115.8	116.0	113.5
Average time when DC of 13 becomes full:	130.5	124.0	120.5	119.6	118.7
Average time when DC of 14 becomes full:	123.7	115.2	110.0	108.6	104.8
Average time when DC of 15 becomes full:	119.2	111.2	106.6	103.8	100.5
Average time when DC of 16 becomes full:	117.0	108.9	104.6	101.2	98.8
Average time when DC of 17 becomes full:	119.3	109.9	106.5	103.1	101.1
Average time when DC of 18 becomes full:	124.0	114.5	110.5	108.5	104.9
Average time when DC of 19 becomes full:	132.2	122.6	120.1	118.8	120.3
Average time when DC of 24 becomes full:	131.5	125.3	119.9	119.8	118.2
Average time when DC of 25 becomes full:	126.2	119.1	113.0	110.5	108.3
Average time when DC of 26 becomes full:	119.7	111.9	108.0	104.8	101.6
Average time when DC of 27 becomes full:	107.7	100.9	98.7	95.0	93.8
Average time when DC of 28 becomes full:	100.7	94.7	91.3	88.1	87.2
Average time when DC of 29 becomes full:	108.2	101.7	97.9	94.8	93.5
Average time when DC of 30 becomes full:	119.7	112.1	107.2	103.8	102.4
Average time when DC of 31 becomes full:	128.5	118.6	114.1	110.5	109.5
Average time when DC of 32 becomes full:	137.3	128.5	126.0	123.9	125.4
Average time when DC of 36 becomes full:	120.2	113.3	107.9	106.9	104.1
Average time when DC of 37 becomes full:	126.9	118.6	113.6	110.6	107.8
Average time when DC of 38 becomes full:	117.2	109.2	105.0	101.6	100.8
Average time when DC of 39 becomes full:	99.4	92.9	90.6	87.8	86.4
Average time when DC of 41 becomes full:	100.0	93.6	90.6	87.9	86.1
Average time when DC of 42 becomes full:	116.5	109.2	105.9	101.8	99.9
Average time when DC of 43 becomes full:	128.7	119.4	114.6	110.5	109.7
Average time when DC of 44 becomes full:	129.4	119.1	114.0	111.5	112.1
Average time when DC of 45 becomes full:	138.8	140.8	144.5	157.4	174.7
Average time when DC of 48 becomes full:	131.4	124.8	118.9	118.5	119.0
Average time when DC of 49 becomes full:	126.7	118.5	112.5	109.6	108.7
Average time when DC of 50 becomes full:	118.6	111.8	107.1	104.2	102.5
Average time when DC of 51 becomes full:	107.8	101.0	97.9	95.2	92.9
Average time when DC of 52 becomes full:	100.0	92.8	90.3	87.5	86.8
Average time when DC of 53 becomes full:	107.0	100.7	97.6	95.4	93.4
Average time when DC of 54 becomes full:	118.7	111.4	107.9	103.5	102.6
Average time when DC of 55 becomes full:	127.6	118.4	113.1	110.2	108.5
Average time when DC of 56 becomes full:	137.3	128.7	125.1	124.2	123.9
Average time when DC of 61 becomes full:	131.8	123.7	121.1	118.7	120.0
Average time when DC of 62 becomes full:	125.4	116.3	111.1	107.8	106.9
Average time when DC of 63 becomes full:	119.6	111.5	106.6	103.8	102.2
Average time when DC of 64 becomes full:	117.2	108.9	104.9	100.7	100.8
Average time when DC of 65 becomes full:	119.9	111.8	107.8	103.4	102.6
Average time when DC of 66 becomes full:	124.7	115.6	111.5	107.1	107.3
Average time when DC of 67 becomes full:	130.7	123.2	121.1	118.3	119.1
Average time when DC of 74 becomes full:	131.0	123.6	119.8	119.4	119.9
Average time when DC of 75 becomes full:	128.3	118.3	112.8	110.0	107.7
Average time when DC of 76 becomes full:	129.1	118.9	115.0	110.4	109.3
Average time when DC of 77 becomes full:	128.1	119.1	114.1	109.1	109.3

Table 4.3: Time taken to fill DC for s=40, h=5, P=0.9, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 2 becomes full:	90.3	84.5	81.8	80.5	78.6
Average time when DC of 3 becomes full:	87.7	82.1	79.5	78.3	76.6
Average time when DC of 4 becomes full:	87.3	81.8	79.2	78.1	76.3
Average time when DC of 5 becomes full:	87.4	81.9	79.4	78.4	76.7
Average time when DC of 6 becomes full:	90.0	84.4	81.6	80.4	78.6
Average time when DC of 13 becomes full:	92.5	86.7	83.6	82.0	80.4
Average time when DC of 14 becomes full:	88.8	83.2	80.3	78.9	77.2
Average time when DC of 15 becomes full:	82.1	77.8	75.5	74.8	73.2
Average time when DC of 16 becomes full:	78.8	74.7	72.8	72.9	70.7
Average time when DC of 17 becomes full:	82.3	77.9	75.5	74.6	73.1
Average time when DC of 18 becomes full:	88.5	82.9	80.1	79.0	77.3
Average time when DC of 19 becomes full:	92.2	86.4	83.6	81.9	80.3
Average time when DC of 24 becomes full:	93.5	87.6	84.4	82.4	81.1
Average time when DC of 25 becomes full:	90.7	85.0	82.0	80.3	78.7
Average time when DC of 26 becomes full:	82.1	78.2	75.8	74.9	73.5
Average time when DC of 27 becomes full:	72.8	69.2	67.7	67.6	65.7
Average time when DC of 28 becomes full:	65.0	61.7	60.5	60.6	58.4
Average time when DC of 29 becomes full:	72.5	69.3	67.2	67.7	66.2
Average time when DC of 30 becomes full:	82.1	78.1	75.7	75.1	73.7
Average time when DC of 31 becomes full:	90.8	84.8	82.2	80.6	79.2
Average time when DC of 32 becomes full:	93.9	87.7	84.5	83.2	81.9
Average time when DC of 36 becomes full:	88.2	82.4	79.2	77.7	76.2
Average time when DC of 37 becomes full:	88.7	83.4	81.0	79.4	77.7
Average time when DC of 38 becomes full:	77.9	74.6	73.1	71.5	70.4
Average time when DC of 39 becomes full:	64.7	61.1	60.5	59.5	58.7
Average time when DC of 41 becomes full:	64.8	62.0	60.3	59.8	58.7
Average time when DC of 42 becomes full:	78.3	74.4	72.8	71.5	70.6
Average time when DC of 43 becomes full:	87.9	83.0	80.7	79.3	78.1
Average time when DC of 44 becomes full:	88.8	82.8	80.5	78.6	77.6
Average time when DC of 45 becomes full:	91.3	85.1	83.6	82.5	82.3
Average time when DC of 48 becomes full:	93.5	87.6	84.4	82.7	81.2
Average time when DC of 49 becomes full:	90.5	84.8	82.0	80.3	78.7
Average time when DC of 50 becomes full:	82.0	77.8	76.1	74.8	73.1
Average time when DC of 51 becomes full:	72.1	68.9	67.8	67.4	65.7
Average time when DC of 52 becomes full:	64.3	61.2	60.3	59.8	58.7
Average time when DC of 53 becomes full:	72.0	68.8	67.5	66.7	65.9
Average time when DC of 54 becomes full:	81.7	77.5	75.4	74.6	73.4
Average time when DC of 55 becomes full:	90.6	84.7	81.9	80.2	79.2
Average time when DC of 56 becomes full:	94.1	87.5	84.7	83.1	82.1
Average time when DC of 61 becomes full:	92.7	86.9	83.7	82.2	80.6
Average time when DC of 62 becomes full:	89.5	83.8	81.2	79.3	77.9
Average time when DC of 63 becomes full:	81.7	77.8	75.6	74.3	73.6
Average time when DC of 64 becomes full:	77.9	74.3	72.8	71.3	70.6
Average time when DC of 65 becomes full:	81.6	77.4	75.3	74.5	73.1
Average time when DC of 66 becomes full:	89.1	83.5	80.7	79.4	78.1
Average time when DC of 67 becomes full:	92.2	86.6	83.7	82.2	80.9
Average time when DC of 74 becomes full:	92.5	86.7	83.7	81.9	80.5
Average time when DC of 75 becomes full:	91.2	84.7	82.1	80.0	78.7
Average time when DC of 76 becomes full:	88.1	82.6	80.2	78.9	77.7
Average time when DC of 77 becomes full:	90.4	84.3	82.0	79.9	78.9

Table 4.4: Time taken to fill DC for s=0, h=14, P=0.3, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 1 becomes full:	288	294.5	321.7	349.4	386.6
Average time when DC of 2 becomes full:	370.6	385.8	414.4	456.4	503.4
Average time when DC of 3 becomes full:	412.4	421.5	452.8	505.7	552.4
Average time when DC of 4 becomes full:	432.8	441.4	472.8	522	576.6
Average time when DC of 5 becomes full:	451.5	459.1	478.6	529.2	589.2
Average time when DC of 6 becomes full:	472.7	478.3	492.4	533.9	597.3
Average time when DC of 7 becomes full:	490.4	494.5	508.1	538.7	611.4
Average time when DC of 8 becomes full:	497.8	497.9	501.7	541.1	601.5
Average time when DC of 9 becomes full:	475.2	476.5	487.9	519	586.3
Average time when DC of 10 becomes full:	356.9	398.6	441.1	486.6	520.2
Average time when DC of 11 becomes full:	217.6	310.6	355.6	404.6	446.9
Average time when DC of 12 becomes full:	284.9	295.8	313.4	351.8	382.7
Average time when DC of 13 becomes full:	331.4	338.8	361.8	397.9	437
Average time when DC of 14 becomes full:	379.5	388.2	410.3	455.2	491
Average time when DC of 15 becomes full:	413.5	416.2	439.1	487.1	536.9
Average time when DC of 16 becomes full:	433.3	437.2	458.7	509.6	557.9
Average time when DC of 17 becomes full:	454.2	456	472	517.9	577.2
Average time when DC of 18 becomes full:	474.9	474.6	488.4	530.3	583.8
Average time when DC of 19 becomes full:	494.4	497.6	506.3	538.9	593.5
Average time when DC of 20 becomes full:	512.7	507.6	516.7	545.4	606.6
Average time when DC of 21 becomes full:	515.5	495.1	520	551.2	599.1
Average time when DC of 22 becomes full:	422.9	463.5	487.4	526.6	581.5
Average time when DC of 23 becomes full:	277.9	368.3	407.7	470.5	497.6
Average time when DC of 24 becomes full:	372.2	381.3	412.1	455.5	506
Average time when DC of 25 becomes full:	378.6	384.7	408.8	455	500.1
Average time when DC of 26 becomes full:	394.6	400.8	425.4	464.9	509
Average time when DC of 27 becomes full:	420	421.2	443.8	480.4	532.8
Average time when DC of 28 becomes full:	441.2	438.2	458.9	494.3	545.3
Average time when DC of 29 becomes full:	456	455.9	473.3	511.2	562.2
Average time when DC of 30 becomes full:	476.8	476	487.9	524.9	577.2
Average time when DC of 31 becomes full:	498	494.4	501.1	530.2	588.7
Average time when DC of 32 becomes full:	514.6	509.8	509.5	542.6	601.8
Average time when DC of 33 becomes full:	528.9	517.6	521.3	555.5	607.9
Average time when DC of 34 becomes full:	491	506	518.3	546.2	607.9
Average time when DC of 35 becomes full:	335.6	411.4	447.8	491.4	553
Average time when DC of 36 becomes full:	415.6	419.6	454	493	557.6
Average time when DC of 37 becomes full:	411.4	416.1	443.5	484.9	538.2
Average time when DC of 38 becomes full:	418.2	419.9	443.5	482.4	528.8
Average time when DC of 39 becomes full:	431.7	432.5	447.1	492.4	538.3
Average time when DC of 40 becomes full:	448.9	444.2	464.5	494	548.5
Average time when DC of 41 becomes full:	463.5	459.6	479.5	509.9	564.9
Average time when DC of 42 becomes full:	480.6	476.5	490	520.6	573.6
Average time when DC of 43 becomes full:	499.4	497.2	502.7	526.5	580.5
Average time when DC of 44 becomes full:	520.8	515.7	512.1	536.7	592.5
Average time when DC of 45 becomes full:	536.3	528	526.7	549.5	603.5
Average time when DC of 46 becomes full:	548.3	535	535.1	572.9	620
Average time when DC of 47 becomes full:	573	566.6	582.3	631.3	697.8
Average time when DC of 48 becomes full:	438.2	434.3	473.1	517.9	581.4
Average time when DC of 49 becomes full:	435.1	435.5	465.3	504	559.5
Average time when DC of 50 becomes full:	437.2	435.2	461.4	494.9	547.2

Table 4.5: Time taken to fill DC for s=0, h=14, P=0.6, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 1 becomes full:	77.0	76.3	76.3	75.8	77.8
Average time when DC of 2 becomes full:	99.3	96.1	96.6	95.5	97.4
Average time when DC of 3 becomes full:	114.7	110.1	109.1	108.9	109.3
Average time when DC of 4 becomes full:	127.4	121.3	120.4	118.8	118.6
Average time when DC of 5 becomes full:	137.0	131.8	130.0	128.5	125.7
Average time when DC of 6 becomes full:	146.1	141.1	138.3	135.6	134.6
Average time when DC of 7 becomes full:	153.3	148.0	144.6	141.1	140.1
Average time when DC of 8 becomes full:	160.1	154.2	149.8	144.2	142.4
Average time when DC of 9 becomes full:	157.9	148.6	142.4	138.7	137.9
Average time when DC of 10 becomes full:	133.3	130.7	126.7	126.0	126.2
Average time when DC of 11 becomes full:	92.7	93.5	91.2	94.0	93.5
Average time when DC of 12 becomes full:	77.1	76.4	76.3	77.0	78.7
Average time when DC of 13 becomes full:	84.7	82.3	82.1	81.9	84.1
Average time when DC of 14 becomes full:	100.0	96.1	95.5	93.5	95.4
Average time when DC of 15 becomes full:	114.4	109.8	108.6	105.8	106.1
Average time when DC of 16 becomes full:	126.5	120.5	117.9	116.0	116.1
Average time when DC of 17 becomes full:	136.5	131.2	127.9	125.3	123.6
Average time when DC of 18 becomes full:	146.0	140.4	136.3	133.4	131.8
Average time when DC of 19 becomes full:	153.6	148.2	143.5	141.6	139.6
Average time when DC of 20 becomes full:	160.2	154.4	149.0	145.6	142.4
Average time when DC of 21 becomes full:	164.3	157.9	150.4	148.9	144.7
Average time when DC of 22 becomes full:	146.0	142.7	139.7	137.7	133.8
Average time when DC of 23 becomes full:	108.1	113.5	113.6	119.1	115.4
Average time when DC of 24 becomes full:	98.3	96.3	96.0	96.5	97.2
Average time when DC of 25 becomes full:	99.0	96.2	94.4	95.1	94.5
Average time when DC of 26 becomes full:	108.6	105.2	102.4	101.3	101.1
Average time when DC of 27 becomes full:	118.6	114.4	112.5	109.6	109.2
Average time when DC of 28 becomes full:	129.2	123.9	120.8	118.7	117.8
Average time when DC of 29 becomes full:	138.2	133.3	128.9	127.8	125.0
Average time when DC of 30 becomes full:	147.1	142.2	137.8	135.1	132.8
Average time when DC of 31 becomes full:	154.9	148.5	144.4	141.2	139.3
Average time when DC of 32 becomes full:	161.6	156.4	151.7	147.8	145.2
Average time when DC of 33 becomes full:	164.1	156.1	151.6	147.8	145.2
Average time when DC of 34 becomes full:	162.1	156.7	151.4	148.0	146.2
Average time when DC of 35 becomes full:	120.3	119.5	117.3	117.4	114.4
Average time when DC of 36 becomes full:	113.1	110.0	109.5	108.8	109.8
Average time when DC of 37 becomes full:	112.1	109.7	107.0	106.3	106.2
Average time when DC of 38 becomes full:	118.2	115.4	111.3	110.1	109.3
Average time when DC of 39 becomes full:	124.9	121.5	119.0	116.2	115.6
Average time when DC of 40 becomes full:	134.1	129.2	125.9	122.4	121.3
Average time when DC of 41 becomes full:	142.3	136.4	133.7	129.9	128.2
Average time when DC of 42 becomes full:	149.4	143.3	140.0	136.6	135.0
Average time when DC of 43 becomes full:	156.8	151.0	147.4	143.4	141.4
Average time when DC of 44 becomes full:	162.5	156.1	152.2	146.8	143.3
Average time when DC of 45 becomes full:	168.7	160.5	156.3	153.6	148.9
Average time when DC of 46 becomes full:	164.6	154.5	147.7	144.7	139.8
Average time when DC of 47 becomes full:	173.2	164.5	158.1	154.0	153.0
Average time when DC of 48 becomes full:	124.6	121.7	120.5	119.5	119.8
Average time when DC of 49 becomes full:	124.3	120.3	118.3	116.1	115.6
Average time when DC of 50 becomes full:	128.2	124.1	121.0	118.3	117.3

Table 4.6: Time taken to fill DC for s=0, h=14, P=0.9, n=10.

	RMPDelayFactor				
	1.0	2.0	3.0	4.0	5.0
Average time when DC of 1 becomes full:	38.6	37.4	37.3	36.2	37.0
Average time when DC of 2 becomes full:	52.2	51.0	50.3	49.9	49.5
Average time when DC of 3 becomes full:	61.6	60.0	58.4	57.7	57.9
Average time when DC of 4 becomes full:	69.8	67.2	65.6	64.6	64.6
Average time when DC of 5 becomes full:	77.1	73.7	71.9	70.4	70.0
Average time when DC of 6 becomes full:	83.7	79.6	77.7	75.7	75.0
Average time when DC of 7 becomes full:	89.6	85.2	82.2	80.2	79.2
Average time when DC of 8 becomes full:	95.2	90.5	87.1	84.5	83.1
Average time when DC of 9 becomes full:	99.9	94.4	90.7	88.2	86.2
Average time when DC of 10 becomes full:	104.1	97.5	93.4	90.8	88.8
Average time when DC of 11 becomes full:	102.6	94.1	90.2	88.6	87.0
Average time when DC of 12 becomes full:	37.9	36.9	36.6	36.8	36.2
Average time when DC of 13 becomes full:	42.8	41.6	40.6	40.3	40.1
Average time when DC of 14 becomes full:	55.4	54.1	52.9	52.0	51.5
Average time when DC of 15 becomes full:	64.8	63.1	61.3	60.5	60.0
Average time when DC of 16 becomes full:	72.8	70.3	68.3	66.9	66.6
Average time when DC of 17 becomes full:	80.1	76.4	74.5	72.6	72.0
Average time when DC of 18 becomes full:	86.3	82.2	79.8	77.8	76.7
Average time when DC of 19 becomes full:	92.4	87.7	84.4	82.2	80.9
Average time when DC of 20 becomes full:	97.7	92.4	89.0	86.1	84.4
Average time when DC of 21 becomes full:	103.2	97.4	93.1	90.1	88.0
Average time when DC of 22 becomes full:	106.0	99.4	95.3	92.5	89.7
Average time when DC of 23 becomes full:	107.1	99.8	95.4	92.7	90.4
Average time when DC of 24 becomes full:	52.1	50.7	50.2	50.1	49.4
Average time when DC of 25 becomes full:	55.5	53.8	53.0	52.5	51.5
Average time when DC of 26 becomes full:	66.5	64.5	62.8	61.9	61.4
Average time when DC of 27 becomes full:	74.2	71.9	70.0	68.9	68.0
Average time when DC of 28 becomes full:	81.0	77.8	75.5	73.8	73.1
Average time when DC of 29 becomes full:	87.1	83.4	80.6	78.7	77.5
Average time when DC of 30 becomes full:	92.5	88.5	85.1	82.7	81.3
Average time when DC of 31 becomes full:	98.0	93.0	89.4	86.8	85.0
Average time when DC of 32 becomes full:	103.0	97.4	93.3	90.2	88.3
Average time when DC of 33 becomes full:	107.5	101.1	96.6	93.5	91.1
Average time when DC of 34 becomes full:	112.1	105.4	100.0	96.3	93.7
Average time when DC of 35 becomes full:	105.2	97.7	92.0	91.2	88.0
Average time when DC of 36 becomes full:	61.6	60.2	58.8	58.5	57.2
Average time when DC of 37 becomes full:	65.0	63.1	61.7	61.0	59.4
Average time when DC of 38 becomes full:	74.2	71.9	69.9	68.9	68.0
Average time when DC of 39 becomes full:	81.6	79.1	76.8	74.9	74.1
Average time when DC of 40 becomes full:	87.4	84.4	81.5	79.6	78.6
Average time when DC of 41 becomes full:	92.7	89.1	85.7	83.4	82.1
Average time when DC of 42 becomes full:	97.7	93.3	89.5	87.1	85.3
Average time when DC of 43 becomes full:	102.6	97.5	93.1	90.4	88.3
Average time when DC of 44 becomes full:	107.2	101.1	96.6	93.6	91.3
Average time when DC of 45 becomes full:	112.0	105.2	100.2	96.7	94.0
Average time when DC of 46 becomes full:	113.3	104.9	99.4	96.4	93.1
Average time when DC of 47 becomes full:	113.7	106.3	100.6	96.9	94.1
Average time when DC of 48 becomes full:	69.7	67.6	65.6	65.4	63.7
Average time when DC of 49 becomes full:	72.7	70.3	68.5	67.5	66.1
Average time when DC of 50 becomes full:	80.6	78.0	75.9	74.5	72.8

Table 4.7: Number of packets received by each node from upstream and downstream neighbors for s=40, h=5, P=0.3 and 0.6, n=100.

	P = 0.3				P = 0.6			
	RMPDelayFactor = 1		RMPDelayFactor = 3		RMPDelayFactor = 1		RMPDelayFactor = 3	
Nodes	packetTTL ≥ nodeTTL	packetTTL < nodeTTL	packetTTL ≥ nodeTTL	packetTTL < nodeTTL	packetTTL ≥ nodeTTL	packetTTL < nodeTTL	packetTTL ≥ nodeTTL	packetTTL < nodeTTL
2	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
3	90.6	9.4	90.3	9.7	82.6	17.4	82.3	17.7
4	63.4	36.6	62.3	37.7	47.3	52.7	47.5	52.2
5	90.8	9.2	90.8	9.2	83.3	16.7	83.5	16.5
6	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
13	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
14	84.5	15.5	84.3	15.7	69.9	30.1	70.1	29.9
15	81.9	18.1	81.9	18.1	74.9	25.1	75.5	24.5
16	62.3	37.7	62.3	37.7	56.4	43.6	55.9	44.1
17	82.0	18.0	82.8	17.2	74.8	25.2	74.1	25.9
18	84.4	15.6	84.2	15.8	69.6	30.4	70.7	29.3
19	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
24	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
25	82.5	17.5	83.0	17.0	67.0	33.0	68.2	31.8
26	82.1	17.9	81.5	18.5	75.0	25.0	75.2	24.8
27	86.9	13.1	86.7	13.3	84.3	15.7	83.8	16.2
28	67.9	32.1	68.5	31.5	65.8	34.2	65.7	34.3
29	87.4	12.6	87.0	13.0	84.7	15.3	83.2	16.8
30	82.1	17.9	82.3	17.7	75.5	24.5	74.8	25.2
31	82.5	17.5	83.2	16.8	66.1	33.9	68.3	31.7
32	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
36	63.7	36.3	62.7	37,3	39.7	60.3	41.1	58.9
37	55.4	44.6	54.0	46.0	45.8	54.2	46.4	53.6
38	65.9	34.1	65.5	34.5	61.1	38.9	59.6	40.4
39	69.7	30.3	67.8	32.2	65.5	34.5	66.3	33.7
41	67.9	32.1	68.6	31.4	65.4	34.5	65.2	34.8
42	66.5	33.5	65.3	34.7	61.4	38.6	59.5	40.5
43	54.8	45.2	54.4	45.6	45.6	54.4	44.5	55.5
44	59.6	40.4	60.4	39.6	38.9	61.1	40.3	59.7
45	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
48	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
49	82.9	17.1	83.2	16.8	67.6	32.4	68.7	31.3
50	82.1	17.9	82.3	17.7	75.3	24.7	76.3	23.7
51	87.3	12.7	87.5	12.5	85.2	14.8	84.2	15.8
52	68.3	31.7	68.6	31.4	66.2	33.8	65.4	34.6
53	87.5	12.5	87.6	12.4	84.5	15.5	84.0	16.0
54	82.5	17.5	82.0	18.0	77.4	22.6	76.7	23.3
55	83.0	17.0	83.2	16.8	67.0	33.0	67.8	32.2
56	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
61	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
62	85.6	14.4	84.9	15.1	72.6	27.4	72.8	27.2
63	82.0	18.0	82.0	18.0	76.8	23.2	76.0	24.0
64	65.3	34.7	65.7	34.3	62.4	37.6	60.0	40.0
65	82.0	18.0	82.0	18.0	77.2	32.8	76.2	23.8
66	84.7	15.3	84.7	15.3	74.5	22.5	74.2	25.8
67	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
74	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
75	82.9	17.1	82.6	17.4	67.4	32.6	66.8	33.2
76	54.3	45.7	55.0	45.0	45.6	54.4	45.3	54.7
77	83.2	16.8	82.7	17.3	65.9	34.1	67.7	32.3
78	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
87	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
88	60.1	39.9	60.7	39.3	37.6	62.4	39.4	60.6
89	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0
100	100.0	0.0	100.0	0.0	100.0	0.0	100.0	0.0

Table 4.8: Number of packets received by each node from upstream and downstream neighbors for s=40, h=5, P=0.9, n=100.

	P = 0.9			
	RMPDelayFactor = 1		RMPDelayFactor = 3	
Nodes	packetTTL ≥ nodeTTL	packetTTL < nodeTTL	packetTTL ≥ nodeTTL	packetTTL < nodeTTL
2	100.0	0.0	100.0	0.0
3	90.7	9.3	90.5	9.5
4	57.3	42.7	54.4	45.6
5	90.2	9.8	90.9	9.1
6	100.0	0.0	100.0	0.0
13	100.0	0.0	100.0	0.0
14	61.6	38.4	63.6	36.4
15	91.4	8.6	86.9	13.1
16	79.7	20.3	74.3	25.7
17	90.0	10.0	88.3	11.7
18	61.7	38.3	66.0	34.0
19	100.0	0.0	100.0	0.0
24	100.0	0.0	100.0	0.0
25	55.5	44.5	59.2	40.8
26	89.6	10.4	84.3	15.7
27	97.9	2.1	97.9	2.1
28	91.1	8.9	90.6	9.4
29	98.1	1.9	96.9	3.1
30	90.0	10.0	85.6	14.4
31	53.5	46.5	56.1	33.9
32	100.0	0.0	100.0	0.0
36	30.7	69.3	35.4	64.6
37	59.4	40.6	53.9	46.1
38	84.9	15.1	80.3	19.7
39	90.6	9.4	90.8	9.2
41	90.4	9.6	90.9	9.1
42	85.6	44.4	81.7	18.3
43	62.3	37.7	52.9	47.1
44	18.0	82.0	26.6	73.4
45	100.0	0.0	100.0	0.0
48	100.0	0.0	100.0	0.0
49	61.4	38.6	65.3	34.7
50	92.4	7.6	89.2	10.8
51	98.3	1.7	97.7	2.3
52	90.8	9.2	90.9	9.1
53	98.4	1.6	97.6	2.4
54	93.9	6.1	89.8	10.2
55	61.9	38.1	63.9	36.1
56	100.0	0.0	100.0	0.0
61	100.0	0.0	100.0	0.0
62	80.8	19.2	79.4	20.6
63	93.6	6.4	91.1	18.9
64	85.2	14.8	65.79.4	20.6
65	93.4	6.6	90.6	9.4
66	83.7	16.3	81.0	19.0
67	100.0	0.0	100.0	0.0
74	100.0	0.0	100.0	0.0
75	63.6	36.4	65.2	34.8
76	60.0	40.0	55.4	44.6
77	60.6	39.4	65.7	34.6
78	100.0	0.0	100.0	0.0
87	100.0	0.0	100.0	0.0
88	16.8	83.2	25.0	75.0
89	100.0	0.0	100.0	0.0
100	100.0	0.0	100.0	0.0

Chapter 5
Conclusions

In this dissertation research, we have explored different protocols for implementing reliable data transfer in wireless sensor networks (WSNs) and introduced a new protocol for implementing reliability during data transfer from a base station (sink) to sensor nodes for time-critical applications with zero-tolerance for data loss in wireless sensor networks.

In chapter 2, we reviewed a number of protocols implementing reliability by different methods in different application scenarios in wireless sensor networks. The methods included congestion control, recovery of data lost during transmission, etc. Data transfer in different application scenarios varied from bulk transfer of data in bursts from sensor nodes to a base station, to continuous transfer of data from sensor nodes to a base station, to data aggregation and filtering at intermediate nodes also during data transfer from sensor nodes to a base station, etc.

However, few of them discussed reliable data transfer from a base station to sensor nodes for time-critical applications with zero tolerance for data loss in energy-constrained wireless sensor networks or were suitable for it. Two protocols which considered implementing reliability in this application scenario were PSFQ and GARUDA. PSFQ uses an in-sequence forwarding of data packets with message priority as $RMP > RT > T$ which tends to delay delivery of the data packets to a node and GARUDA uses a cluster based method which has an overhead of cluster formation and this can delay the delivery of data packets to a node considerably in contrast to a method which does not use clusters as shown in chapter 2. It also proves costly in terms of number of messages. Such delays in delivery time are not suitable for time-critical applications.

We presented a new protocol OSDRMP for recovery of lost data during data transfer from a base station to sensor nodes for time-critical applications with zero tolerance for data loss in wireless sensor networks. The protocol is based on the following:

- Non-acknowledgement of packets at receiving nodes.
- Hop-by-hop detection and recovery of lost packets.
- Out-of-sequence forwarding of packets at nodes.

- A priority order for sending different types of messages at a node.
- Delayed request for missing packets at nodes.

There are three types of messages exchanged during the process of data recovery: Transmission (T), ReTransmission (RT), Request for Missing Packets(RMP). We analyzed and established the advantages of non-acknowledgement based transmission, hop-by-hop detection and recovery of lost packets, and out-of-sequence forwarding of packets over acknowledgement based transmission, end-to-end detection and recovery of lost packets, and in-sequence forwarding respectively. We also justified the requirement for delay in requesting packets missing at nodes. The superiority of the OSDRMP protocol was demonstrated against a well-known protocol PSFQ in terms of both the delivery time of all packets (from a base station to sensor nodes) and the number of packets exchanged in the process, via extensive simulations. We also investigated the time taken to fill the DC of a node in a network and concluded that the time had dependencies on the number of shortest paths between the source and the node, the number of upstream neighbors of the node and the node's degree.

In non-acknowledgement based methods, at least one packet needs to reach a node in order for the node to request missing packets. Though we assume that the data set sent by the base station will consist of enough packets for at least one packet to reach a node, we present a method that ensures at least one packet delivery to a node at the cost of some extra energy in terms of messages. In this method, a node sending one or more packets to its neighbors will wait for an acknowledgement for only one packet or a message from each of its neighbors indicating that the neighbor has a packet. If it does not receive this, it will retransmit packets repeatedly till it receives an acknowledgement or a message from each neighbor thus ensuring that each neighbor has received at least one packet.

A new method for generating reports for a base station on request while minimizing the number of messages has been suggested by us. Base stations in wireless sensor networks do not have an idea of the exact number of destination sensor nodes at a particular distance. Also, knowledge of intermediate nodes status cannot help base station in knowing for sure whether the entire data has reached the destination nodes or not. We use this knowledge in our reporting method where the base station needs to know that at least one destination

node has received the entire data in order to be satisfied. This prevents the base station from waiting for an unnecessary time period for reports from other destination nodes since it does not know the number of destination nodes. Intermediate node status is not required. This reduces size of messages and minimizes energy consumption in wireless sensor networks.

We further investigated different ways of improving the performance of the OSDRMP protocol:

- The effective degree of a node x is the number of neighbors which send more than a certain number of packets to x's data cache. The delay in sending in RMPs in OSDRMP protocol is dependent on the degree of x in order to give a chance to all neighbors of x to send the missing packets to it. However, if the delay is not dependent on those nodes which contribute very less to the number of packets in x's data cache, we can reduce unnecessary delay and hence, delivery time. Thus, the effective degree of a node is used to determine the delay in RMPs at each node. Our simulations showed that this reduced the delivery time but increased the number of messages for the OSDRMP protocol due to increase in number of RMPs.
- The priority order at a node x determines the order in which x sends messages when more than one message type is present at x at a time unit. While giving priority to RMPs enables x to receive missing packets faster, giving priority to Ts enables downstream neighbors of x to receive packets faster by ensuring that they have all the packets sent by the base station even when x and other upstream neighbors do not have the complete set. From chapter 3, we see that the priority order $T > RT > RMP$ is more suitable for out-of-sequence forwarding in comparison to $RMP > RT > T$. However, we wanted to see whether giving priority to RMPs at destination nodes or determining the priority order at each node probabilistically with nodes nearer to the destination having more probability of putting priority on RMPs will enable destination nodes to request and receive packets faster in comparison to the case where all nodes have the priority order as $T > RT > RMP$. In our simulation results, the variations did show any significant improvement over the original OSDRMP protocol with $T > RT > RMP$ priority order at all nodes. This is because the advantage gained

by putting priority on RMPs was negated by its effect on out-of-sequence forwarding as discussed in Chapter 3.

- We performed simulations in order to determine the direction from which a node received most Ts. Our results showed that upstream neighbors of a node contributed maximum to the number of packets received at a node. So we concluded that upstream neighbors of a node x are enough to satisfy its request for missing packets and we can use this property to reduce the number of messages. Hence, we modified the OSDRMP protocol so that every node responds to RMPs from its downstream neighbors only. In our simulations, this selective response to requests for data packets did not however show any improvements in the performance of the OSDRMP protocol in terms of both the delivery time and the number of messages. The reason is that lesser number of responses to RMPs at a node lowered the chance of a node getting packets and increased the number of RMPs.

Our studies show that while we have presented a reliable and efficient data transfer protocol, We also want to explore the following in our future work.

- Methods for further reduction in the number of messages like efficient methods for calculation of the delay in requesting missing packets and determination of optimal message priority orders at nodes.
- Application of the OSDRMP protocol to implement reliability for other application scenarios in wireless sensor networks and also for applications in other wireless networks.

Bibliography

1. A.Mainwaring, J.Polastre, R.Zewczyk, D.Culler, J.Andderson, "Wireless Sensor Networks for Habitat Monitoring." *ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02)*, Atlanta, GA, September 2002.

2. L.Schwiebert, S. Gupta, J.Weinmann, "'Research Challenges in Wireless Sensors of Biomedical Sensors." *In Mobile Computing and Networking*, pp.151, 165, 2001.

3. Wei Ye, J.Heidemann and Deborah Estrin, "An Energy-effcient MAC Protocol for Wireless Sensor Networks." *Proc. IEEE INFOCOM*, New York, NY, June 2002.

4. S.Singh and C.S.Raghevendra, "PAMAS: Power Aware Multi-access Protocol with Signaling for Ad-hoc Networks." *ACM Computer Communication Review*, Vol.28, No.3, pp.526, July 1998.

5. K.Sohrabi, J. Gao, V. Ailawadhi and G. Pottie, "Protocols for Self-organization of a Wireless Sensor Network." *IEEE Personal Comm. Magazine*, Vol.7, No.5, pp.16-27, October 2000.

6. C. Intanagonwiwat, Ramesh Govindan and Deborah Estrin, "Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks." *In Proceedings of the Sixth Annual International Conference on Mobile Computing and Networks (MobiCOM 2000)*, Boston, Massachusetts, August 2000.

7. S. Heddetiniemi and A. Leistman, "A Survey of Gossiping and Broadcasting in Communication Networks." *Networks*, Vol.18, 1988.

8. W.R. Heinzelmann, J. Kulik and H. Balakrishnan,"Adaptive Protocols for Information Dissemination in Wireless Sensor Networks." *Proc. ACM MobiCOM '99*, pp.174-185, Seattle, WA, 1999.

9. K. Sohrabi, "'Protocols for Self-Organization in a Wireless Sensor Network." *IEEE Personal Comm.* pp.16-27, October 2000.

10. G.Tolle, J. Polastre, R. Szewczyk, N. Turner, K. Tu, P. Buonadonna, S. Burgess, D. Gay, W. Hong, T. Dawson and D. Culler, "A Macroscope in the Redwoods." *Proceedings of the Third ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2-4, 2005.

11. Péter Völgyesi, András Nádás, Ákos Lédeczi and Károly Molnár, "Reliable Multihop Bulk Transfer Service for Wireless Sensor Networks." *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems(EBCS'06)*, 2006.

12. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. "System architecture directions for network sensors." *Proc. of the 9th International Conf. on Architectural Support for Programming Languages and Operating Systems*, pp. 93-104, November 2000.

13. Nurcan Tezcan, Wenye Wang and Mo-Yuen Chow, "A Bidirectional Reliable Transport Mechanism for Wireless Sensor Networks." *Military Communications Conference (MILCOM)*, Octber 2005.

14. C. Wan, A. Campbell, L. Krishnahmurthy. "PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks." *ACM International Workshop on Wireless Sensor Networks and Applications*, Atlanta, Georgia, September 2002.

15. S.J. Park, R. Vedantham, R. Sivakumar, and I.F. Akyildiz, "A Scalable Approach for Reliable Downstream Data Delivery in Wireless Sensor Networks." *Proc. Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, Tokyo, Japan, May 2004.

16. Hari Balakrishnan, Srinivasan Seshan, Elan Amir and Randy H. Katz, "Improving TCP/IP Performance over Wireless Networks." *Proc. First ACM Conference on Mobile Computing and Networking*, November, 95.

17. Antonio DeSimone, Mooi Choo Chuah and On-Ching Yue, "Throughput Performance of Transport-Layer Protocols over Wireless LANs." *Proceedings of IEEE GLOBECOM, pages 542–549*, December 1993.

18. F. Stann and J. Heidemann, "RMST: Reliable Data Transport in Sensor Networks." *Proc.* 1*supst IEEE Intl. Workshop on Sensor Network Protocols and Appl.*, Alaska, May 2003.

19. Y. Sankarsubramaniam, O. Akan, I. Akyildiz, "ESRT: Event-to-sink Reliable Transport in Wireless Sensor Networks." *Proc. ACM Mobihoc 2003*, Maryland, June 2003.

20. C-Y Wan, S. B. Eisenman and A. T. Campbell, "CODA: Congestion Detection and Avoidance in Sensor Networks." *Proc. First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003), pp.266-279*, Los Angeles, CA, Nov 2003.

21. C.Wang, K. Sohraby, B. Li, "SenTCP: A Hop-by-Hop Congestion Control Protocol for Wireless Sensor Networks." *IEEE INFOCOM 2005 (Poster Paper)*, March 2005.

22. W. Ye and J. Heidemann, "Medium Access Control in Wireless Sensor Networks." *USC/ISI Technical Report ISI-TR-580*, October 2003.

23. LAN MAN Standards Committee of the IEEE Computer Society, "Wireless LAN medium access control (MAC) and physical layer (PHY) specification." *IEEE Std 802.11-1999 edition*, IEEE, New York, NY, USA, 1999.

24. Phil Karn, "MACA: A New Channel Access Method for Packet Radio." *In Proceedings of the 9th ARRL Computer Networking Conference*, London, Ontario, Canada, Sept. 1990, pp. 134140.

25. V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A Media Access Protocol for Wireless LANs." *In Proceedings of the ACM SIGCOMM Conference*, London, UK, Sept. 1994, pp. 212225.

26. Katayoun Sohrabi and Gregory J. Pottie, "Performance of a Novel Self-organization Protocol for Wireless Ad hoc Sensor Networks." *In Proceedings of the IEEE 50th Vehicular Technology Conference*, pp. 12221226, 1999.

27. Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan, "Energy-efficient Communication Protocols for Wireless Microsensor Networks." *In Proceedings of the Hawaii International Conference on Systems Sciences*, Jan. 2000.

28. Bluetooth SIG Inc., "Specification of the Bluetooth system: Core." *http://www.bluetooth.org/*, 2001.

29. Jaap C. Haartsen, "The Bluetooth radio system." *IEEE Personal Communications Magazine*, pp. 2836, Feb. 2000.

30. 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.

31. 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.

32. K. Jamieson, H. Balakrishnan, and Y. C. Tay, "Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks." *MIT Laboratory for Computer Science*, Tech. Rep. 894, May 2003.

33. G. Lu, B. Krishnamachari, C.S. Raghavendra, "An adaptive energyefficient and low-latency MAC for data gathering in wireless sensor networks." *Proceedings of 18th International Parallel and Distributed Processing Symposium*, Pages: 224, 26-30 April 2004.

34. B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating Congestion in Wireless Sensor Networks." *In SenSys 04*, Baltimore, Maryland, 2004.

35. I.F.Akyildiz, W.Su, Y. Snakarasubramaniam and E. Cayirci, "A Survey on Sensor Networks." *IEEE Communications Magazine*, August, 2002.

36. G.J.Pottie and W.J.Kaiser, "Wireless Integrated Network Sensors." *ACM Communications*, Vol.43, No.5, pp 551-558, May, 2000.

37. E. Shih et al. "Physical Layer Drive Protocol and Algorithm Design for Enegy-Efficient Wireless Sensor Networks." Proceedings of ACM Mobicom '01, pp272-286, Italy, July, 2001.

38. http://computer.howstuffworks.com/mote4.htm.

39. J.N. Al-Karaki, A.E. Kamal, "Routing Techniques in Wireless Sensor Networks: A Survey." *IEEE Wireless Communications*, Volume 11, Issue 6, Page(s): 6 - 28, Dec. 2004.

40. D. Braginsky and D. Estrin, "Rumor Routing Algorithm for Sensor Networks." *Proceedings of the First Workshop on Sensor Networks and Applications (WSNA)*, Atlanta, GA, October 2002.

41. F. Ye, A. Chen, S. Liu, L. Zhang, "A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks." *Proceedings of the 10th International Conference on Computer Communications and Networks (ICCCN)*, pp. 304-309, 2001.

42. C. Schurgers and M.B. Srivastava, "Energy Efficient Routing in Wireless Sensor Networks." *MILCOM Proceedings on Communications for Network-Centric Operations: Creating the Information Force*, McLean, VA, 2001.

43. M. Chu, H. Haussecker, and F. Zhao, "Scalable Information-Driven Sensor Querying and Routing for Ad hoc Heterogeneous Sensor Networks." *The International Journal of High Performance Computing Applications*, Vol. 16, No. 3, August 2002.

44. Y. Yao and J. Gehrke, "The Cougar Approach to In-network Query Processing in Sensor Networks." *SIGMOD Record*, September 2002.

45. N. Sadagopan et al., "The ACQUIRE mechanism for Efficient Querying in Sensor Networks." *Proceedings of the First International Workshop on Sensor Network Protocol and Applications*, Anchorage, Alaska, May 2003.

46. R. C. Shah and J. Rabaey, "Energy Aware Routing for Low Energy Ad Hoc Sensor Networks." *IEEE Wireless Communications and Networking Conference (WCNC)*, March 17-21, 2002, Orlando, FL.

47. S. Lindsey, C. Raghavendra, "PEGASIS: Power-Efficient Gathering in Sensor Information Systems." *IEEE Aerospace Conference Proceedings*, Vol. 3, 9-16 pp. 1125-1130, 2002.

48. A. Manjeshwar and D. P. Agarwal, "APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks," *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS*, pp. 195-202, 2002.

49. L. Subramanian and R. H. Katz, "An Architecture for Building Self Configurable Systems." *Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing*, Boston, MA, August 2000.

50. Q. Fang, F. Zhao, and L. Guibas, "Lightweight Sensing and Communication Protocols for Target Enumeration and Aggregation", *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing (MOBIHOC)*, pp. 165-176, 2003.

51. Jamal N. Al-Karaki, Raza Ul-Mustafa, Ahmed E. Kamal, "Data Aggregation in Wireless Sensor Networks - Exact and Approximate Algorithms.", *Proceedings of IEEE Workshop on High Performance Switching and Routing (HPSR)*, Phoenix, Arizona, April 18-21, 2004.

52. Q. Li and J. Aslam and D. Rus, "Hierarchical Power-aware Routing in Sensor Networks." *In Proceedings of the DIMACS Workshop on Pervasive Networking*, May, 2001.

53. F. Ye, H. Luo, J. Cheng, S. Lu, L. Zhang, "A Two-tier Data Dissemination Model for Large-Scale Wireless Sensor Networks", *Proceedings of ACM/IEEE MOBICOM*, 2002.

54. 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.

55. R. Patra and S. Nedevschi. "DTNLite: A Reliable Data Transfer Architecture for Sensor Networks." *Technical report, 2003*, CS294–1 Course Project Report, Berkeley.

56. A. Boukerche, R.W.N. Pazzi and R. B. Araujo, "A Fast and Reliable Protocol for Wireless Sensor Networks in Critical Conditions Monitoring Applications." *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, Pages: 157 - 164, Venice, Italy, 2004.

57. N. Tezcan, E. Cayirci, M.U. Caglayan, " End-to-end Reliable Event Transfer in Wireless Sensor Networks." *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Vol. 2, Pages: 989- 994 Vol.2, Sept, 2004.

58. N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, "A Wireless Sensor Network for Structural Monitoring." *Proceedings of the 2nd international conference on Embedded networked sensor systems*, Pages 13-24, Baltimore, Maryland, 2004.

59. Y.G. Iyer, S. Gandham and S. Venkatesan, " STCP: A Generic Transport Layer Protocol for Wireless Sensor Networks." *Proceedings of 14th International Conference on Computer Communications and Networks*, Pages 449-454, October 17-19, 2005.

Appendix: Permission Letters

From: j.hansson@ieee.org
To: ddatta1@lsu.edu
CC:
Subject: Yur Permission Grant
Date: Wednesday, June 13, 2007 2:48:45 PM
Dear Damayanti Datta :

This is in response to your letter below, in which you have requested permission to reprint, in your upcoming thesis/dissertation, your the described IEEE copyrighted material, We are happy to grant this permission.

Our only requirement in regards to distributing the paper copies is that the following copyright/credit notice appears prominently on the first page of each reprinted paper, with the appropriate details filled in:

© 2007 IEEE. Reprinted, with permission, from (complete publication information).

If Louisiana State University should wish to place a copy of your IEEE copyrighted paper on its web site, we are happy to grant this permission also.

Our only requirement is that the following IEEE notice appears prominently on the first page/screen of the reprinted paper, with the appropriate details filled in:

Copyright© IEEE. Reprinted from (relevant publication info).

This material is posted here with permission of the IEEE. Such permission of the IEEE does not in any way imply IEEE endorsement of any of Louisiana State University's products or services. Internal or personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution must be obtained from the IEEE by writing to pubs-permissions@ieee.org.

By choosing to view this document, you agree to all provisions of the copyright laws protecting it.

Thank you.

Sincerely,

Jacqueline Hansson

IEEE Intellectual Property Rights Office
445 Hoes Lane
Piscataway, NJ 08855-1331 USA
+1 732 562 3828 (phone)
+1 732 562 1746(fax)
e-mail: j.hansson@ieee.org

From: "Damayanti Datta" ¡ddatta1@lsu.edu¿
To: copyrights@ieee.org
CC:
Subject: permission to publish
Date: Wednesday, June 13, 2007 2:08:54 PM
Hi,

I need permission to publish the contents of the following paper as a part of my PhD dissertation work. The paper is Reliable and Efficient Data Transfer in Wireless Sensor Networks via Out-of-Sequence Forwarding and Delayed Request for Missing Packets by Datta, Damayanti Kundu, Sukhamay. Dept. of Comput. Sci., Louisiana State Univ., Baton Rouge, LA; This paper appears in: Information Technology, 2007. ITNG '07. Fourth International Conference on Publication Date: 2-4 April 2007, On page(s): 128-133, ISBN: 0-7695-2776-0, INSPEC Accession Number: 9465309, Digital Object Identifier: 10.1109/ITNG.2007.165

Please let me know if you need any more information.
Thanks,
Damayanti Datta

Vita

Damayanti Datta was born in Kolkata, India. She received her Bachelor of Engineering degree in chemical engineering from Jadavpur University, Kolkata, India, in 1996. She worked in Paharpur Cooling Towers as a chemical engineer from 1996-1997 and in Cognizant Technology Solutions as a software engineer from 1997-2000. In August 2000, she joined the master's program in computer science at Georgia Southwestern State University and graduated in December, 2001. She joined the doctoral program in computer science at Louisiana State University in January, 2002.