ONE: ADAPTIVE **O**NE-TO-**N** **E**RROR RECOVERY IN WIRELESS SENSOR NETWORKS

by

MUSTAFA ONUR ERGİN

Submitted to the Institute of Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science
in
Computer Engineering

Yeditepe University

2007

ONE: ADAPTIVE **O**NE-TO-**N** **E**RROR RECOVERY IN WIRELESS SENSOR NETWORKS

APPROVED BY:

Prof. Dr. Şebnem Baydere ……
(Thesis Supervisor)

Assist. Prof. Dr. Gürhan Küçük ……

Assoc. Prof. Dr. Sema Oktuğ ……
(İstanbul Technical University)

DATE OF APPROVAL: ...../...../.....

ACKNOWLEDGEMENTS

MUSTAFA KEMAL ATATÜRK (1881-∞)

"Peace at Home, peace in the World."

"There are two Mustafa Kemals. One the flesh-and-blood Mustafa Kemal who now stands before you and who will pass away. The other is you, all of you here who will go to the far corners of our land to spread the ideals which must be defended with your lives if necessary. I stand for the nation's dreams, and my life's work is to make them come true."

Mustafa Kemal Atatürk, the founder of the Turkish Republic and its first President, stands as a towering figure of the 20^{th} Century. Among the great leaders of history, few have achieved so much in so short period, transformed the life of a nation as decisively, and given such profound inspiration to the world at large.

We miss him..

ABSTRACT

ONE: ADAPTIVE ONE-TO-N ERROR RECOVERY IN WIRELESS SENSOR NETWORKS

This study describes an adaptive error recovery mechanism for sink-to-sensors reliable data dissemination in multi hop wireless sensor networks. The proposed error recovery mechanism (ONE) modifies the negative acknowledgement (NACK) based Selective Reject (SR) retransmission scheme to be used for one-to-n type single-hop transmission in a cross-layered manner. The mechanism is analytically modeled and further extended to multi-hop topologies.

The data buffer to be transmitted is indexed with send and receive pointers by using bit arrays and a dynamic window is introduced on it. ONE provides a multi-hop mass data transmission with 100% reliability in one-to-n type of communications by successfully utilizing the parameters of the window and efficiently managing the data buffer.

It has been shown that the adaptive behavior of the scheme enables optimization of some system parameters that affect the total number of packets sent time to complete in a reliable session. This optimization resulted in a significant improvement in terms of time to complete the bulk transmission in the whole network. The cost of this improvement, which is an increase in the total number of packets sent, has also been kept at the minimum possible level.

The analysis of the proposed scheme is given and the effect of hop-count, node density and loss ratio parameters on the overall performance are shown. Finally, the improvements have been compared with some representative studies in the literature.

ÖZET

ONE: KABLOSUZ ALGILAYICI AĞLARDA 1'DEN N'YE UYARLANIR HATA DÜZELTİMİ

Bu çalışma çoklu atlamalı kablosuz algılayıcı ağlarda çıkış düğümünden algılayıcılara güvenilir veri dağılımı için uyarlanır bir hata düzeltme mekanizmasını açıklamaktadır. Önerilen hata düzeltme mekanizması (ONE), olumsuz alıntı tabanlı bir tekrar iletim sistemi olan Selective Reject (SR)'i tek atlamalı 1'den n'ye tip iletimde kullanılmak üzere çapraz katmanlı biçimde uyarlar. Mekanizma çözümsel olarak modellenmiş ve sonra çok atlamalı topolojilere genişletilmiştir.

İletilecek olan ara bellek, gönderme ve alma işaretçileriyle bit dizilimi kullanılarak işaretlenmiş ve üzerinde dinamik bir pencere tanımlanmıştır. ONE, pencerenin başarılı parametre kullanımı ve verimli veri ara bellek yönetimi kullanılarak 1'den n'ye tipi iletişiminde kitle veri transferini yüzde yüz güvenilirlik ile sağlar.

Sistemin uyarlanır davranışının, güvenilir bir oturumdaki toplam gönderilen paket sayısını ve tamamlanma süresini etkileyen bazı sistem parametrelerinin eniyilenmesini sağladığı göste-rilmiştir. Bu eniyileme, tüm ağ içerisindeki toptan iletimin tamamlanma süresi açısından kayda değer bir gelişmeyle sonuçlanmıştır. Toplam gönderilen paket sayısındaki artış olan bu gelişmenin maliyeti de mümkün olan en alt seviyede tutulmuştur.

Önerilen sistemin çözümlemesi verilmiş ve hoplama sayısı, düğüm yoğunluğu ve kayıp oranı parametrelerinin bütün başarım üzerindeki etkisi gösterilmiştir. Son olarak, elde edilen gelişmeler literatürdeki bazı örnek çalışmalarla karşılaştırılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

WSN	Wireless Sensor Networks
MTU	Maximum Transmission Unit
TTL	Time To Live
NACK	Negative Acknowledgement
ACK	Acknowledgement
TCP	Transport Control Protocol
CPU	Central Processing Unit
ARQ	Automatic Repeat Request
RR	Receive Ready

1. INTRODUCTION

A wireless sensor network (WSN) is the network of small autonomous devices with sensors attached on to gather the environmental data. These devices have low capacity processing power with very limited resources. WSNs typically consist of hundreds or thousands of these devices that form networks to perform a specific application. Their applications help collecting the environmental information mostly in where humans cannot reach or exist all the time. The gathered data are processed within the network and transferred to the sink nodes, which are usually more powerful than the sensor nodes, for evaluation at where the network interfaces with the human control. The WSN nodes usually run on batteries which limit their lifetime and power constraints. The applications and the algorithms that run on those nodes must meet energy constraints to maintain the network lifetime as long as possible.

In wireless sensor networks, the radio hardware on the nodes is known as a big battery-killer. Hence, energy efficient algorithms are developed and they tend to decrease the number of retransmissions required for communication. Especially for the algorithms which require bulk data to be transmitted at once, such as code propagation, hop-by-hop error recovery protocols are beneficial [7]. Due to the lack of transport layer in WSNs, the existing code propagation studies introduce their own solutions (discussed in Chapter 3.2) to the error recovery, though they are not generic. There have been some studies to design energy conserving variants of the classical Go-Back-N sliding window protocols for wireless communication [1]. These studies indicate the trade-off between energy, delay and throughput. However, since they mainly target one-to-one wireless links and do not have an adaptive nature for one-to-n communication links; they are not suitable for resource poor networks like WSNs. In WSNs, *negative acknowledgements* (NACKs) are used for loss indication rather then *acknowledgements* (ACKs) to decrease the energy spent for radio communication [7]. As cross layer data buffers are useful for resource optimization, Selective Reject(SR) retransmission scheme is a better choice than Go-Back-N as suggested in [3]. Additionally, in multi hop topologies where the transmission has a recursive behavior, the size of send/receive windows at intermediate nodes have an effect on the overall performance.

In the scope of this study, an adaptive cross-layer scheme is proposed that exploits channel characteristics of the network to optimize the total number of packets sent in a session and the data transfer time by carefully setting the send/receive window size parameters. Like in SR, a window in the segment is defined which is being transmitted. However, the functionality of the window is modified, so that it is enabled after the first packet loss. Therefore, in the proposed scheme the sender does not necessarily wait until all the buffer is transmitted. Instead, it determines where to start retransmission by using the receivers's window parameter W. Receiver utilizes another window parameter S to determine when to initiate the send session in its own neighborhood. These windows allow the application to access the data buffer earlier than it is fully received.

In this study, first the effect of window size is analyzed when the purpose is one-to-one and one-to-n single hop reliable data transmission. Then the analysis is extended to multi hop transmission. The results have revealed that network window parameters set on the send/receive data buffers can be adjusted to satisfy either delay requirements of time-critical application sessions or energy requirements of energy-critical application sessions at the cost of one another.

The structure of this thesis is as follows: WSN characteristics and technologies are discussed in Chapter 2. Chapter 3 gives an overview to the related transport layer studies and some code propagation algorithms are summarized in order to have a better understanding for the need for sink-to-sensor error recovery in wireless sensor networks. In Chapter 4 the proposed scheme is explained in detail. Chapter 5 contains the analysis of the proposed model followed by simulation results in Chapter 6. The conclusion and the future work is stated in Chapter 7.

2. WIRELESS SENSOR NETWORKS

Typically, the wireless sensor networks consist of many sensor nodes, which can be in scales of hundreds or thousands, and one or more sink nodes, which are generally more powerful. WSNs are composed of very low power and low capacity devices and they communicate and cooperate together as well as with other devices, such as actuators, in order to accomplish a specific task. For example, thinking of a forest fire detection system, there must be hundreds of wireless sensor nodes, distributed all around the area and they are expected to sense the environmental parameters continuously and send the compilation of those parameters to some higher level via sink nodes. On detection of an event, like fire, the sensor network must be capable of activating the actuators, such as fire extinguishers, and informing the control center immediately.

2.1. WSN Characteristics

There are several key components that are integrated in a typical wireless sensor network device as described in [14]:

Low-power embedded processor: The computational tasks on a WSN device include the processing of both locally sensed information as well as information communicated by other sensor. At present, primarily due to economic constraints, the embedded processors are often significantly constrained in terms of computational power (e.g., many of the devices used currently in research and development have only an eight-bit 16-MHz processors). Due to constraints of such processors, devices typically run specialized component-based embedded operating systems, such as TinyOS [21]. However, it should be kept in mind that a sensor network may be heterogeneous and include at least some nodes with significantly greater computational power. Moreover, given Moore's law [22], future WSN devices may possess extremely powerful embedded processors. They will also incorporate advanced low-power design techniques, such as efficient sleep modes and dynamic voltage scaling to provide significant energy savings.

Memory/Storage: Storage in the form of random access and read-only memory includes

both program memory (from which instructions are executed by the processor), and data memory (for storing raw and processed sensor measurements and other local information). The quantities of memory and storage on board of a WSN device are often limited primarily by economic considerations and are also likely to improve over time.

Radio transceiver: WSN devices include a low-rate, short-range wireless radio (10-100 kbps, $<$ 100m). While currently quite limited in capability too, these radios are likely to improve in sophistication over time - including improvements in cost, spectral efficiency, tunability and immunity to noise, fading, and interference. Radio communication is often the most power-intensive operation in a WSN device, and hence the radio must incorporate energy-efficient sleep and wake-up modes.

Sensors: Due to bandwidth and power constraints, WSN devices primarily support only low-data-rate sensing. Many applications call for multi-modal sensing, so each device may have several sensors on board. The specific sensors used are highly dependent on the application; for example, they may include temperature sensors, light sensors, humidity sensors, pressure sensors, accelerometers, magnetometers, chemical sensors, acoustic sensors, or even low-resolution imagers.

Geopositioning system: In many WSN applications, it is important for all sensor measurements to be location stamped. The simplest way to obtain positioning is to preconfigure sensor locations at deployment, but this may only be feasible in limited deployments. Particularly for outdoor operations, when the network is deployed in an ad hoc manner, such information is most easily obtained via satellite-based GPS. However even in such applications, only a fraction of the nodes may be equipped with GPS capability, due to environmental and economic constraints. In this case, other nodes must obtain their locations indirectly through network localization algorithms.

Power source: For flexible deployment, the WSN device is likely to be battery powered (e.g. using LiMH AA batteries). While some of the nodes may be wired to a continuous power source in some applications, and energy harvesting techniques may provide a degree of energy renewal in some cases, the finite battery energy is likely to be the most critical resource bottleneck in most WSN applications.

Depending on the application, WSN devices can be networked together in a number of ways. In basic data-gathering applications, for instance, there is a node referred to as the

sink to which all data from *source* sensor nodes are directed. The simplest logical topology for communication of gathered data is a single-hop star topology, where all nodes send their data directly to the sink. In networks with lower transmission power settings or where nodes are deployed over a large area, a multi-hop tree structure may be used for data-gathering. In this case, some nodes may act both as sources themselves, as well as routers for other sources.

One interesting characteristic of wireless sensor networks is that they often allow for the possibility of in-network processing. Intermediate nodes along the path do not act merely as packet forwarders, but may also examine and process the content of the packets going through them. This is often done for the purpose of data compression or for signal processing to improve the quality of the collected information.

2.2. WSN Platforms

The wireless sensor network nodes, which are sometimes called motes, are manufactured by several producers in various capacities and functionalities. In this section, an overview of the most commonly used sensor network platforms is given.

2.2.1. MICA Platform

Mica motes of Crossbow Inc. [19] are one of the widest used platforms. There are several different types of motes available to use in different applications. Most popular one at the moment is MicaZ platform. Also, this company provides Mica2, Mica2dot and Mica motes.

- **Mica2**

  The Mica2 motes (see Figure 2.1) appear in three models according to their RF frequency band: the MPR400 (915 MHz), MPR410 (433 MHz), and MPR420 (315 MHz). The motes use the Chipcon CC1000, FSK modulated radio. All models utilize an Atmega128L micro-controller and a frequency tunable radio with extended range. The MPR4x0 and MPR5x0 radios are compatible and can communicate with

each other. (The x = 0, 1, or 2 depending on the model / frequency band.)

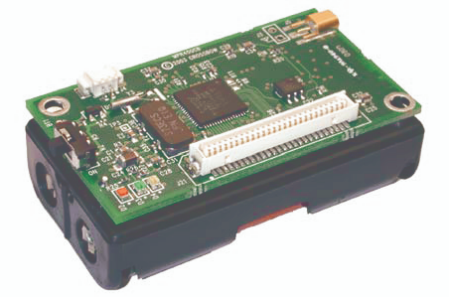

Figure 2.1. Photo of a Mica2 (MPR4x0) without an antenna.

- **MicaZ**

  The MicaZ(model MPR2400) (see Figure 2.2) is the latest generation of motes from Crossbow Technology. The MPR2400 (2400 MHz to 2483.5 MHz band) uses the Chipcon CC2420, IEEE 802.15.4 compliant, ZigBee ready radio frequency transceiver integrated with an AtMega128L micro-controller. The same Mica2, 51 pin I/O connector, and serial flash memory is used; all Mica2 application software and sensor boards are compatible with the MPR2400.

Figure 2.2. Photo of the MPR2400 MicaZ with standard antenna.

- **Mica2dot**

  The Mica2dot (see Figure 2.3) is a mote designed for applications where physical size is important. In Figure 2.3, a photo of the Mica2dot is shown next to a US quarter from the top side and the bottom side. Typically the Mica2dot has a 3 V coin-cell battery holder attached to the bottom-side, which is not included to the figure. Like the Mica2,

these are available in three models according to the frequency of the RF transceiver: the MPR500 (915 MHz), MPR510 (433 MHz), and MPR520 (315 MHz). The motes use the Chipcon CC1000 FSK-modulated radio. All models utilize an ATMega128L micro-controller and a frequency tunable radio with extended range. The Mica2dot mote has an on-board thermistor (Panasonic ERT-J1VR103J) which is a surface mount component.

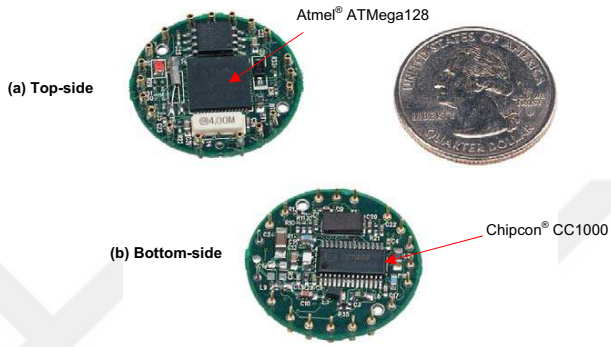

Figure 2.3. Photos of the Mica2dot shown next to a US quarter.

- **Mica**

  The Mica mote is a discontinued platform. It was the second generation mote module used in many ground breaking research and development efforts. It included an Atmel ATMega128L. It used an amplitude shift keying radio the TR1000 by RF Monolithics, Inc.

2.2.2. Tmote Platform

Tmote platform, formerly called Telos, is produced by Moteiv Corporation [20]. Tmotes have embedded sensors on and they use Texas Instruments CPUs. Currently there are two types of Tmotes: Tmote Sky and Tmote mini.

- **Tmote Sky**

  Tmote Sky (see Figure 2.4) is an ultra low power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping. Tmote Sky leverages industry standards like USB and IEEE 802.15.4 to interoperate seamlessly with other devices. By using industry standards, integrating humidity, temperature,

and light sensors, and providing flexible interconnection with peripherals, Tmote Sky enables a wide range of mesh network applications. Tmote Sky is a drop-in replacement for Moteivs successful Telos design. Tmote Sky includes increased performance, functionality, and expansion. Tmote Sky is part of a line of modules featuring on-board sensors to increase robustness while decreasing cost and package size.

Its features are:

- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- 8MHz Texas Instruments MSP430 microcontroller (10k RAM, 48k Flash)
- Integrated ADC, DAC, Supply Voltage Supervisor, and DMA Controller
- Integrated onboard antenna with 50m range indoors / 125m range outdoors
- Integrated Humidity, Temperature, and Light sensors
- Ultra low current consumption
- Fast wakeup from sleep ($< 6\mu$s)
- Programming and data collection via USB
- 16-pin expansion support and optional SMA antenna connector

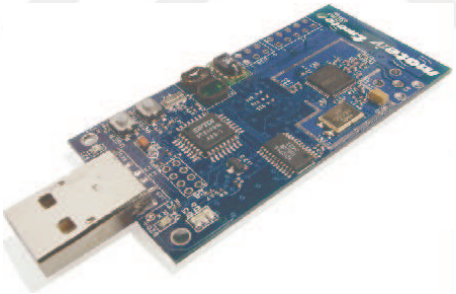

Figure 2.4. Front and Back of the Tmote Sky module

- **Tmote Mini**

  Tmote Mini builds on the success of the Tmote Sky platform, and sets a new standard for flexibility, extensibility, and cost-effectiveness. The Tmote Mini combines a Texas Instruments MSP430 microcontroller with a TI/Chipcon CC2420 low-power radio in an industry-standard miniSDIO form factor. The Tmote Mini forms the ”mote core” of a WSN solution, putting all the required hardware and software together in a single one square inch package. This modular approach allows developers, OEMs, and system integrators to concentrate on their application needs rather than WSN platform development.

The Tmote Mini comes in two configurations; the standard configuration has -25 to 0dBm of RF output, and the Tmote Mini Plus has an integrated power amplifier for a +20 dBm RF boost. Both versions are in the miniSDIO form factor, with the Mini Plus slightly longer than the standard configuration. Moreover, Tmote Mini offers multiple interfaces for UART, SPI, I2C, 1-wire as well as 8 ADC signal inputs and 2 DAC output channels. Either version can have an antenna connected through the exposed leads, or via a convenient U.FL connector for use with off-the-shelf components.

The Tmote Mini is functionally equivalent to Moteiv's Tmote Sky platform. All Tmote Sky code runs without modifications on the Tmote Mini module. Tmote Mini has the same features with the Tmote Sky platform.

2.2.3. BTNode Platform

The BTnode (see Figure 2.5) is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller. It serves as a demonstration platform for research in mobile and ad-hoc connected networks (MANETs) and distributed sensor networks. It contains both the low-power radio that is the same as used on the Berkeley Mica2 motes and a Bluetooth radio. Both radios can be operated simultaneously or be independently powered off completely when not in use, considerably reducing the idle power consumption of the device.

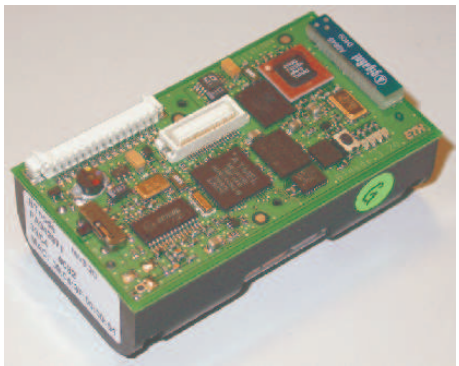

Figure 2.5. BTNode Rev.3

Its features are:

- Microcontroller: Atmel ATmega 128L (8 MHz @ 8 MIPS)
- Memories: 64+180 Kbyte RAM, 128 Kbyte FLASH ROM, 4 Kbyte EEPROM
- Bluetooth subsystem: Zeevo ZV4002, supporting AFH/SFH
- Scatternets with max. 4 Piconets/7 Slaves, BT v1.2 compatible
- Low-power radio: Chipcon CC1000 operating in ISM band 433-915 MHz
- External Interfaces: ISP, UART, SPI, I2C, GPIO, ADC, Timer, 4 LEDs
- Standard C Programming, TinyOS compatible

2.2.4. EYES Platform

EyesIFXv2 is a sensor node (see Figure 2.6) developed by Infineon for the Energy-efficient self-organizing and collaborative wireless sensor networks project EYES. Infineon has combined EYES baseband hardware with a number of optimized peripheral sets to create a series of chips aimed at specific automotive, industrial and consumer applications. Infineon has recently released a new version of the Eyes node, the EyesIFXv2.1. The components of this new board are almost the same of the older v2.0. Two leds have been added to show the radio activity. The two node versions have the same radio unit, so they can communicate without problems. Also the compiled binary images should work properly on both the platforms.

EYES nodes are equipped with an ultra-low power MSP430F1611 microcontroller by Texas Instruments, with 10 kB onchip RAM and 48 kB flash/ROM. Additionally, there is a 4Mb Atmel serial EPROM connect via the SPI bus.

2.2.5. Imote2 Platform

The Imote2 is an advanced wireless sensor node platform. It is built around the low power PXA271 XScale processor and integrates an 802.15.4 radio (CC2420) with a built-in 2.4GHz antenna. The Imote2 is a modular stackable platform and can be expanded with extension boards to customize the system to a specific application. Through the extension board connectors sensor boards can provide specific analog or digital interfaces. A battery board is provided to supply system power, or it can be powered via the integrated USB

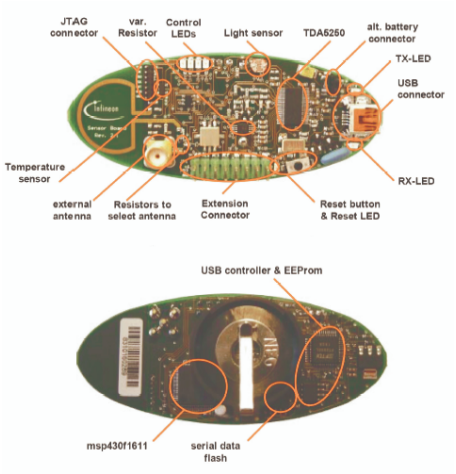

Figure 2.6. EyesIFXv2

interface. It is distinguished from the other nodes by being a relatively powerful platform and having the ability to run embedded Linux on.

Its features are:

- PXA271 XScale Processor at 13416MHz
- Wireless MMX DSP Coprocessor
- 256kB SRAM, 32MB FLASH, 32MB SDRAM
- Integrated 802.15.4 Radio
- Multi-color Status Indicator LED
- Integrated 2.4GHz Antenna, Optional External SMA Connector
- USB Client With On-board mini-B
- Rich set of Standard I/O: 3xUART, 2xSPI, I2C, SDIO, GPIOs
- Application Specific I/O: I2S, AC97, Camera Chip Interface, JTAG

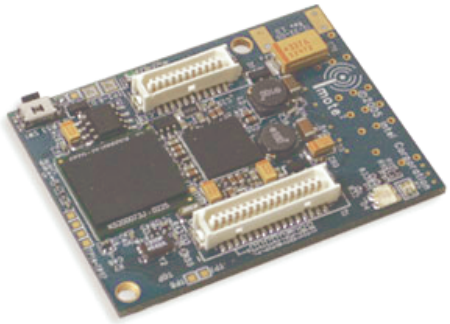

Figure 2.7. Imote2

3. BACKGROUND AND RELATED WORK

There are a few number of studies exist for reliable data dissemination in data centric networks. Most of them exist as a part of task specific studies, like in [5] and they are not generic. Many error recovery algorithms are developed for reliable event delivery from sensor-to-sink. In these delivery schemes, end-to-end reliability may not be very critical due to the sheer amount of correlated data flows and an event in the field can be tracked with a certain accuracy at the sink.

This study has been motivated by the basics of Selective Reject ARQ mechanism, which is explained in the next section. It is followed by representative studies on code propagation, which requires high reliability in sink-to-sensors type of bulk data transmission. Later, most significant examples of error recovery in WSNs are explained.

3.1. Selective Reject ARQ Mechanism

Selective Reject (SR) ARQ is a significant study for a one-to-one communication scenario. It is based on retransmission of the frames that are indicated by an SREJ (NACK) or which time out while successfully received frames are acknowledged with an RR (receive ready / ACK) packet. Figure 3.1 illustrates the SR ARQ in the transmission between nodes A and B.

SR ARQ is a more efficient mechanism compare to go-back-N, since it tries to minimize the amount of retransmissions. On the other hand, the receiver must contain storage to save post-SREJ frames until the frame in error is retransmitted, and contain logic for reinserting the frame in the proper sequence and also the transmitter must contain some logic to be able to send frames out of sequence.

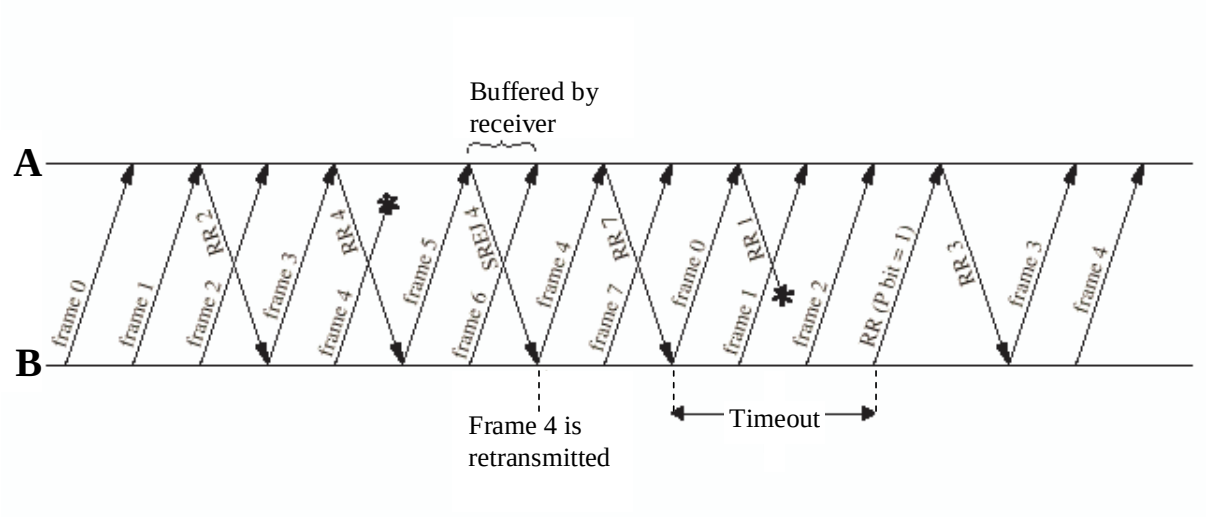

Figure 3.1. Selective Reject ARQ behavior

3.2. Code Propagation in wireless sensor networks

Code propagation is a typical example of a sink-to-sensors type of transmission. It requires a bulk of data to be transmitted at once with the need of 100% end-to-end reliability. Also, a code propagation algorithm must be energy-efficient and fast, since the primary task of the network must be kept continuous. The need for propagating the code arises from the fact that the sensor network nodes are not physically reachable for most of the time. The code running on the nodes can be buggy, or an update may be required. More extremely, the task of the network or the need to the network can change and deploying a new network of wireless sensors may not be feasible.

The lack of an applicable transport layer in wireless sensor networks forced the researchers develop their own solutions in their code propagation algorithms. Unfortunately, these solutions are very specific to the algorithms that they belong to. Hence, they are not generic and other algorithms cannot benefit from these solutions. In the following sections, some representative code propagation algorithms are summarized.

The easiest way of propagating a code is using the flooding approach. However, the cost of flooding is extremely high in comparison to the cost of any average application running within the network. Therefore, more intelligent and less costly algorithms have been

designed. The major algorithms, discussed in the following sections are; i)Trickle, which is only a version advertisement algorithm; ii)Deluge, an improvement over Trickle and is currently available for using with the latest versions of TinyOS; iii)MOAP, a Mica2 specific code propagation and update mechanism which aims to decrease the cost by decreasing the number of retransmissions and increase the reliability; and lastly iv)MNP, which has a smarter method for decreasing the number of code sources within the network and hence decreases the number of total transmitted packets.

3.2.1. Trickle

Trickle [23] is a version advertisement algorithm, which uses a polite gossip technique. In this technique, nodes periodically broadcast a summary of the code they have. But if those nodes hear a summary, which is identical to theirs, they stay quite. Upon hearing an older summary than they currently have, they start to broadcast the update. Trickle prevents the network to be flooded with packets, hence each node hears a small trickle of the packets, just enough to keep them up-to-date.

3.2.2. Deluge

Deluge [24] is a data dissemination protocol which aims to propagate large amounts of data in a multicast approach. It provides the data as fix-sized objects to the network and distributes that data in an epidemic manner. Deluge uses the advantage of the three-stage (advertise-request-data) handshaking method and is very similar to SPIN-RL [26] which supplies reliability in lossy-network models by making nodes to request data many times if some are detected as lost in broadcast networks. It can actually be considered as an improvement over Trickle [23], with using a similar technique as SPIN does, but with adding a metadata suppression system. Deluge mainly differs from Trickle by providing support for dissemination of large data.

Every node, running Deluge on, has a version number, which is incremented with each update that it admits. The nodes compare their version numbers with their neighbors when they hear an advertisement. Basically, a node stays silent if it hears an advertisement with the

same version number, accepts the update if it hears a greater version number, or advertises its own code if it hears a lower version number. The code itself is divided into pages and pages are divided into a fixed number of packets. The transmission is based upon page sending. In other words, a node can be sending or receiving packets from only one page at a time. The protocol is based on continuous version advertisements of the code image and it is NACK-based. The basic communication cycle is:

- Advertisements, containing the version number and a bit vector to indicate the complete pages that the advertiser node has, are broadcasted periodically. A non-constant delay is used between those advertisements in order to efficient use of the energy and the bandwidth. If the node is in a state of updating, the advertisements are made faster.
- If a node decides on the need for update, depending on the advertisements made, it first determines the lowest version numbered page it requires. Then it starts listening the advertisements to see if someone is offering that page.
- Upon choosing an advertiser for that required page, it sends a request, indicating the needed page and packets, to that particular node.
- When an advertiser receives a request, it waits for a while to allow selection of which page and which packets of that page to send.
- After that waiting period, the advertiser broadcasts the chosen packets.
- When a node completes reception of the packets of a page, it broadcasts the advertisement of that completed page before requesting other packets that it needs.

This algorithm does not require keeping a neighborhood table and it is easy to implement. The protocol is defined to be NACK-based, but there is no explicit NACK messages for indication of failure. Instead, the sender starts advertising the same page after the completion of the update session. If the receiver did not receive all the packets successfully, it will re-request the necessary packets from the advertiser. This mechanism does the same effect as the usual NACK system. This approach results in a less stateful process. To decrease the contention in the network, the nodes prefer to select the same advertiser which is in the same neighborhood with other requesting nodes. If a node does not select an advertiser, the other old versioned nodes will not select that node as a sender, either. Another issue that increases contention is running this protocol in the highest packet transmit rate. Since high data rates

increase the probability of collisions and recovering from collision situation is costly, Deluge prefers not to transmit its packets at the highest rate.

3.2.3. MOAP

MOAP [25] is a code distribution protocol which is specifically developed to run on Mica2 motes. It claims that a simple windowed retransmission scheme has better effects on energy and memory constraints in embedded systems. When compared to flooding using mechanisms, MOAP leads to a 60% to 90% reduction in retransmissions. It separately focuses on the three areas of code propagation mechanism; dissemination protocol, retransmission system and storage management of the code segments. For dissemination of the code, MOAP uses the Ripple [25] dissemination protocol and hence it achieves the reduction of the transmissions. Its repair mechanism is powered by a sliding window scheme and retransmissions are preferred to be unicasted in order to suppress the duplication of the messages.

MOAP states the problem in three parts. First, it focuses on dissemination of the code. The protocol extends a single-hop mechanism to a multi-hop system, in a recursive manner. The data is propagated in a so called neighborhood-by-neighborhood behavior. Within each neighborhood, a subset of the nodes is selected as sources; while the others are eventually happen to be receivers. Any receiver completes its update, becomes a source for its own neighborhoods which are not in the range of the initial source. Once a node becomes source, it publishes its new code and the other nodes, which will to receive that code, subscribe to that source. If no node subscribes itself to that source, that means there is another source within that neighborhood and the new source node stays silent. So, this technique results in one-hop away source nodes to appear in the neighborhoods, and hence the losses of packets dramatically decrease. This mechanism is called Ripple. Ripple guarantees that if there is a transmission of the code, the source is one-hop away from the receiver. But, this approach has a disadvantage. Due to the fact that; any node has to have the complete image to become a source, required time to completion of the update of the whole network seriously increases. However, the requirement of the completion of the entire image can be tolerated, but this obviously results in additional complexity.

The next issue that MOAP focuses on is the reliability mechanism. A sender would need to keep state information for all the receivers, if it was responsible for the success of the delivery. Since it will be too costly to make the sender to be responsible for reliability, the receivers are expected to maintain the reliability mechanism by keeping the state of its sender, which is just a single node. When a receiver detects that it did not received correctly and fully what it had to receive, it sends a NACK. Since all of the nodes are expected to have the same code image at the end, the missing packets are expected to be found one hop away also. Once the need for retransmission is decided, the nodes will need to make a request. If those requests for repairs are broadcasted, many nodes will reply and contention will significantly increase. Hence, a specific node must be chosen and a unicast request should be sent. But if that node fails, repairing may become impossible. To prevent this inconsistency, a source discovery mechanism can be used.

The last issue is segment management. The received segments must be placed, so that a reconstruction can be possible after a complete code dissemination. The essence of the segment management comes up with the need for determining the weather a segment is present or not. Otherwise, requests for retransmissions or handling retransmission requests will not be possible. To prevent high consumption of either the memory or the energy by keeping a presence bitmap for the whole image, a sliding window mechanism is preferred. At any point, the receiver will know if it has successfully received the packets of a specific segment window. With this mechanism, nodes will accept and store the packets up to the size of that window. However, the sent segments must fall inside of that window in order to be accepted. A random segment is not allowed and it will be ignored. This method prevents making extra memory I/O. But, out-of-order packet replacements cannot be handled in this approach.

3.2.4. MNP

MNP [5] proposes a greedy sender selection algorithm. Its purpose is to select only one node as the sender of the code in a neighborhood. It is greedy, because it tries to select the most efficient node as the one. Additionally, it uses pipelining for data propagation. And for energy constraints, MNP puts the nodes in sleep mode if they are not interested in the

propagated code.

The main property of MNP is its sender selection algorithm. After receiving the whole code image, nodes become a source. The algorithm starts with keeping a request counter and it is incremented by one, whenever a new nodes requests code. There are two types of messages to be used for sender selection: Advertisement and download request type messages. For the source node functionality, advertisement messages are used both for announcing the new versions of code, and for detecting which sources have fewer requesters than others. The advertisements are sent with the request counter value. Upon receiving an advertisement of the newer code, nodes broadcast a download request, along with indicating the received request counter and the advertiser. Those nodes, which need to be updated, send requests to all advertising nodes. This system will let a source to be aware of how many requesters does another source in that same neighborhood have. So, the sources will determine which of them will send the code first. If a source hears another source transmitting the code, it resets its request counter, recalculates its requesters and switches to sleep state for a while. When a source completes the code transmission, it also sleeps for a while and lets the others to transmit the new code. This method helps the fair distribution of the workload within the network. The sleep state period is proportional to the size of the code image and lasts for almost the time required to transmit the complete image. When it wakes up, it resets its counters and starts advertising the code again.

In the requester part of the algorithm, a pipeline mechanism shows up. Pipelining is a desirable method, when the amount of the data being sent is very large. As usual, the whole image is divided into several segments. And each segment consists of a fixed number of packets. Segments have their own increasing segment id. And the nodes will receive the segments sequentially. These segments ids are also indicated in the advertisement and download request messages. If the advertisement made is for a segment that a potential receiver node doesn't have, those receivers send a download request including that segment id in the message. If a node receives a further segment id than it expects, it makes a request for the segment that it needs in order. If a sender receives a request for a previous segment than it advertises at that moment, it starts advertising that lower numbered segment, even if it was not directly targeted by the request message. The decision of advertising the next

segment is controlled by timeouts.

Whenever a node decides to be a sender, it broadcasts a start download message. The nodes will alter to download state, upon hearing that download message with the expected segment id. The received packets are stored in the EEPROM. Simultaneously, it keeps the track of the missing packets. At the time, a sender sends an end download message, the download session ends. The receiver then starts advertising that segment, which it has just received. In the download state, if the timeout expires without any packets received by the sender, the receiver concludes that the sender died and switches to the fail state.

MNP assigns a unique id for each packet. Since all segments are fixed sized, a bitmap (it is called missingVector here) is used for loss detection by the receivers. In the beginning of the transmission of a segment, all of the bits of the missingVector are set to 1. On reception of a packet, the corresponding bit in the vector is set to 0. The arriving order of the packets is not important in this approach. Any packet can be retrieved from any node. Senders have a forwardVector to keep the list of the packets which a requester needs. The requesters put their missingVector into the request packet. The sender then marks its forwardVector according to that missingVector sent. After completion of the transmission of a segment, the sender sends a query message to its receivers. The receivers then return the list of their missing packets.

3.3. Transport Layer in Wireless Sensor Networks

Reliability in delivery is a key issue in all kind of networks. For years, many solutions have been developed to satisfy reliability requirements. However, most of them are not applicable to the low-capacity wireless sensor nodes because of their own constraints such as speed and energy issues. The protocols to fulfill the requirements of such networks must be fast enough to keep the main task working continuously while keeping the energy consumption at the minimum possible level. A good discussion in this topic is given in [15].

A wireless sensor network is an event driven paradigm that relies on the collective effort of numerous sensor nodes. This collaborative nature brings several advantages over tra-

ditional sensing including greater accuracy, larger coverage area and extraction of localized features. The realization of these potential gains, however, directly depends on the efficient reliable communication between the wireless sensor network entities, i.e., the sensor nodes and the sink.

To accomplish this, in addition to robust modulation and media access, link error control and fault tolerant routing, a reliable transport mechanism is imperative. The functionalities and design of a suitable transport solution for the wireless sensor networks are the main issues in this study.

The need for transport layer in the wireless sensor networks is pointed out in the literature [16, 17]. In general, the main objectives of the transport layer are;

i) to bridge application and network layers by application multiplexing and demultiplexing;
ii) to provide data delivery service between the sensor and the sink with an error control mechanism tailored according to the specific reliability requirement of the application layer; and
iii) to regulate the amount of traffic injected to the network via flow and congestion control mechanisms.

Although these objectives are still valid, the required transport layer functionalities to achieve these objectives in the wireless sensor networks are subject to significant modifications in order to accommodate unique characteristics of the wireless sensor network paradigm. The energy, processing, and hardware limitations of the wireless sensor nodes bring further constraints on the transport layer protocol design. For example, the conventional end-to-end retransmission-based error control and the window-based additive-increase multiplicative-decrease congestion control mechanisms adopted by the vastly used *Transport Control Protocol* (TCP) protocols may not be feasible for the wireless sensor network domain and hence, may lead to the waste of scarce resources.

On the other hand, unlike the other conventional networking paradigms, the wireless

sensor networks are deployed with a specific sensing application objective. For example, sensor nodes can be used within a certain deployment scenario to perform continuous sensing, event detection, event identification, location sensing, and local control of actuators for a wide range of applications such as military, environment, health, space exploration, and disaster relief.

The specific objective of a sensor network also influences the design requirements of the transport layer protocols. For example, a sensor network deployed for different applications may require different reliability level as well as different congestion control approaches.

Consequently, the development of transport layer protocols is a challenging effort, since the limitations of the sensor nodes and the specific application requirements primarily determine the design principles of the transport layer protocols. With this respect, the main objectives and the desired features of the transport layer protocols that can address the unique requirements of the wireless sensor networks paradigm can be stated as follows:

- *Reliable Transport:* Based on the application requirements, the extracted event features should be reliably transferred to the sink. Similarly, the programming/retasking data for sensor operation, command and queries should be reliably delivered to the target sensor nodes to assure the proper functioning of the wireless sensor network.
- *Congestion Control:* Packet loss due to congestion can impair event detection at the sink even when enough information is sent out by the sensors. Hence, congestion control is an important component of the transport layer to achieve reliable event detection. Furthermore, congestion control not only increases the network efficiency but also helps conserving scarce sensor network resources.
- *Self-configuration:* The transport layer protocols must be adaptive to dynamic topologies caused by node mobility/failure/temporary power-down, spatial variation of events and random node deployment.
- *Energy Awareness:* The transport layer functionalities should be energy-aware, i.e., the error and congestion control objectives must be achieved with minimum possible energy expenditure. For instance, if reliability levels at the sink are found to be in excess of that required for the event detection, the sensor nodes can conserve energy

by reducing the amount of information sent out or temporarily powering down.

- *Biased Implementation:* The algorithms must be designed such that they mainly run on the sink with minimum functionalities required at sensor nodes. This helps conserving limited sensor resources and shifts the burden to the powerful sink.
- *Constrained Routing/Addressing:* Unlike protocols such as TCP, the transport layer protocols for wireless sensor networks should not assume the existence of an end-to-end global addressing. It is more likely to have attribute-based naming and data-centric routing, which addresses different transport layer approaches.

Due to the application-oriented and collaborative nature of the wireless sensor networks, the main data flow takes place in the *forward path*, where the sensor nodes transmit their data to the sink. The *reverse path*, on the other hand, carries the data originated from the sink such as programming/retasking binaries, queries and commands to the sensor nodes. Although the above objectives and the desired features are common for the transport layer protocol, require different functionalities to handle the transport needs of the forward and reverse paths.

For example, the correlated data flows in the forward path are loss-tolerant to the extent that event features are reliably communicated to the sink. However, data flows in the reverse channel are mainly related to the operational communication such as dissemination of the new operating system binaries, which usually requires 100% delivery. Therefore, a reliability mechanism would not suffice to address the requirements of both forward and reverse paths. Hence the transport layer issues pertaining to these distinct cases are explained in the following sections.

3.3.1. ESRT: Event-to-Sink Reliable Transport

ESRT [9] provides a sensor-to-sink reliability model and it focuses on reliable event detection and event transfer. It is one of the first studies that appears in the context of sensor-to-sink reliability.

The algorithms of ESRT are developed to run mostly on the sink, with less functionality

required at resource constrained sensor nodes. If the sensor-to-sink reliability is lower than required, ESRT adjusts the reporting frequency of sensor nodes aggressively in order to reach the target reliability level faster. If the reliability is higher than required, then ESRT reduces the reporting frequency in order to conserve energy while still keeping the reliability at a high level.

The motivation behind ESRT is the applications where the sink is not interested in the individual sensor nodes' data, but the information within the event radius as illustrated in Figure 3.2 .

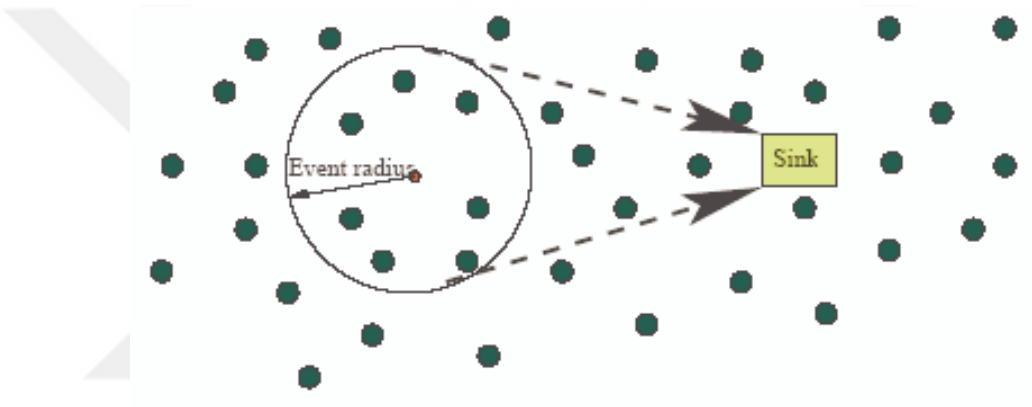

Figure 3.2. Typical sensor network topology with event and sink.

ESRT aims to achieve reliable event detection with minimum energy expenditure and congestion resolution. Its self-configuration feature helps achieving flexibility under dynamic topologies by self-adjusting the operating point. ESRT is energy-aware because it conserves energy by reducing the nodes' reporting rate if the reliability levels at the sink are more that required. It also provides a congestion control by conservatively reducing the reporting rate to reduce energy consumption. Individual node IDs are not required in ESRT for operation. This is also in tune with the event-to-sink model rather than the traditional end-to-end model. The algorithms of ESRT mainly run on the sink with minimum functionalities required at sensor nodes. This is done to conserve limited sensor resources and shifts the burden to the high-powered sink.

The transport problem in WSN is defined in ESRT as configuring the reporting rate, f, of sensor nodes so as to achieve the required event detection reliability, R, at the sink with

minimum resource utilization.

Let the desired reliability as laid down by the application be R. Hence, a normalized measure of reliability is $\eta = \frac{r}{R}$, where r is the initial reliability. Therefore, η_i denotes the normalized reliability at the end of decision interval i. ESRT defines five characteristics in reliability as below:

- (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)

where ϵ is an application dependent protocol parameter and f_{max} is the upper limit of data injection for which the network is unable to handle more frequent injections because of congestions. The sink derives a reliability indicator η_i at the end of decision interval i. Coupled with a congestion detection mechanism (to determine $f \lessgtr f_{max}$), this can help the sink determine in which of the above regions the network currently resides.

The operation of ESRT is closely tied to the state of the network at the end of i^{th} decision interval, S_i, where;

$$S_i \in \{(NC, LR), (NC, HR), (C, HR), (C, LR), OOR\}$$

The primary motive of ESRT is to achieve and maintain operation in state **OOR**. Hence, the aim is to configure the reporting frequency f to achieve the desired event detection accuracy with minimum energy expenditure. To help accomplish this, ESRT uses a congestion control mechanism that serves the dual purpose of reliable detection and energy conservation.

The network can reside in any one of the five states S_i. Depending on the current state S_i, ESRT calculates an updated reporting frequency f_{i+1}, which is then broadcast to the

sensor nodes. For example, if $S_i \in \{(NC, LR), (C, LR)\}$, the observed reliability levels are inadequate to detect the desired event features. In such a case, ESRT aggressively updates the reporting frequency to reliably track the event as soon as possible.

This self-configuring nature of ESRT helps it adapt to dynamic topology and random deployment. Also, in the case where the reliability levels exceed those for event detection, it decreases f in a controlled manner in order to conserve energy.

The algorithms of ESRT mainly run on the sink, with minimal functionality at the sensor nodes. The sensor nodes only need to listen to the sink broadcast at the end of each decision interval and update their reporting rates and they must deploy a simple local congestion detection support mechanism.

ESRT identifies the current state S_i from the reliability indicator η_i computed by the sink for decision interval i and from a congestion detection mechanism, using the decision boundaries.

Depending on the current state S_i, and the values of f_i and η_i , ESRT calculates the updated reporting frequency f_{i+1} and broadcasts to the sensor nodes. At the end of the next decision interval, the sink derives a new reliability indicator n_{i+1} corresponding to the updated reporting frequency f_{i+1} of sensor nodes. In conjunction with any congestion reports, ESRT then determines the new network state S_{i+1}. This process is repeated until the optimal operating region(state OOR) is reached.

State operations are explained below:

1. (NC,LR) (No Congestion, Low Reliability) : In this state, no congestion is experienced and the achieved reliability is lower than that required, i.e. $\eta < 1 - \epsilon$ and $f < fmax$. In order to improve the reliability to acceptable levels, the sensor information must be increased. To maintain the event-to-sink reliability, the reporting frequency f is aggressively increased to attain the required reliability as soon as possible.
2. (NC,HR) (No Congestion, High Reliability) : In this state, the required reliability level

is exceeded, and there is no congestion in the network, i.e. $\eta > 1 + \epsilon$ and $f \leq f_{max}$. This is because sensor nodes report more frequently than required. Therefore the reporting frequency should be reduced in order to conserve energy. Then, the sink reduces reporting frequency f in a controlled manner so that the event-to-sink reliability is always maintained.

3. (C,HR) (Congestion, High Reliability) : In this state, the reliability is higher than required, and congestion is experienced, i.e., $\eta > 1$ and $f > f + max$. This is due to the unique feature of WSN where required event detection reliability can be attained even when some of the sensor data packets are lost. In this case ESRT decreases the reporting frequency in order to avoid congestion and conserve energy in sensor nodes. In this case also, this decrease should be performed carefully such that the event-to-sink reliability is always maintained.
4. (C,LR) (Congestion, Low Reliability) : In this state the observed reliability is inadequate and congestion is experienced, i.e., $\eta \leq 1$ and $f > f_{max}$. So, ESRT reduces reporting frequency aggressively in order to bring the network to state OOR as soon as possible.
5. OOR (Optimal Operating Region) : In this state, the network is operating within ϵ tolerance of the optimal point, where the required reliability is attained with minimum energy expenditure. Hence, the reporting frequency of sensor nodes is left unchanged for the next decision interval.

Network State (S_i)	Description	ESRT Action
(NC,LR)	No Congestion, Low Reliability	Multiplicative increase f Achieve required reliability as soon as possible
(NC, HR)	No Congestion, High Reliability	Decrease f conservatively Cautiously reduce energy consumption so as not compromise on reliability
(C,HR)	Congestion, High Reliability	Decrease f aggressively to state (NC,HR) to relieve congestion Then follow action in (NC,HR)
(C,LR)	Congestion, Low/equal Reliability	Decrease f exponentially
		Relieve congestion as soon as possible
OOR	Optimal Operating Region	f remains unchanged

Table 3.1. The summary of the ESRT protocol operation in each of the five states.

3.3.2. RMST: Reliable Multi-Segment Transport

RMST [7] is a transport layer protocol to provide reliable multi-segment sensor-to-sink event delivery and it is designed to provide reliability for Directed Diffusion [8], which is a data-centric, data dissemination paradigm for WSNs. RMST is NACK-based and losses are repaired hop-by-hop which are detected by the receivers. It benefits from diffusion routing, adding minimal additional control traffic. RMST guarantees delivery, even when multiple hops exhibit very high error rates.

RMST was designed to run in conjunction with Directed Diffusion [12] which provides multipoint-to-multipoint communication for sensor nets much like traditional multicasting does so for wired nets. Sensitivity to energy conservation, data-centric routing, and the limitation of traffic volume via in-network processing are examples of some motivations to shape directed diffusion.

In diffusion, a sink subscribes to an interest that names a particular type and source of data. The naming of data is accomplished via attribute-value pairs. For example, an interest in counting how many people pass through a particular geographic region could be injected into an arbitrary node in a sensor net (i.e. an access point). Sensor node applications that have data available publish the fact, alerting the local diffusion code to look for matching interests. The sensors whose publications match a given interest, and the collection of sinks that expressed that interest, constitute a group that will eventually be connected by a distribution tree.

Diffusion is built on a modular architecture that allows for great flexibility in adapting diffusion to specific applications. There is a diffusion core that communicates directly to the MAC layer below it and installable modules, called filters, above. Much like streams modules that can be assembled to create a particular networking stack, filters are installed via core diffusion to influence routing or add arbitrary application-specific behavior to a sensor net. The basic directed diffusion algorithm is implemented in the gradient filter. The gradient filter can be thought as a networking layer pushed on top of the MAC layer. Message traffic arriving at the core is passed to the highest priority filter.

The RMST protocol is implemented as a filter that could be attached to any diffusion node on an as needed basis without recompilation of the diffusion core or Gradient filter. Figure 3.3 demonstrates the relationship of RMST to a basic diffusion node. Two modes are defined; caching and non-caching, which are configurable at run time.

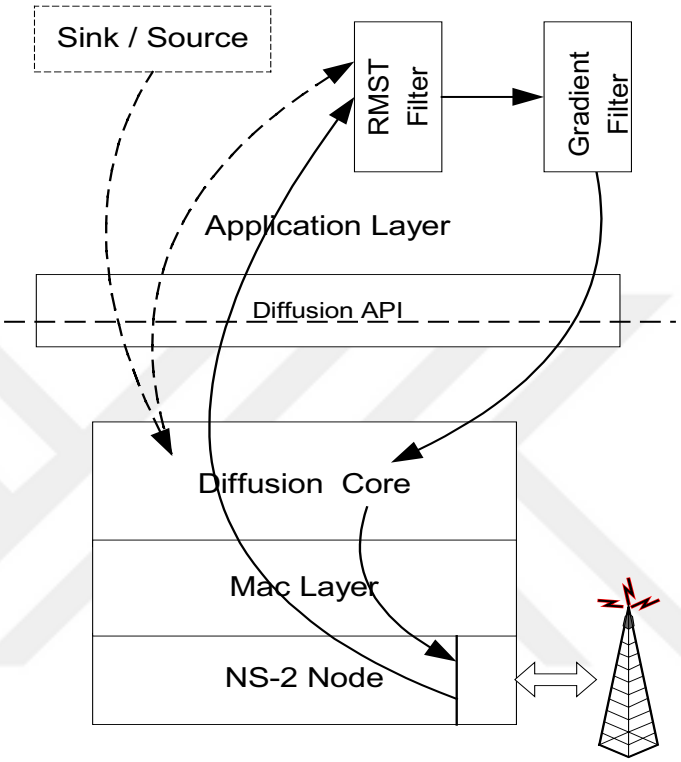

Figure 3.3. Layering of Diffusion with RMST filter.

Reliability in RMST refers to the eventual delivery to all subscribing sinks of any and all fragments related to a unique RMST entity. A unique RMST entity is a data set consisting of one or more fragments coming from the same source. Delivery order, which is not guaranteed, is transparent to the clients of RMST. RMST does not include any real time guarantees.

RMST addresses two distinct transport services that need to be added to diffusion: effective management of the fragmentation and reassembly of units based on application semantics, and guaranteed delivery. Although these requirements are orthogonal, many applications require both.

In RMST, receivers are responsible for detecting whether or not a fragment needs to

be re-sent. The term ”receiver” here, does not necessarily mean sink. In the non-caching mode, only sinks monitor 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 sensor.

There are two types of loss detected by a ”receiver”: a ”hole” in a sequence of fragments, and a truncated sequence. When a hole in a sequence of fragments is detected, the missing fragments should be specifically requested. This amounts to a selective-request ARQ-based behavior. 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.

When a node fails, the normal behavior of Diffusion is to reestablish a new set of data gradients via an exploratory interest. To this extent sensor networks are self-repairing. RMST benefits from the underlying diffusion behavior related to failed nodes. RMST can rely on the mechanisms in diffusion that guarantee the eventual discovery of a path from sensor-to-sink.

In caching mode, the caching of fragments along reinforced paths is used to limit power loss due to end-to-end retransmission. In non-caching mode, the underlying MAC layer is exploited to limit the transport layer overhead.

There is a single control message generated by RMST, the NACK, which must be defined by an attribute. Loss detection is primarily timer driven in RMST. Where loss detection occurs depends on whether a node is configured for caching or non-caching mode. In non-caching mode, only sinks set timers to detect loss. In caching mode, each caching node on the reinforced path from sensor-to-sink detects loss. The basic mechanism for loss detection is a watchdog timer. A watchdog timer is instantiated for each new flow (RmstNo) that is added to a caching nodes RMST database. 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.

Caching at the sinks makes delivery order unimportant, and allows for wide latitude

in the relationship between the watchdog timer interval and the sensor send rate. When the watchdog interval is significantly slower than the send rate, the cache size must increase to retain more incomplete sets of fragments. Setting the watchdog timer faster than the send rate, minimizes the required cache size, but increases NACK traffic.

The only control message added to normal diffusion by RMST is the NACK. NACKs are unicasted in the reverse direction along the reinforced path from sensor-to-sink. When the RMST filter gets a cache hit for a NACKed fragment, it unicasts that fragment to the requesting neighbor. When an RMST filter 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 sensor. In caching mode, the natural progression of traffic from sensor-to-sink, causes holes to be sensed sooner upstream, thus making NACK forwarding an unlikely event.

The reinforced paths that diffusion constructs for control and data are unidirectional, from sensor to sink. RMST needs a back-channel in order to deliver NACKs to upstream neighbors. RMST filters snoop at the last hops of reinforcement messages in order to construct reverse reinforced paths (in a distributed fashion). The creation of the back-channel is free of charge in that no additional transmissions or control data traffic are needed to create it. There is a back-channel associated with every flow. The back-channel is maintained in both caching and non-caching modes.

In the case of node failure a new reinforced path will be established by diffusion. Some nodes that aren't on the original reinforced path may later reside on it. The RMST filter automatically adapts to any such changes by creating a new back-channel that mirrors the new reinforced path. In caching mode, an RMST node that suddenly finds itself on a reinforced path (mid-sequence) will begin caching from the first received fragment on. NACKs for earlier fragments will be forwarded on the back-channel toward the sensor. In effect, the RMST filter will transfer responsibility for back-channel and caching maintenance to the new reinforced path.

In caching mode, a node maintains a local cache of traffic in progress or recently trans-

mitted. In non-caching mode, only the sensors and sinks maintain a cache. The cache is indexed by the application specific flow id. 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, used by the watchdog timer, 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. On receipt of a fragment a caching mode filter must identify missing or late fragments and add them to the hole map.

3.3.3. PSFQ: Pump Slowly, Fetch Quickly

The main idea of PSFQ [6] is to distribute the data from a sensor node relatively slowly, but letting the nodes that detect data loss to fetch for any missing segment from the intermediate neighbors very quickly.

PSFQ represents a simple approach with minimum requirements on the routing infrastructure (as opposed to IP multicast/unicast routing requirements), minimum signaling thereby reducing the communication cost for data reliability, and responsive to high error rates allowing successful operation under highly error-prone conditions.

The lost messages are detected when a higher sequence number than expected is received at a node triggering the fetch operation. Such a system is equivalent to a negative acknowledgement system. PSFQ is designed to achieve the following goals:

- to ensure that all data segments are delivered to all the intended receivers with minimum support from the underlying transport infrastructure
- to minimize the number of transmissions for lost detection and recovery operations with minimal signaling;
- to operate correctly even in an environment where the radio link quality is very poor;
- to provide loose delay bounds for data delivery to all the intended receivers.

PSFQ provides three functions: message relaying (pump operation), relay-initiated er-

ror recovery (fetch operation) and selective status reporting (report operation). A user (sensor node) injects messages into the network and intermediate nodes buffer and relay messages with the proper schedule to achieve loose delay bounds. A relay node maintains a data cache and uses cached information to detect data loss, initiating error recovery operations if necessary. As in many negative acknowledgement systems, there is no way for the sensor node to know when the receivers have received the data messages. First, the data segments must be retained indefinitely at the sensor for possible retransmissions. Next, it is important for the user to obtain statistics about the dissemination status (e.g., the percentage of nodes that have obtained the complete execution image for a re-tasking application) in the network as a basis for subsequent decision-making, (e.g., the correct time to switch over to the new task in the case of re-tasking). Therefore, it is necessary to incorporate a feedback and reporting mechanism into PSFQ that is flexible (i.e., adaptive to the environment) and scalable (i.e., minimize the overhead).

A number of concerns associated with the PSFQ model are related to the timer issues that control the loose service properties, such as, statistical delay bounds. Important protocol parameters include message pumping speed and loss recovery speed.

There are three main operations are defined in PSFQ; pump, fetch and report operations.

In the case where a specific node (instead of a whole group) needs to be addressed, PSFQ can work on top of existing routing or data dissemination scheme, (e.g. directed diffusion, DSDV [27], etc.), to achieve reliability. A user node uses TTL-based methods to control the scope of its re-tasking operation; note that, since the term ”sensor” in sensor network usually denotes a sensor node which has sensed data to be sent, ”user node” is used in PSFQ to refer to a node which distributes the code segments to avoid confusion. To enable local loss recovery and in-sequence data delivery, a data cache is created and maintained at intermediate nodes.

An ”inject message” is defined to associate with the pump operation in PSFQ. The **pump** operation is important in controlling four performance factors associated with the

example re-tasking application. First, the timely dissemination of code segments to all target nodes used for re-tasking the sensor nodes. Second, to provide basic flow control so that the re-tasking operation does not overwhelm the regular operations of the sensor network, (e.g., monitoring environmental conditions). Next, for densely deployed sensor networks in which nodes are generally within transmission range of more than one neighboring node, avoiding redundant messaging is needed to save power and to minimize contention/collision over the wireless channel. Finally, it is wanted to localize loss, avoiding the propagation of loss events to downstream nodes. This requires mechanisms to ensure in-sequence data forwarding at intermediate nodes. The first two performance factors require proper scheduling for data forwarding. A simple scheduling scheme is adopted, which uses two timers T_{min} and T_{max} for scheduling purposes.

A user node broadcasts a packet to its neighbors every T_{min} until all the data fragments has been sent out. Then, the neighbors that receive this packet will check their local data caches discarding any duplicates. If message arrives for the first time, PSFQ will buffer the packet and decrease the TTL field in the header by 1. If the TTL value is not zero and there is no gap in the sequence number, then PSFQ sets a schedule to forward the message. The packet will be delayed for a random period between T_{min} and T_{max} and then relayed to its neighbors that are one or more hops away from the sensor. In this specific reference case, PSFQ simply rebroadcast the packet. The data cache has several potential uses, one of which is loop prevention, i.e., if a received data message has a matching data cache entry then the data message is silently dropped. A packet propagates outward from the sensor node up to TTL hops away in this mode. The random delay before forwarding a message is necessary to avoid collisions because RTS/CTS dialogues are inappropriate in broadcasting operations when the timing of rebroadcasts among interfering nodes can be highly correlated.

A node goes into **fetch** mode once a sequence number gap in a file fragments is detected. A fetch operation is the proactive act of requesting a retransmission from neighboring nodes once loss is detected at a receiving node. PSFQ uses the concept of ”loss aggregation” whenever loss is detected; that is, it attempts to batch up all message losses in a single fetch operation whenever possible.

PSFQ aggregates loss such that the fetch operation deals with a ”window” of lost packets instead of a single packet loss. Also, in a dense network where a node usually has more than one neighbor, it is possible that each of its neighbors only obtains or retains part of the missing segments in the loss window. PSFQ allows different segments of the loss window to be recovered from different neighbors. In order to reduce redundant retransmissions of the same segment, each neighbor waits for a random time before transmitting segments, (i.e., sets a retransmission timer to a random value, and sends the packet only when the timer goes off). Other nodes that have the data and scheduled retransmissions will cancel their timers if they hear the same ”repair” (i.e., retransmission of a packet loss) from a neighboring node. The retransmitted packets and the fetch control messages can also get lost. This will likely cause multiple gaps in sequence number of messages received by a node after several such failures. Aggregating multiple loss windows in the fetch operation increases the likelihood of successful recovery in the sense that as long as one fetch control message is heard by one neighbor all the missing segments could be resent by this neighbor.

NACK message associate with the fetch operation is defined as the control message that requests a retransmission from neighboring nodes. The loss window represents a pair of sequence numbers that denote the left and right edge of a loss window. When there is more than one sequence number gap, each gap corresponds to a loss window and will be appended into the NACK message.

In fetch mode, a node aggressively sends out NACK messages to its immediate neighbors to request missing segments. If no reply is heard or only a partial set of missing segments are recovered within a period T_r then the node will resend the NACK every T_r interval (with slight randomization to avoid synchronization between neighbors) until all the missing segments are recovered or the number of retries exceed a preset threshold thereby ending the fetch operation. The first NACK is cancel (to keep the number of duplicates low) in the case where a NACK for the same missing segments is overheard from another node before the NACK is sent. In general, retransmissions in response to a NACK coming from other nodes are not guaranteed to be overheard by the node that cancelled its first NACK.

Neighbors do not relay NACK messages unless the number of times the same NACK

is heard exceeds a predefined threshold while the missing segments requested by the NACK message are no longer retained in a nodes data cache. In this case, the NACK is relayed once, which in effect broadens the NACK scope to one more hop to increase the chances of recovery.

Each neighbor that receives a NACK message checks the loss window field. If the missing segment is found in its data cache, the neighboring node schedules a reply event (sending the missing segment) at a random time between (0, T_r). Neighbors will cancel this event whenever a reply to the same NACK for the same segment (same file ID and sequence number) is overheard. In the case where the loss window in a NACK message contains more than one segment to be resent, or more than one loss window exists in the NACK message, then neighboring nodes that are capable of recovering missing segments will schedule their reply events such that packets are sent (in-sequence) at a speed that is not faster than once every T_r.

Nodes in fetch mode maintain their own loss windows to keep track of missing segments. When a fetching node receives a portion of a missing segment that only represents a partial amount of its loss window, it needs to update its loss windows accordingly, possibly splitting the loss window or creating new loss windows.

The fetch operation is a reactive loss recovery scheme in the sense that a loss is detected only when a packet with higher sequence number is received. However, if the last segment of a file is lost, there is no way for the receiving node to detect this loss since no packet with higher sequence number will be sent. In addition, if the file to be injected into the network is small (e.g., a script instead of binary code), it is not unusual to lose all subsequent segments up to the last segment following bursty loss. In this case, the loss is also undetectable and thus non-recoverable with such a reactive loss detection scheme. In order to cope with these problems, PSFQ provides a proactive fetch operation such that a node can also enter fetch mode proactively and send a NACK message for the next segment or the remaining segments if the last segment has not been received and no new packet is delivered after a period of time T_{pro}. When a proactive fetch operation is triggered, a node will manually create a loss event and send out a NACK control message with the desired loss window. The detailed use and

basic analysis of T_{pro} is explained in [6].

PSFQs **report** message and feedback mechanisms are addressed to minimize the number of messages used for feedback purposes. The report message is designed to travel from the furthest target node back to the user on a hop-by-hop basis. Each node enroute toward the user is capable of piggybacking their report message in an aggregated manner. Nodes can add/append their own feedback information to the original report message sent by the most distant target node as it propagates back toward the user that initially requested the report.

A node enters the report mode when it receives an inject data message with the report bit set in the TTL field. The user node sets the report bit in the inject message whenever it needs to know the latest status of the surrounding nodes. To reduce the number of report messages and to avoid report implosion, only the last hop nodes will respond immediately by initiating a report message sending it to its parent node at a random time between (0, Δ). Each node along the path toward the sensor node will piggyback their report message by adding their own node ID and sequence number pair into the report, and then propagate the aggregated report toward the user node. Each node will ignore the report if it found its own ID in the report to avoid looping. Nodes that are not last hop nodes but are in report mode will wait for a period of time ($T_{report} = T_{max} \times TTL + \Delta$) sufficient to receive a report message from a last hop node, enabling it to piggyback its state information. In this case a node will create a new report message and send it prior to relaying the previously received report that had no space remaining to piggybacking data. This ensures that other nodes route toward the user node will use the newer report message rather than creating new reports because they themselves receive the original report with no space for piggybacking additional status.

3.4. Summary

In this chapter the basic functionality of the Selective Reject (SR) algorithm is explained, which is also the basic motivation of this study. It is an effective mechanism for one-to-one type of communication however applying SR to a one-to-n type wireless communication will be very costly. It is followed by the need and characteristics of the transport

layer in WSNs. The code propagation examples are given in order to have a better understanding for the importance of reliable bulk data transmission. Later, state-of-the-art in the transport layer reliability techniques in WSNs are given. Three significant studies have been taken as representative studies; ESRT, RMST and PSFQ. ESRT provides an efficient mechanism in the data collection and event detection in wireless sensor networks. However, it focuses on sensors-to-sink data delivery, which does not directly overlap with the scope of this theses. RMST points that the use of NACK mechanism will significantly decrease the cost of data transfer in wireless sensor networks but it aims on sensors-to-sink delivery, too. PSFQ is a representative work for transport layer solutions in sink-to-sensors type of data dissemination. It introduces an efficient and low cost solution to the problem. The drawback of the PSFQ is its time requirements, which is too costly for the time-critical sensor network applications. ONE attacks to the field for this need. A comparison of these algorithms are given in Table 3.2 in terms of the metrics explained in Section 3.3.

Algorithm	Self Configuration	cached/ non-cached	Constrained Addressing	sink-to-sensor/ sensor-to-sink	Speed
ESRT	Yes	non-cached	No	sensor-to-sink	High
RMST	No	both	Yes	sensor-to-sink	Medium
PSFQ	No	cached	Yes	sink-to-sensor	Low

Table 3.2. Comparison of ESRT, RMST and PSFQ in terms of transport layer constraints

4. ONE: Adaptive One-to-N Error Recovery in Wireless Sensor Networks

The motivation behind the proposed protocol is to generalize the classical one-to-one SR mechanism to one-to-n sensor topologies, with deriving a ”request based” error recovery. In the nature of wireless sensor networks, each node is associated with a role to keep the continuous operation of the network. This role can either be associated to one specific node, like a sink node, or many nodes collaborated together to fulfill a dedicated task, like detecting an environmental phenomenon.

In an effective data propagation, the nodes must distinguish their roles until the 100 % delivery is achieved. When the high level application has data to propagate into the network, it calls the services of ONE from the transport protocol. ONE assigns the initiator of the bulk transmission in a neighborhood as the sender, while others eventually are receivers. Sender node keeps an application level buffer of size K, which will be transmitted to all its neighbors(receivers). Every node maintains two bit-arrays to keep track of the packets they expect to receive, and intend to send. The state diagram of the algorithm is shown in Fig 4.1.

4.1. Sender Algorithm

Sender node initiates a session by sending the first available packet in its buffer and ends the session when it goes into the *DONE* state. In the beginning of the transmission, the sender enters to *TRANSMIT* state, and starts to send the data packets from its application buffer until a request for the missing packets is made. Each packet contains the id of that session, its sequence number in the buffer and the total size of the data buffer to be transmitted to let the receivers allocate and maintain their data structures.

Sender does not wait for an acknowledgement for each single packet, the only cause to resend a packet, is receiving a request (which is a NACK with receive index). On reception of a request, it stops the transmission, switches to *COLLECT_REQUESTS* state and broadcasts

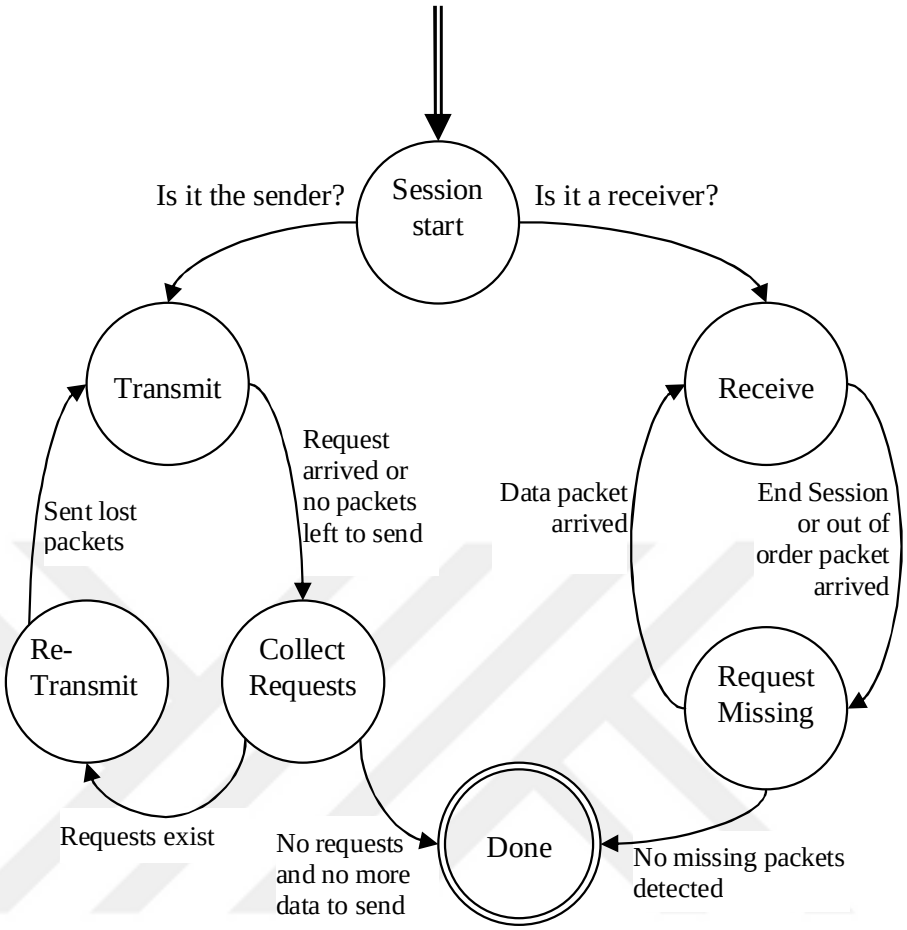

Figure 4.1. The state Diagram of the algorithm

a notification to the neighborhood. Upon reception of this notification, all other receivers, send their requests for the missing packets to their sender node (See Section 4.2). Then the sender combines all the requests it receives.

These request packets include the sender id and a bit-array of required packets (RPA). Each receiver keeps a bit-array (RPA) to identify successfully received and lost packets. The sender keeps another bit-array of to-be-sent packets (SPA) to determine the next packet to be sent. Collecting requests into the sender's SPA is illustrated in Figure 4.2.

During the transmission state, the RPA in the first received request packet is set as the SPA. The latter request packets are bitwise OR'ed to the SPA. The requests are sent for just once, unless a receive timeout fires. Sender will respond only to the requests, which include its own id. Otherwise, the request packet will be discarded. The algorithm is shown

REQUEST COLLECTION: (from 1-to-4 network)

Buffer index	1	2	3	4	5	6	7	8	9	10	11	12	----	K-1	K
(receiver 1) RPA	0	1	0	0	1	0	0	0	0	1	1	1	…..	1	1
(receiver 2) RPA	0	0	0	0	1	1	0	0	0	1	1	1	…..	1	1
(receiver 3) RPA	0	0	1	0	1	0	0	0	0	1	1	1	…..	1	1
(receiver 4) RPA	0	0	0	0	1	0	0	1	0	1	1	1	…..	1	1
+															
(sender) SPA	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**…..**	**1**	**1**

Figure 4.2. Collecting requests

in Figure 4.4.

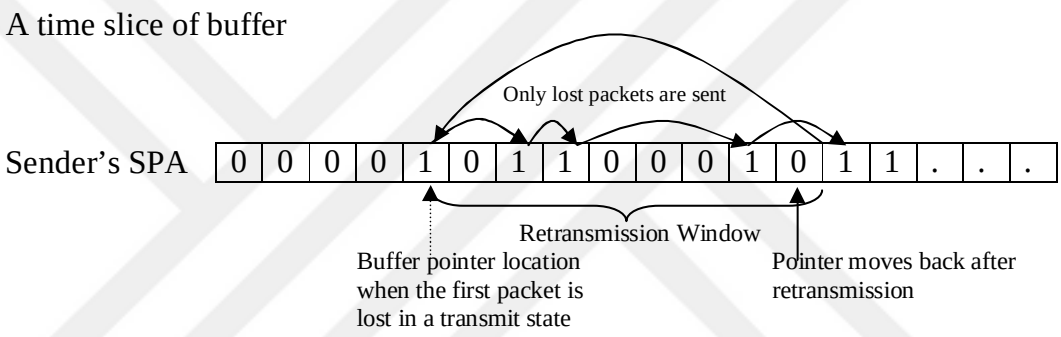

Figure 4.3. Window management

```
function forward() {
   while ( there are packets to send )
   {    broadcast (next packet);
        sleep (Propagation_rate);    }
   broadcast(propagation_end_message);
}
event MessageReceive(packet) {
   if (packet.type = request  AND  packet.target = SELF_ID)
   {    State = COLLECT_REQUESTS;
        collectOtherRequests(packet); }      else { return; }
}
function collectOtherRequests(packet) {
   sleep (collection_time);
   Self.SPA |= packet.RPA;
   State = RETRANSMIT;  forward();
}
```

Figure 4.4. Sender Algorithm

The sender switches its state to *RETRANSMIT* state after collecting the requests from

all its receivers. Among all the sent packets, requested (lost) ones are retransmitted in this state, as illustrated in Figure 4.3. When retransmission ends, sender enters back to the *TRANSMIT* state and the transmission continues from where it was interrupted. After the retransmission state, if the sender receives another request packet that is sent to itself, the above procedure is repeated.

After transmitting the last packet, sender broadcasts the request notification to the receivers and collects the requests to retransmit lost packets. After each notification broadcast, the sender waits enough to collect the requests, replies to requests if any made and broadcasts another notification. If no request arrives after θ consecutive notifications, it concludes that all receivers have filled their receive buffers for that session and goes into *DONE* state.

4.2. Receiver Algorithm

With the first data packet received from a new session, the node records its sender, allocates its receive buffer and RPA by using the information in the data packet and places the packet into its buffer. No extra session control mechanism is required.

Whenever a receiver node hears a packet of that session, it enters to the RECEIVE state and stores the data in its buffer. The data packets need not to be arrived in the correct order in order to be accepted. Any data packet, belonging to the current session, is accepted no matter who broadcasts it. A receiver will not respond or react unless it receives an out-of-order packet, or if it does not receive any packet until its receive timeout expires. The only response that a receiver generates is requesting the missing packets, by sending a request type packet.

When an out-of-order packet is received, receiver does not send a request immediately. First, it stores the index of the last correct ordered packet (R_c), then it starts a timer and waits for enough time to allow the sender to transmit W more packets. When this timer fires, it switches its state to *REQUEST_MISSING* and it sends a request packet, indicating its sender id, session id and what does its RPA contain at that moment. Then the sender switches its state to *COLLECT_REQUESTS* state as described in Section 4.1. So; all the other

receivers in the neighborhood get notified by the sender and they, too, send their requests. To decrease the chance of collisions during these request transmissions, the nodes chose themselves a random slot within the request collection time interval of the sender. Just like the slotted Aloha protocol, nodes logically divide the time interval into slots, and transmit their requests in the beginning of their randomly selected slots. These slots are calculated using the packet transmission time. In Figure 4.5, the pseudocode for receiver algorithm is given.

```
event MessageReceive(packet) {
   if (packet.type = data)
   {      State = RECEIVE;
          Timer.cancel();  insertData (packet); }
   else if (packet.type = END)
         { State = REQUEST_MISSING;  request_missing();  }
}
function insertData (packet) {
   buffer.insert(packet.order) = packet;
   MarkRPA(packet.order);
   if ( packet.order > Rc + W ) {
      State = REQUEST_MISSING;
   } else if ( !haveMissing() )   { State = DONE; }
           else { Timer.start(); }
}
request_missing() {
   unicast (sender, RPA);
}
event Timer.start () {
   sleep (Receive_Interval);
}
```

Figure 4.5. Receiver Algorithm

In this scheme, W corresponds to the transmission window of SR. The reason of waiting some time, enough to send W more packets, is to allow other nodes in the neighborhood to lose some data, and let the sender to retransmit more number of common packets in one retransmit state. Using a smaller W value will increase the overhead of request packets, but it will ensure that the early packet losses will be corrected sooner. Hence, the W value determines how many times the sender will go into the retransmission state. When W is 1, overhead of request packets will increase but the data buffer will be able to be used by

the higher level application earlier. Using a greater W will not change the total number of retransmitted packets, but it will change the overhead caused by **requests**. But in this case, the buffer will be available to be used by the application much later.

Another reason for a receiver to make a retransmission request is firing of the receive timer. If any of the receiver nodes does not receive any packet from its sender within a certain time interval, it again sends a request packet to its sender. If it does not receive a response from the sender after θ requests, it concludes that the sender is dead and terminates the session.

The receiver switches back to the *RECEIVE* state whenever it receives a data packet from its sender. If it has no missing packets left, it switches its state to *DONE* state and stays silent unless it is in hybrid mode.

4.2.1. Hybrid Mode

A receiver node goes into hybrid mode after it receives S packets correctly into its buffer and it starts a new session for the further hop. In this mode, it continues to place received data into its data buffer. Also, it begins to send the packets it has received so far. This mechanism allows the nodes to start new sessions earlier and eventually time to complete decreases for the whole topology.

In hybrid mode, the node will behave as both a sender and a receiver (ie. it can collect requests while it is sending its own request to its sender). It will have two internal states and both algorithms will run concurrently. In wireless sensors network devices, this concurrency can easily be achieved by using task calls and by maintaining multiple timers. In this mode, the nodes will keep both an RPA and an SPA and they will maintain them in parallel.

5. ANALYTICAL MODEL

In this chapter, ONE is analyzed in terms of total number of packets sent in the network and the completion time of all neighborhoods until all the nodes successfully receive all the packets in the session.

The symbols that are used to represent equations are:

$K \rightarrow$ Data buffer size as number of packets.
$n \rightarrow$ The number of immediate receivers.
$p \rightarrow$ Packet loss probability
$W \rightarrow$ Number of packets which are expected to be transmitted after k^{th} packet is lost.
$S \rightarrow$ Number of packets required to receive before entering to hybrid mode
$t_d \rightarrow$ Packet transmission interval (time)
$t_c \rightarrow$ Request collection time

Each of K packets to be sent, has a probability p of being lost. This loss comes from the channel characteristics of the wireless medium an it is assumed to be constant and same for each packet. As it is explained in Chapter 4, the only response that a receiver node generates is a request (NACK), and is sent some while after the first packet loss encountered after the beginning or a retransmission repeat. First packet loss on any receiver can be defined with a random variable k, which comes from geometric distribution. The receiver which has lost the k^{th} packet, waits for sender to transmit W more packets to make a missing request. If packet transmission takes t time units, then this waiting time will be $W \times t$ time units.

The collisions are not taken into account for this methodology. Since data packet transmission is one way in a neighborhood, the packets from sender to receiver nodes will less likely to collide. Though, the request packets of receivers nodes can collide. If a collision occurs in requests, the node of which packet is collided can either receive its required packets during the sender's retransmission, or it can send another request after $W \times t$ amount of time. This latency in the request does not effect the overall completion time since many requests

are made within a session.

The model for the algorithm is created in terms of the total number of packets that the sender transmits in a session. First, one-to-one single-hop analysis is stated. This analysis show how do a sender and a receiver behave on the simplest scheme. Later, the analysis is extended to one-to-n single-hop transmission case.

5.1. Single-hop One-to-One Analysis

Figure 5.1 illustrates how the receiver and the sender transmit their packets and respond to them. k is the order of packet, which gets lost after $k-1$ successful transmissions. This

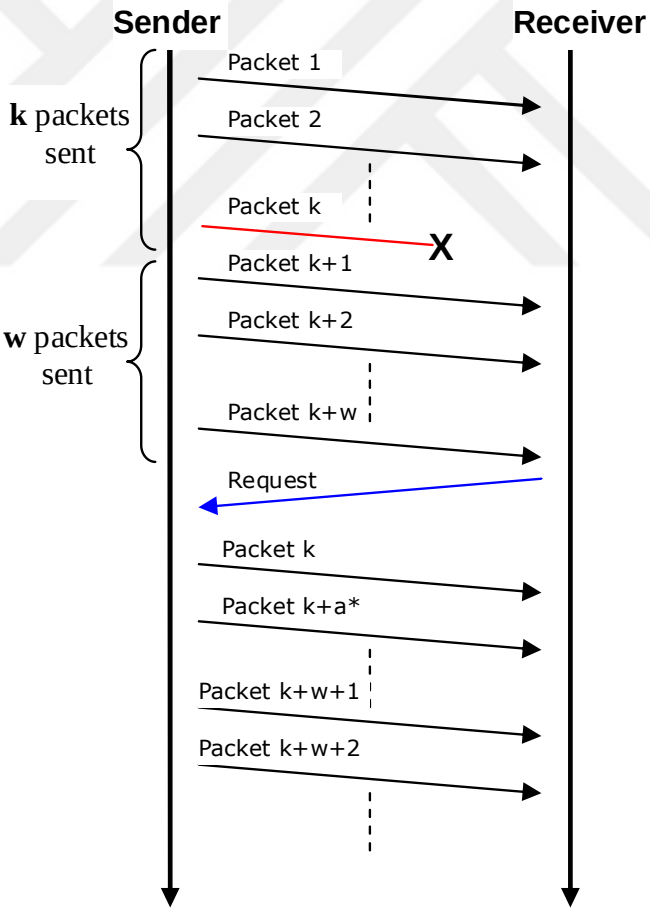

Figure 5.1. One-to-One transmission (a* is the index of lost packets within w)

is the case of *Geometric Distribution*. The *Expected value* of the random variable k is;

$$E[k] = \sum_{i=1}^{\infty} ip(1-p)^{i-1} = \frac{1}{p} \tag{5.1}$$

Until the first request is made, at least $E[k] + W$ packets will be sent. And the channel characteristics will apply on these packets, too. Of those packets, some of them are expected to be lost. The number of lost packets is defined as a random variable L and it is expected to be:

$$\begin{aligned} E[L] &= p(E[k] + W) \\ &= p\left(\frac{1}{p} + W\right) \\ &= 1 + pW \end{aligned} \tag{5.2}$$

This is also the number of the packets that need to be retransmitted until the sender switches back to transmit state. If those were not lost, for the next retransmission, there would be

$$E[k] + W = \frac{1}{p} + W$$

packets less to send. But $1 + pw$ packets will needed to be resent. The random variable X_j is defined as the successfully transmitted packets in j^th repeat of transmit sate. In total, remaining packets will be X_j less. Expected value of X_j is;

$$E[X_j] = \left(\left(\frac{1}{p} + W\right) - (1 + pW)\right) \tag{5.3}$$

This value is called α.

Each time a request is sent while the actual data is being transmitted, sender changes behavior to collect requests and to resend missing data. So; consider this event can be considered to be periodic and ends at m^{th} repeat. For each repeat , the number of remaining

packets will be:

$$\begin{aligned}
\text{Repeat } 1: &\quad K \\
\text{Repeat } 2: &\quad K-\alpha \\
\text{Repeat } 3: &\quad K-2\alpha \\
&\quad \vdots \\
\text{Repeat } m: &\quad K-(m-1)\alpha = 0
\end{aligned}$$

To send all K packets, m repeats of retransmissions are required. m can be derived as:

$$\begin{aligned}
K-(m-1)\alpha = 0 \qquad & \text{then;} \\
m &= \frac{K}{\alpha}+1 \\
m &= \frac{K}{(\frac{1}{p}+W)-(1+pW)}+1 \\
m &= \frac{Kp}{(1-p)(1+pW)}+1
\end{aligned} \tag{5.4}$$

The request and notification packets causes an overhead (O) in addition to the retransmitted packets. O is bound to the number of retransmission repeats and the number of nodes in the session. For n receivers and one sender, $(m-1)\times(n+1)$ packets will be sent as requests and notifications. And for all overhead packets, loss probability of p will still apply due to the channel characteristics. Hence, expected value of overhead becomes:

$$\begin{aligned}
E[O] &= (1-p)\times(m-1)\times(n+1) \\
&= (1-p)\times\left(\frac{Kp}{(1-p)(1+pW)}+1-1\right)\times(n+1) \\
&= \frac{Kp(n+1)}{1+pW}
\end{aligned} \tag{5.5}$$

Here, there are $m-1$ number of lost packet retransmissions are made because there

are no packets lost in the m^{th} repeat in order to complete the session.

Apart from those retransmitted packets, there are still K packets that need to be sent. So, using (5.2) and (5.4) the random variable N can be defined as the total number of sent packets and $E[N]$ can be derived as follows:

$$E[N] = K + (m-1) \times E[L] + E[O]$$
$$E[N] = K + \left(\frac{Kp}{(1-p)(1+pW)} + 1 - 1 \right) \times (1+pW) + \left(\frac{Kp(n+1)}{1+pW} \right)$$

$$= \frac{K}{(1-p)} + \frac{Kp(n+1)}{1+pW} \tag{5.6}$$

5.2. Single-hop One-to-n Analysis

The sender does not care about the number of the receivers. It only reacts on a request. So, in a one-to-n neighborhood, all the receivers can be considered as a single major receiver, which has a different probability of packet loss. The behavior of the algorithm in this case is illustrated in Figure 5.2 To extend the above analysis to one-to-n transmission, the probability of the very first packet which gets lost must be considered. Each of n receivers can lose a packet with a probability of p. Although a single trial is taken for sending one packet, the probability of it's being lost for different receivers will be independent. So, the probability of one packet to get lost when it is sent to n receivers must be calculated. To consider a packet as lost, having one node losing it is sufficient. The case of having *at least* one node that loses a packet is the complement case of having all nodes receive a packet successfully. The probability of one packet to get successfully transmitted to all the nodes in the network is $(1-p)^n$; hence the probability of having at least one node which loses that packet is:

$$p' = 1 - (1-p)^n \tag{5.7}$$

So, p' is the probability of packet loss in *one-to-n* scheme.

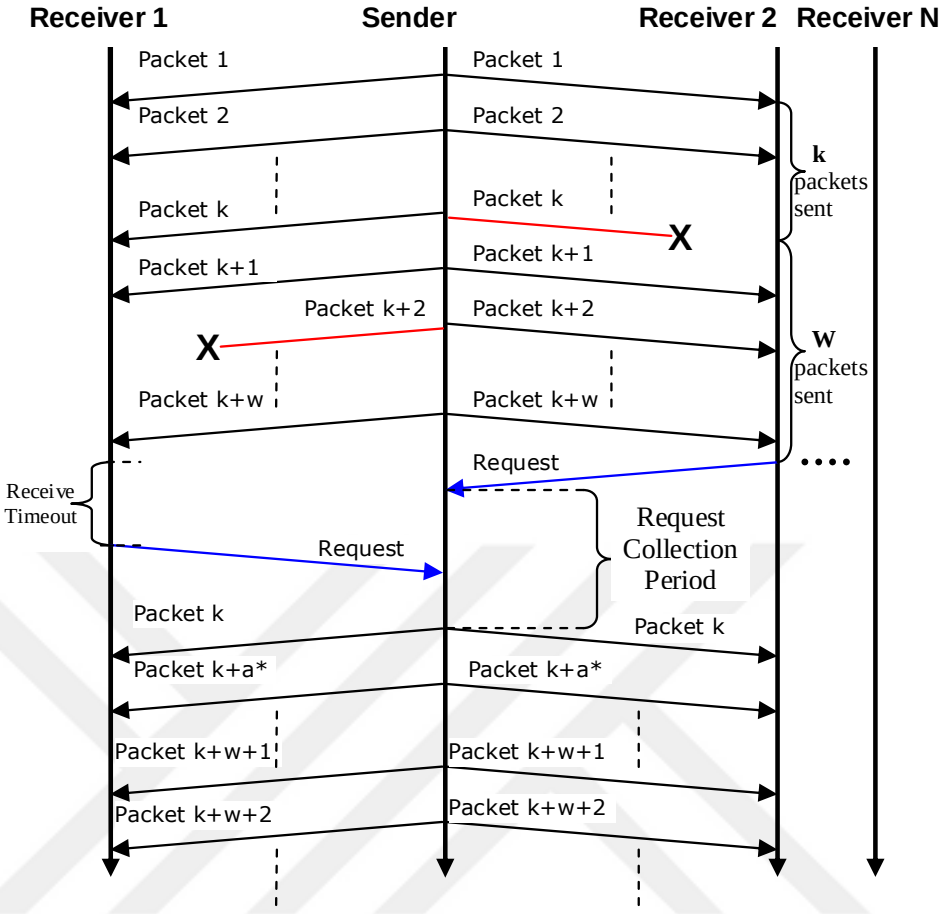

Figure 5.2. One-to-n transmission (a* is the index of lost packets within w)

In order to extend the one-to-one analysis to one-to-n, replacing p by p' is sufficient. A random variable, N', is defined as the total number of sent packets, including initial K packets. So, $E[N']$ is;

$$E[N'] = \frac{K}{(1-p')} + \frac{Kp'(n+1)}{1+p'W} \tag{5.8}$$

5.3. Multi-hop One-to-n analysis

Most wireless sensor network applications are designed in multihop topologies and the transmissions will travel several neighborhoods away from the sink. In Figure 5.3, sessions are shown by s_1 to s_4. The dotted circles represent neighborhoods, which have a separate session within and small circles represent the nodes. The dark grey node is the sink node

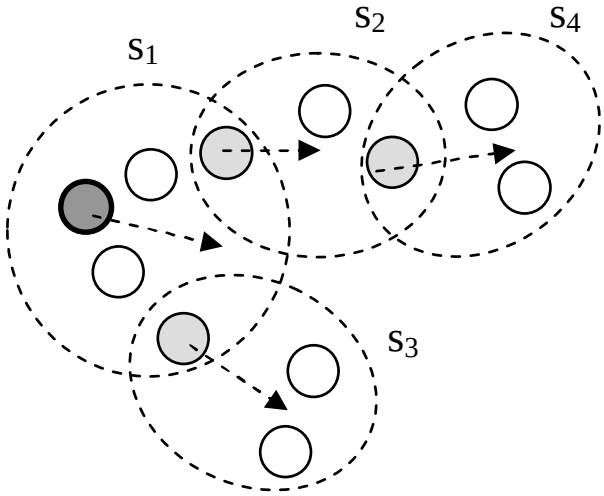

Figure 5.3. Multi-hop one-to-n topology

which initiates the first session. Light gray nodes are the senders of their neighborhood and the arrows show the direction of the data transmission. Notice, sessions s_2 and s_3 are expected to occur simultaneously. Each session will have its own retransmission states and they are less likely to intersect since the nodes chose the first data sending node as their senders and direct their requests to that node only. In such a topology, the number of total sent packets are independent for each session. That is, the sum of s_1, s_2, s_3 and s_4.

For simplicity, each session can be considered having the same number of receivers(n) with a constant packet loss rate of p. Since each packet transmission is independent from another, for constant size of data buffer the same expected value of total sent packets will be found. In such a multi-hop topology, the total number of sent packets (random variable N'') is given in Equation 5.9 where s is the number of sessions.

$$E[N''] = \sum_{i=1}^{s} E[N'] = \sum_{i=1}^{s} \left(\frac{K}{(1-p')} + \frac{Kp'(n+1)}{1+p'W} \right) \tag{5.9}$$

5.4. Time Analysis

To analyze time in a multi-hop network, the time to complete a session is calculated. That is the time, when there are no receivers left in a neighborhood with missing packets in

its data buffer.

Both packet receive and transmit delays are considered as t_d time units and processing of incoming packets in a wireless sensor device takes negligibly short time. The transmission time is distinguished from the request collection time, which may take different time units. The notification and request packets are transmitted within the sender's request collection interval, which is defined as t_c. Since random variables are functions, the total number of program packets can be defined by the function $f_p(K, W)$ and number of retransmission repeats by $f_m(K, W)$. The time to transmit request and notification packets is not bound to their quantity because they all transmitted at once within a request collection time period of the sender. So, it is bound to the number of retransmission repeats, which is

$$f_m(K, W) - 1 = m - 1 \tag{5.10}$$

So, the random variable T is defined as the time to complete a one-to-n session. (defined by Equation5.4).

$$T = f_p(K, W)t_d + (f_m(K, W) - 1)t_c \tag{5.11}$$

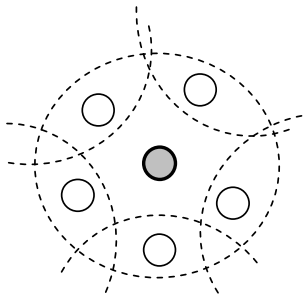

Figure 5.4. 1-hop neighborhood

In Figure 5.4, the gray colored node is the sender and the rest are receivers. The circles and the arches represent connectivities. For this topology, the expected value of time to complete is given in Equation 5.12, where p' is the packet loss probability in one-to-n scheme.

$$E[T] = t_d \left(\frac{K}{1-p'}) \right) + t_c \left(\frac{Kp'}{(1+p'W)(1-p')} \right) \tag{5.12}$$

For multi-hop time analysis, the time to start forwarding (which is bound to S) is important. The random variable T' is defined as the time to complete is the completion time of the session in the last hop neighborhood, plus that session's start time. In Figure 5.5, the dark gray colored node is the sink node which initiates the first session. Light gray colored nodes are the senders for next hops. The arrows coming out of nodes represent further sessions. In this topology, the start time of the right-most neighborhood is the sum of completion time of the last hop (which is Equation 5.11) and the previous two hops' session start times. This equation is given in Equation 5.13.

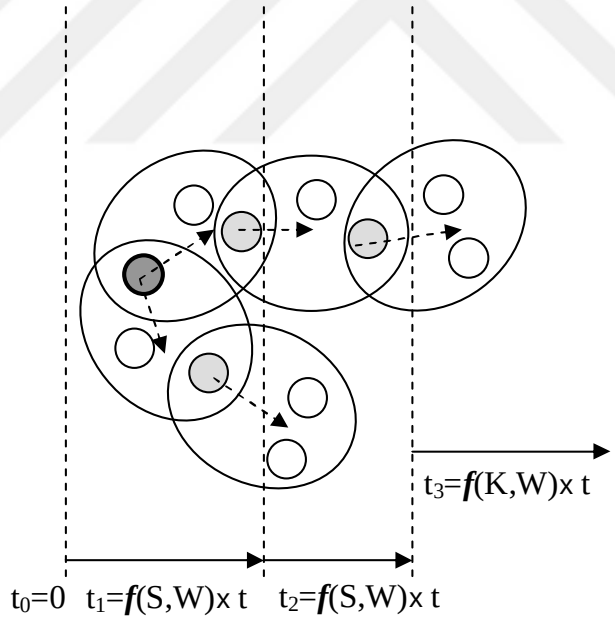

Figure 5.5. Timing diagram of 3-hop one-to-n transmission

$$\begin{aligned} T' =& f_p(S,W) \times t_d + (f_m(S,W) - 1) \times t_c \\ &+ f_p(S,W) \times t_d + (f_m(S,W) - 1) \times t_c \\ &+ f(K,W) \times t_d + (f_m(K,W) - 1) \times t_c \end{aligned} \tag{5.13}$$

The random variable T' is defined as the completion time of the session in the furthermost hop from the sink. Defining h as the maximum hop number in a topology and using constant S and W values, the total time required to transmit a buffer completely can be generalized as in Equation 5.14.

$$T' = \sum_{i=1}^{h-1}(f_p(S,W) \times t_d + (f_m(S,W) - 1) \times t_c) + f_p(K,W) \times t_d + (f_m(K,W) - 1) \times t_c \quad (5.14)$$

And hence, its expected value is;

$$E[T'] = \sum_{i=1}^{h-1}\left(\frac{S}{1-p'}t_d + \frac{Sp'}{(1+p'W)(1-p')}t_c\right) + \frac{K}{1-p'}t_d + \frac{Kp'}{(1+p'W)(1-p')}t_c$$

The following charts have been created by using the $E[N'']$ and $E[T']$ calculations with varying buffer size and hop counts. In calculations, the following values are used; $S = 30$, $W = 40$, $n = 10$, $p = 0.3$, $t_d = 20$ and $t_c = 20$.

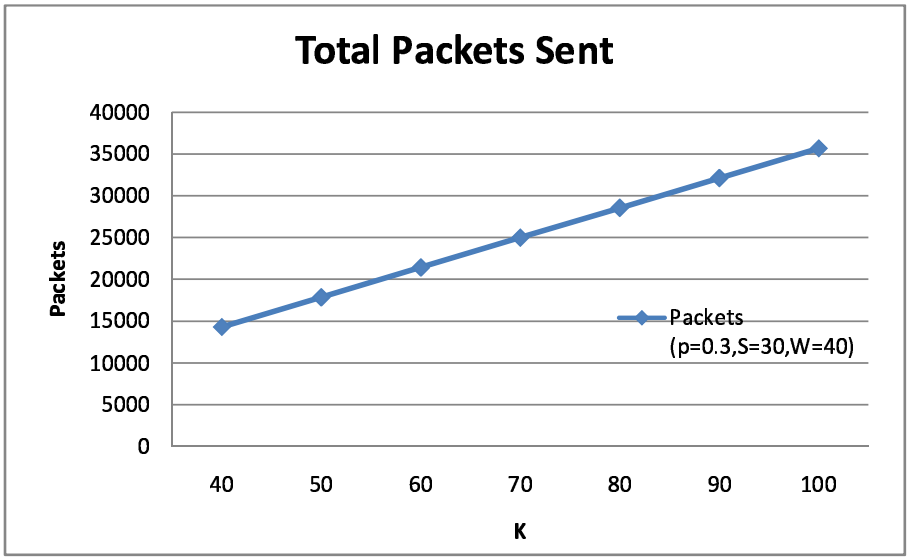

Figure 5.6. Effect of buffer size(K) to total transmissions in 10-hop topology

Figure 5.6 shows how buffer size affects the algorithm in a 10-hop ($= 10^{10} + 1$ nodes)

topology. It affects the total number of sent packets directly proportional. The effect of increasing number of neighborhood to the total number of transmissions is shown in Figure 5.7.

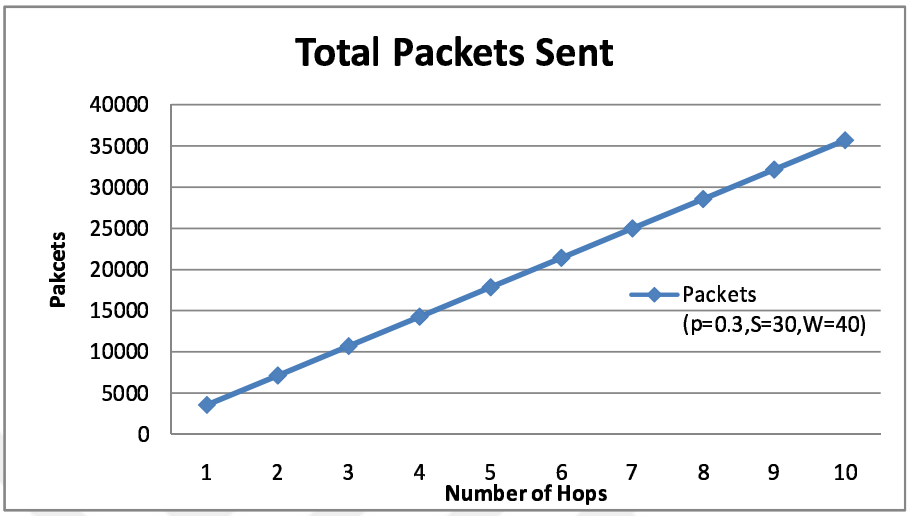

Figure 5.7. Effect of neighborhood-count(s) to total transmissions in 10-hop topology

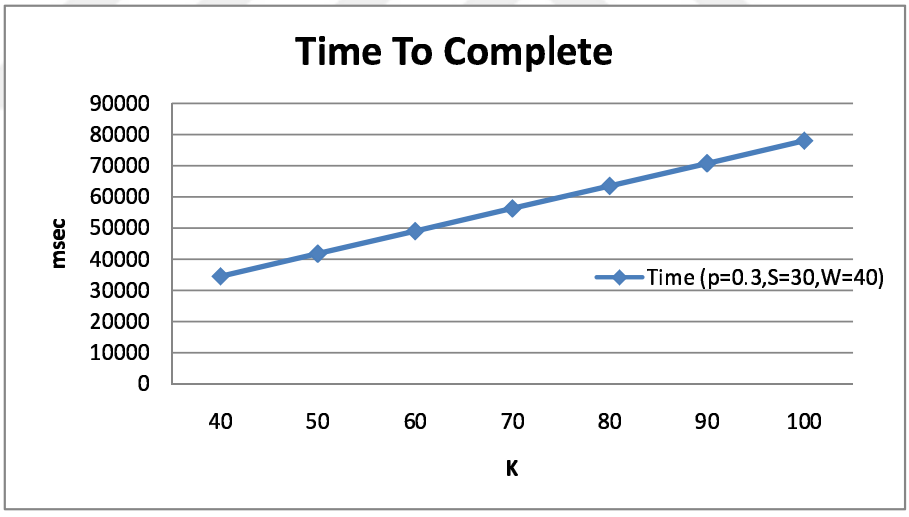

Figure 5.8. Effect of buffer size(K) to time in 10-hop topology

Time to complete increases linearly as the buffer to be transmitted is increased as shown in Figure 5.8.

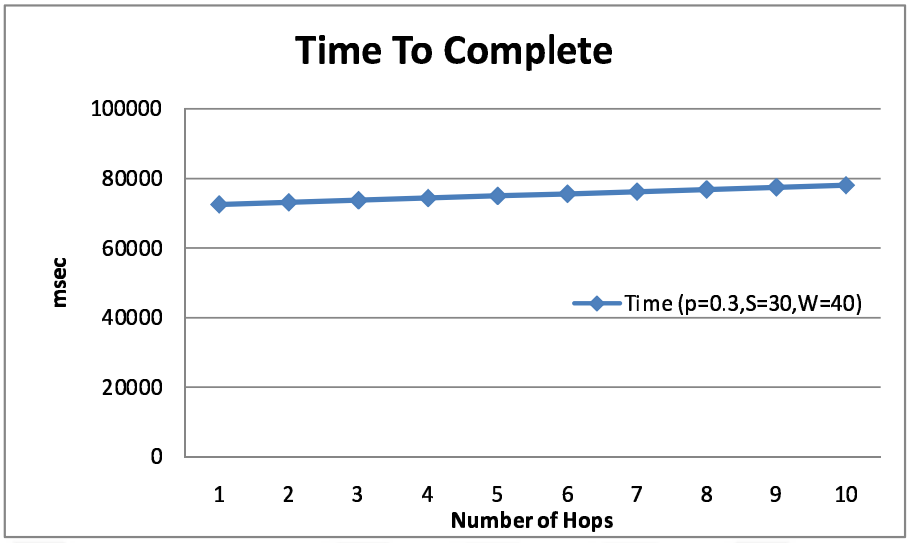

Figure 5.9. Effect of hop-count(h) to time in 10-hop topology

Figure 5.9 points to the strongest feature of ONE. As the number of hops increase linearly, the time to complete increases too. However, the increase in time to complete is much slower than that of hop-count.

6. SIMULATIONS AND EXPERIMENTS

In order to observe the behavior of ONE, first, a Monte Carlo simulation is developed. This simulation is used to see how do the window parameters affect the performance of ONE in a large scale 2-hop neighborhood. The observations taken from the Monte Carlo simulation has helped to choose the best W and S values for given buffer size, and later they have been used in OmNet++ tests. The OmNet++ simulator is used with Mobility Framework Extension which supports wireless channel models [10]. The implemented OmNet++ simulation has helped to observe the behavior of the algorithm under varying parameters and the results have been compared with the performances of PSFQ and SRM [11].

6.1. Simulation Environment: OmNet++

OMNeT++ is a discrete event simulation environment. Its primary application area is the simulation of communication networks, but because of its generic and flexible architecture, is successfully used in other areas like the simulation of complex IT systems, queueing networks or hardware architectures as well.

OMNeT++ provides a component architecture for models. Components (modules) are programmed in C++, then assembled into larger components and models using a high-level language (NED). Reusability of models comes for free. OMNeT++ has extensive GUI support, and due to its modular architecture, the simulation kernel (and models) can be embedded easily into your applications.

Although OMNeT++ is not a network simulator itself, it is currently gaining widespread popularity as a network simulation platform in the scientific community as well as in industrial settings, and building up a large user community.

6.1.1. Mobility Framework

The framework consists of an architecture for mobility support and dynamic connection management, a model of a mobile node in OMNeT++ and a specification of the internal structure. Also, a library of standard modules for the lower layers of the ISO/OSI protocol stack is provided. Mobility framework is suited for ad hoc networks as well as for access point-based networks. The design of Mobility Framework suffices the following items:

- **Usability:** Even the most highly performing and flexible software will not be deployed if the users struggle to put it to work. To keep the integration effort low, simple interfaces are created and a programming reference and design guide are provided. To enable rapid prototyping of higher layer protocols, standard modules for the lower layers of the protocol stack and exemplary network models are also created. Doxygen [18] is used for documentation of the code and design concepts and users are advised to do so with own extensions to the framework.
- **Extensibility:** No matter how thorough the design and how voluminous the implementation, there will always be problems where the final user will need to adapt the software to his own needs. Therefore, the Mobility Framework is designed so that new submodules can be easily derived from existing ones or created from scratch and integrated into the system just like the original ones.
- **Performance and Scalability:** Simulations of mobile networks often involve hundreds of nodes (sensor networks for instance) and millions of events to complete. A key design factor in the Mobility Framework is that execution time is reasonable and memory requirements do not grow faster than the simulated network.

In Figure 6.1, a snapshot of an ad hoc network simulation using the mobility framework is shown.

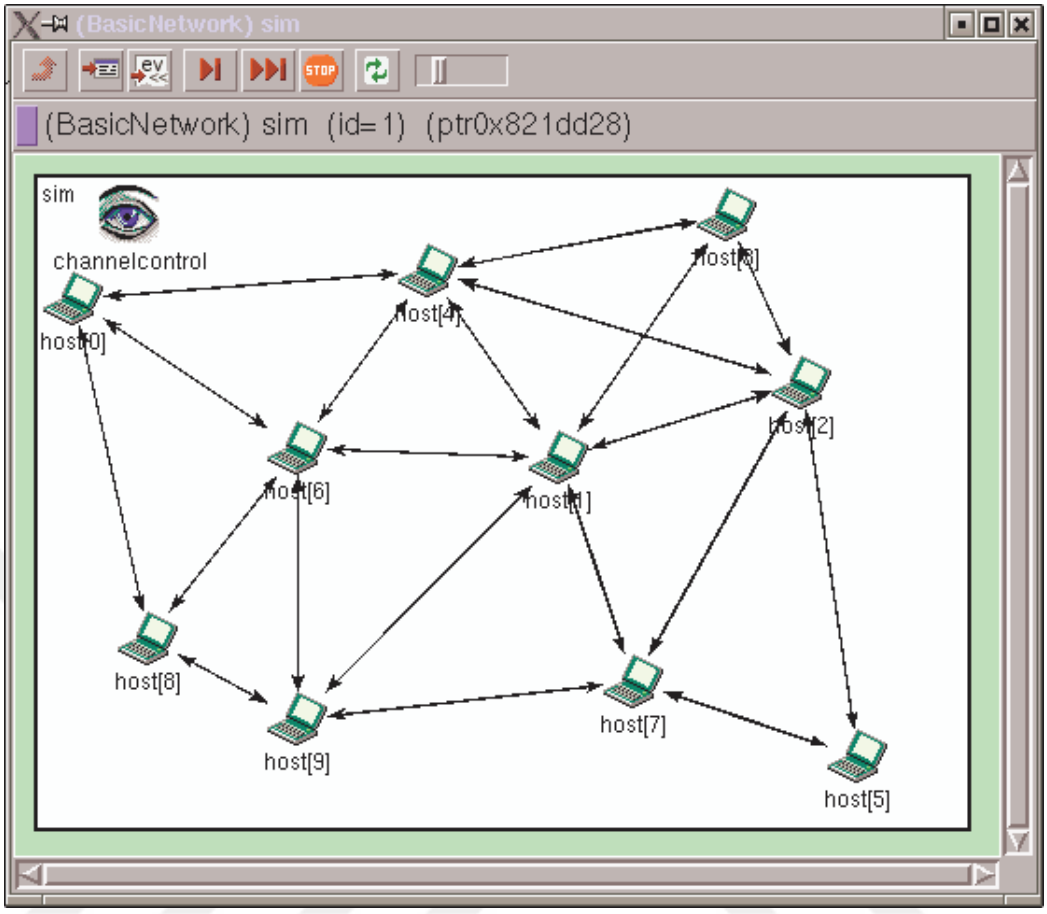

Figure 6.1. Snapshot of an ad hoc network simulation using the mobility framework

6.2. Methodology

The notations that are used for protocol parameters are;

- Buffer size (K)
- Node density (n) = Number of immediate receivers
- Packet loss probability (p)
- Retransmission window (W)
- Sub-buffer size to start next hop session (S)

Using the simulations, the impact of various parameters in ”time to complete” and ”total packets sent” a session i.e. sending the full buffer to all the nodes in a network of n nodes is studied. Later, the performance of **ONE** has been compared with that of PSFQ and SRM is compared in terms of ”average delivery ratio”.

6.3. Experiments and Results

Before testing the proposed algorithm on OmNet++, a Monte Carlo simulation is implemented to observe how window parameters (W and S) affect the performance of the algorithm for their various values. Using this Monte Carlo simulation, first, a two-hop network is built, where there is one sink and each node has 10 neighbors, which results in a 111 nodes network including the sink node. The topology diagram is shown in Figure 6.2. The dark grey node is the sink and initiates the first session for 100 packets data. The light grey nodes are the receivers of the first session and the senders of the second hop sessions. All second hop sessions occur simultaneously. The simulation was run with setting the protocol parameters to $p = 0.5, K = 100, n = 110$ and the results obtained are shown in the following graphs (Figure 6.3).

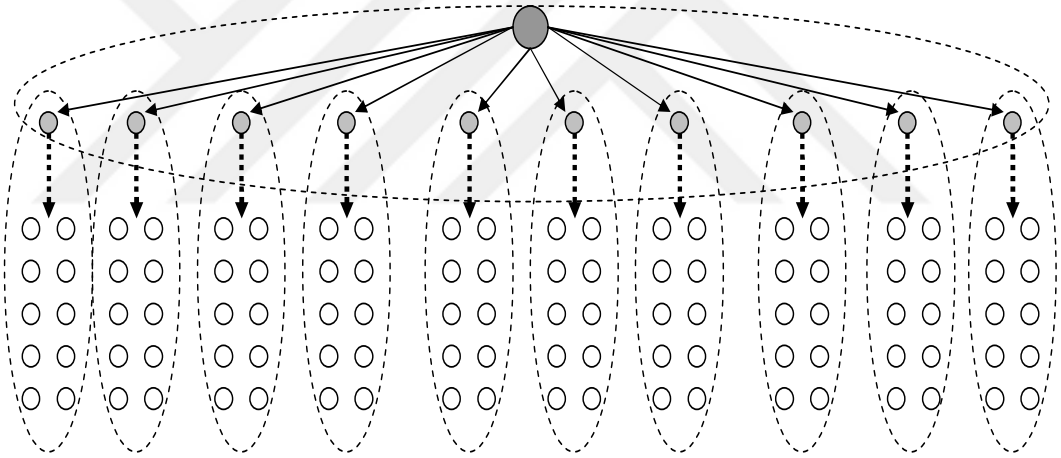

Figure 6.2. The sample test topology of 111 Nodes in a two-hop 1-to-10 network

Since each packet transmission is an independent event, the total number of data packets transmitted cannot be changed in a session , and in the whole topology. However, by defining the S and W parameters, the time to complete a multi-hop topology can change with a cost of overhead (which is the packets that are sent during the request collection state). The Figure 6.3 shows the behavior of the algorithm with varying S and W parameters. The Time chart of the figure shows that using a smaller S with a small W can decrease the time to complete, but it causes more overhead. S and W parameters need to be set experimentally by using the channel characteristics and the application requirements. For a time-critic application, smaller W must be set, while greater W should be used for

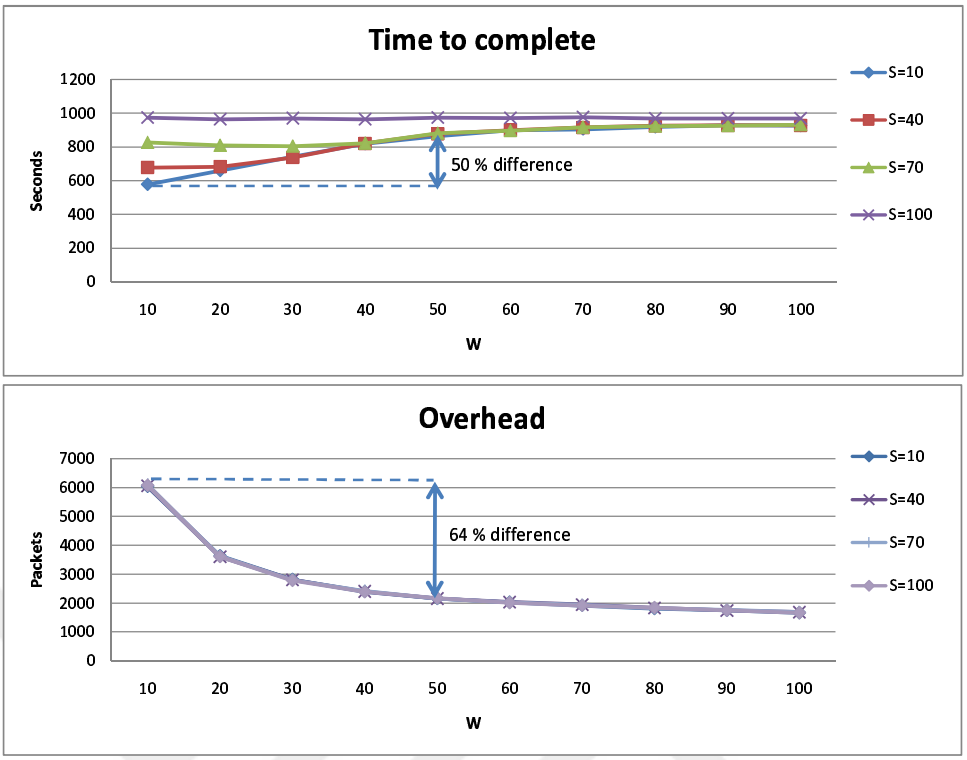

Figure 6.3. Effect of S and W over total sent packets and time to complete in a two hop 1-to-10 topology

energy-critic applications.

To understand the affect of channel characteristics on various topologies, the simulation is carried to OmNet++ environment. For sampling, S and W are set to $S = 30$ and $W = 40$ for the scalability tests (Figure 6.4). Different sizes of networks of 4, 16, 36, 64, 100 nodes with square grid topologies are constructed. For example, in 64 nodes topology, nodes are located on an 8x8 grid and the horizontally or vertically furthermost node is 8 hops away from the sink.

Figure 6.4 shows us that even when the number of nodes grows exponentially with increasing hop count, the time-to-complete grows logarithmically which is very beneficial. In these tests, the channel model of the simulation was used with the carrier Frequency of 868e+6 Hz, signal attenuation threshold of -110 dBm, path loss alpha of 3.5. The packets are transmitted with 100 bps bitrate and each bit has a loss probability of 5e-3. The transmission power is set to 1mW which is typical to wireless sensor nodes. In the tests, nodes are placed

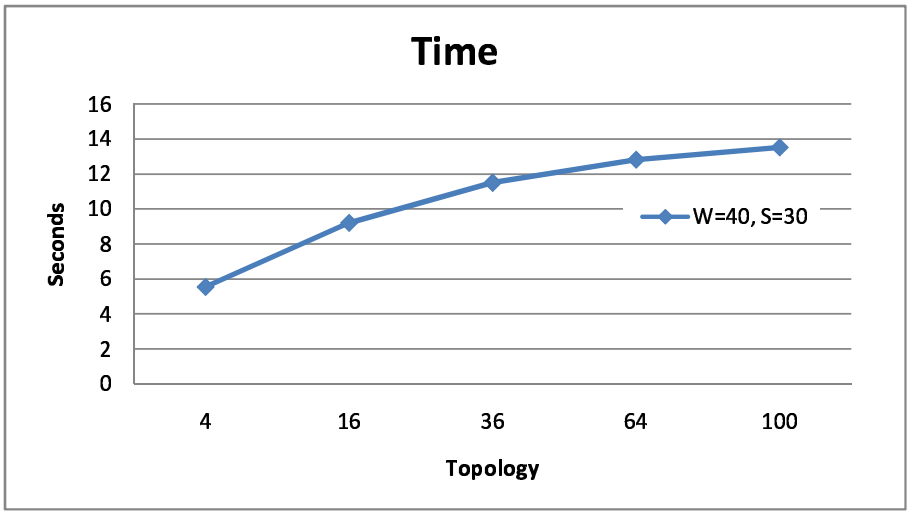

Figure 6.4. Time to complete under various network sizes.

so that each node can transmit to just one other node horizontally or vertically in each way.

Figure 6.5 and Figure 6.6 show how ONE behaves on different grid topologies with same network size. 6, 9, 12 and 18 hop grid topologies of 36 nodes are built. The results show that ONE is topology resilient, i.e. increasing hop count only slightly increases total transmissions and time to complete.

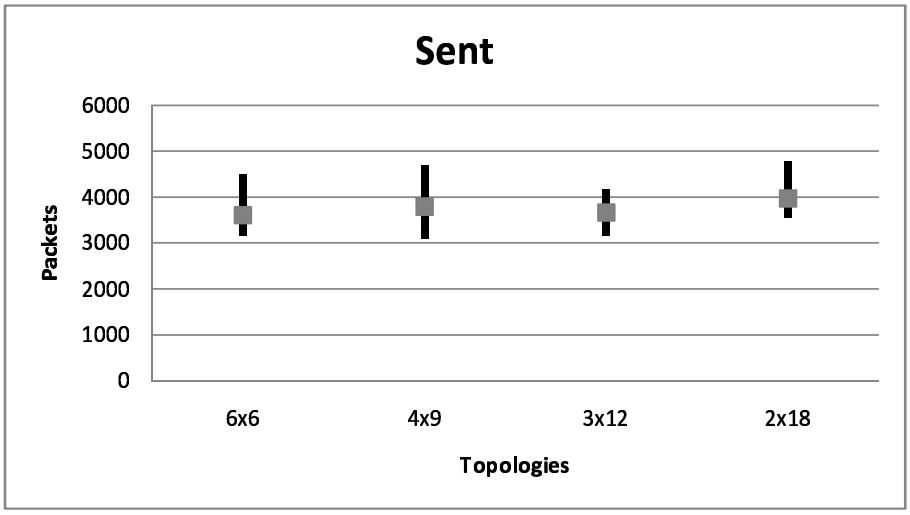

Figure 6.5. Total sent packets under various topologies of 36 nodes.

In order to see the scalability of ONE, single hop topologies of 16, 25, 36, 49, 64, 81 and 100 nodes were tried. The loss probability of the channel is set to 30%. The sink

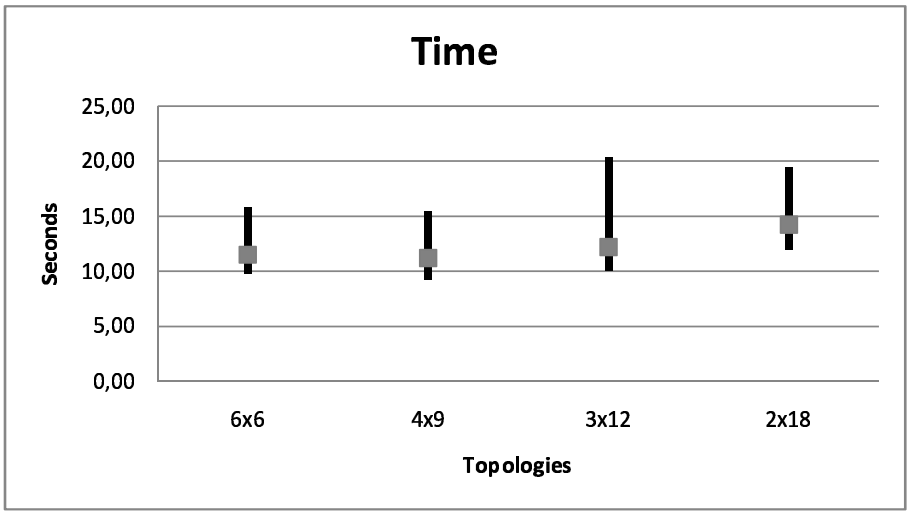

Figure 6.6. Time to complete under various topologies of 36 nodes.

nodes begun transmission of a 100 packets data buffer with 20ms intervals between each transmission.The results are shown in Figures 6.7 and 6.8.

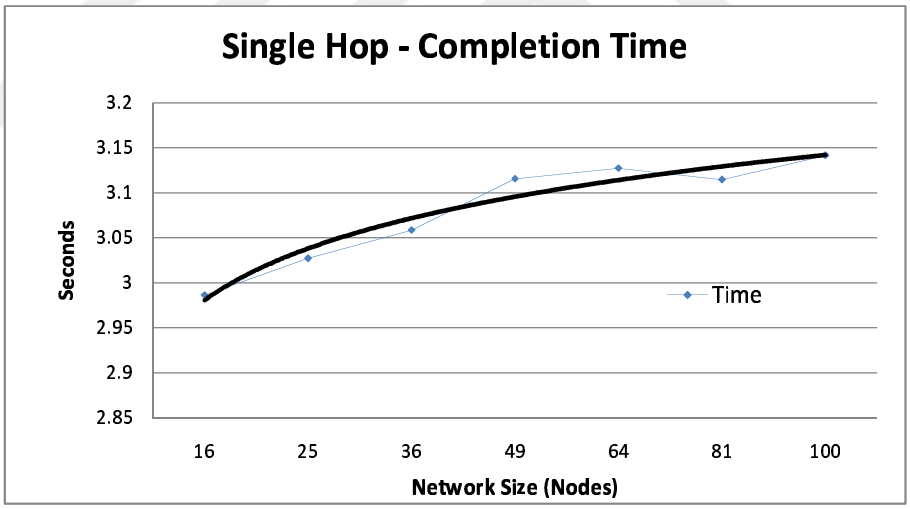

Figure 6.7. Scalability of ONE in single-hop topologies in terms of time to complete.

Still, although the number of nodes exponentially increase, the time to complete increases logarithmically with the increasing scales of node numbers. The trendline on the graph shows the tendency of the results.

In terms of total number of packets sent within the network, ONE is still beneficial, since it increases linearly.

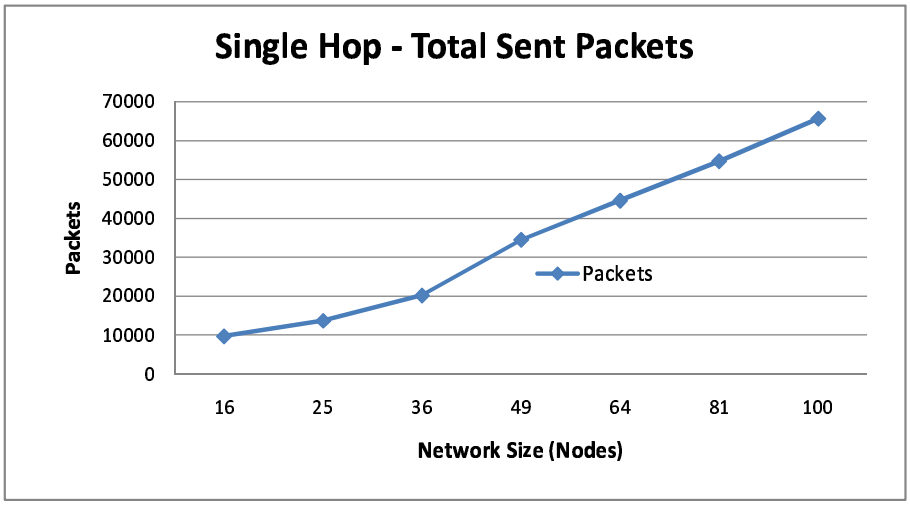

Figure 6.8. Scalability of ONE in single-hop topologies in terms of total packets sent.

To compare ONE with PSFQ and SRM, the same parameters in the simulation are set and the same tests were performed with exactly the exact same topology of 13 nodes. In Figure 6.9 the connectivity of the nodes in the topology is shown. The grey node (node 0) is the initiator of the transmission. The dashed circles represent sample transmission ranges. The algorithm terminates then all 13 nodes receive the buffer completely. The data buffer size is used 50 packets, as it was used in PSFQ. Differently, to be more realistic about wireless sensor networks, those packets are broadcasted in 20ms intervals, while PSFQ was experimented with 10ms intervals. That means, the injection speed of the packets is twice slower than that of PSFQ. Although ONE completes much earlier than both PSFQ and SRM, the simulation was run for 100 seconds, since it is the amount of time that those two algorithms were run for. Figure 6.10 shows that by the end of the simulation, all nodes could receive the packets with 100% success rate, while the others still have missing ones.

6.4. Summary

ONE defines two new parameters on buffer management; W and S. In order see to how do these parameters affect the transmission cost and the time to complete a session, a Monte Carlo simulation is implemented. This simulation is run in a two-hop 1-to-10 (111 nodes) topology. The results have shown that the time to complete can be reduced by setting these parameters to very low values. However, the gain from time has revealed the cost of total

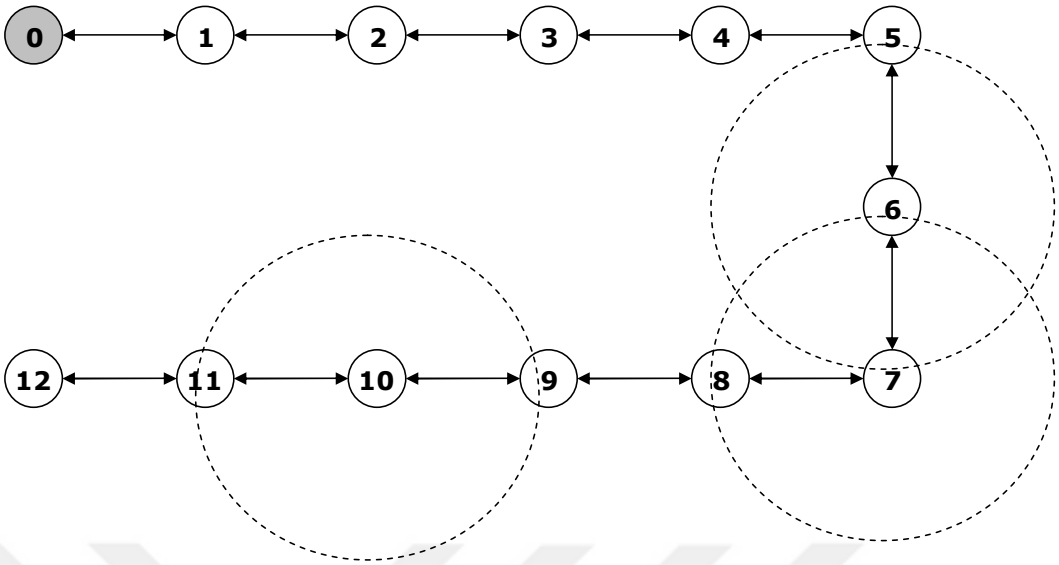

Figure 6.9. 12 hop topology of 13 nodes

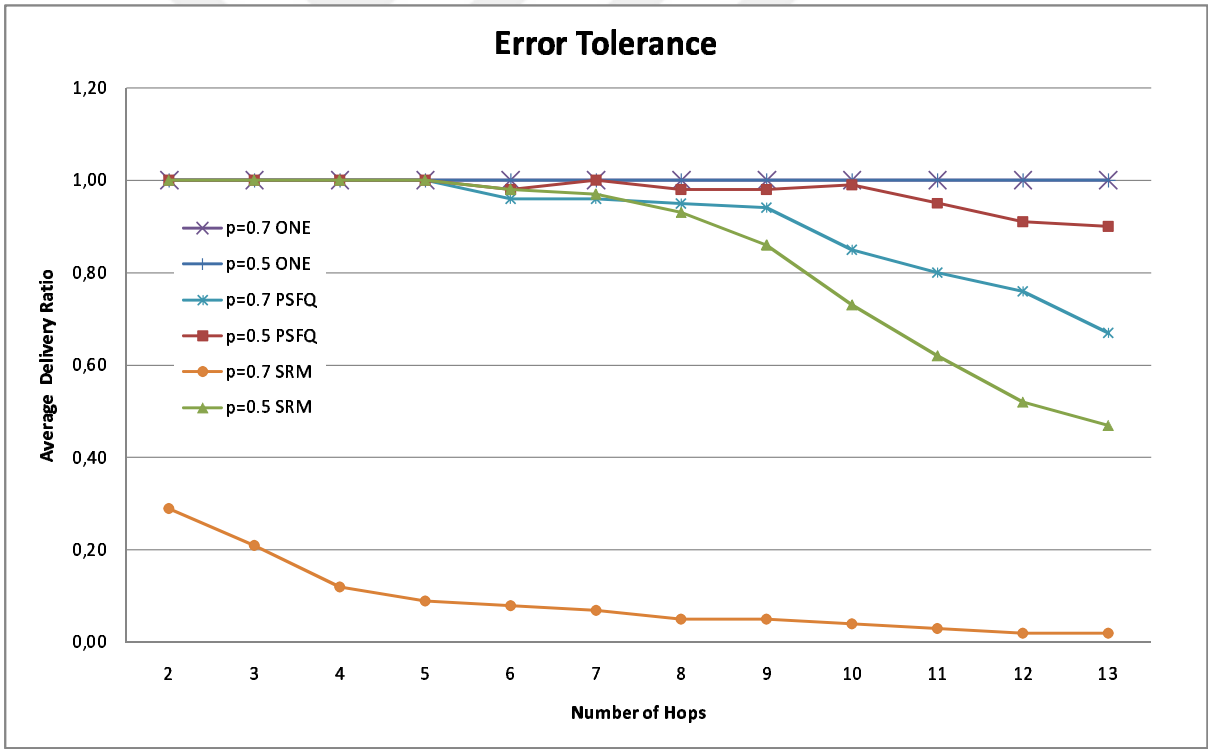

Figure 6.10. Comparison with PSFQ and SRM algorithms under different packet error rates.

packets sent into the network. The increase in the transmissions is caused by the overhead of the control packets (requests and notifications). The obtained results have also shown the need to careful selection of these parameters will be very beneficial depending on the type of the task of the sensor network, which can be either time-critical or energy-critical.

The results of the OmNet++ simulations have supported the results gained from the Monte Carlo simulation, by indicating the similar behavior of the algorithm. The simulations have shown the affect of the channel on the performance of ONE. For the last step, ONE has been compared to PSFQ and SRM. PSFQ was chosen since it is a representative work in the sink-to-sensors data transmission in WSNs. The results have revealed that ONE performs significantly better than those algorithms in terms of time, which was measured as the average delivery ratio.

7. CONCLUSIONS AND FUTURE WORK

Wireless sensor networks typically consist of a large number of nodes cooperating together and/or with other devices in order to accomplish a common task. The nodes within these networks are typically very resource poor devices.

Such devices require careful programming in order to meet their resource constraints, especially battery limitations. The algorithms running on them must be designed to perform the operations as fast as possible while consuming the minimum level of energy.

In wireless sensor networks, nodes may often require reprogramming or retasking. Both empirical studies and simulations show that, many of the data in a mass transmission get lost due to channel characteristics. These losses are bound to the quality of the hardware used on them. Since high cost and high capacity radio interfaces cannot be used on most of them, this problem must be handled with the effective software protocols.

The proposed error recovery scheme provides reliable data transfer in low capable wireless sensor networks. It uses the application level buffer directly by keeping bit arrays to store send and receive position indexes which gives a cross-layer behavior and hence it is very compatible with any type of high level application.

The performance of the scheme for one-to-n communication in single-hop and multi-hop clusters have been examined in terms of time to complete, total packets sent and average delivery ratio. Two new parameters, S and W, are introduced which effectively reduce the time to complete the whole bulk transmission while keeping the total number of transmissions as low as possible. These window parameters are meant to be set depending on the channel characteristics and application requirements, hence ONE becomes an adaptive solution. The results of the experiments have revealed that a significant amount of decrease in the time to complete in comparison to PSFQ and SRM is gained. It is also shown that ONE is topology resilient and highly scalable.

As for the future work, the algorithm will be improved in order to have a cooperation among the receivers. The receiver algorithm will be modified so that they will detect the lost packets of their neighbors by looking at their requests. If a receiver neighbor have received that packet already, it will transmit that packet to that specific node. This improvement will help distributing the energy consumption more fairly within the network. Also, since the transmission will not be interrupted at the sender's side, the algorithm will complete faster than it is at the moment. The cooperation improvement in the algorithm will bring the need of extending the analytical model.

In the analysis of the algorithm, the packet loss probability (p) in the channel is considered to have the same affect in all neighborhoods and for all nodes. In real environments, this consideration will not be very realistic. Different neighborhoods, and even different nodes in a neighborhood will have different packet loss probability from one another node. This fact rises up the need of self-configuration of the window parameters W and S. The algorithm will be extended in order to let the nodes calculate their loss ratios, and they will be able to chose their own window parameter values depending on the predefined network task constraints (i.e. running a time-critical application). This improvement will also bring the synchronization problem, which needs to be solved, since the requests are made using the W parameter.

REFERENCES

1. Chlamtac, I., Petrioli, C., and Redi, J. Energy-conserving go-back-N ARQ protocols for wireless data networks. In Proc. of IEEE International Conference on Universal Personal Communication (ICUPC'98), Oct. 1998.

2. S. Floyd, V. Jacobson, C. Liu, S. Macanne and L. Zhang. A Reliable Multicast Framework for Lightweight Session and Application Layer Framing. IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 784-803, Dec. 1997.

3. H. Karl, A. Willig, Protocols and Architectures for Wireless Sensor Networks, Wiley, ISBN-13 978-0-470-09510-2, 2006.

4. William Stallings, Data and Computer Communications, Prentice Hall, Seventh Edition, ISBN 0-13-183311-1, 1994.

5. S. S. Kulkarni, L. Wang. MNP: Multihop Network Reprogramming Service for Sensor Networks. Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICSCS'05), 2005.

6. 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, Sept 2002.

7. Tann, F., and Heidemann, J. RMST: *Reliable data transport in sensor networks*. In Proceedings of the First International Workshop on Sensor Net Protocols and Applications (Anchorage, Alaska, USA, April 2003), IEEE, pp. 102112.

8. C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: *A scalable and Robust Communication Paradigm for Sensor Networks*. In Proceedings of ACM/IEEE International Conference on Mobile Computing and Networking, pages 56-67, Boston, MA, USA, August 2000.

9. Sankarasubramaniam, Y., Akan, O. B., and Akyildiz, I. F. ESRT: *Event-to-sink reliable transport in wireless sensor networks*. In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking and computing, Annapolis, Maryland, USA, June 2003, pp. 177188.

10. W. Drytkiewicz, S. Sroka, V. Handziski, A. Koepke, and H. Karl, ”A Mobility Framework for OMNeT++”, 3rd International OMNeT++ Workshop, at Budapest University of Technology and Economics, Department of Telecommunications Budapest, Hungary, January 2003.

11. S. Floyd, V. Jacobson, C. Liu, S. Macanne and L. Zhang. A Reliable Multicast Framework for Lightweight Session and Application Layer Framing. IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 784-803, Dec. 1997.

12. C. Intanagonwiwat, R. Govindan, and D. Estrin, ”Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks” In Proc. ACM MOBICOM ’00, Boston, Massachussetts, August 2002.

13. S. D Servetto, and G. Barrenechea, ”Constrained Random Walks on Random Graphs: Routing Algorithms for Large Scale Wireless Sensor Networks,” In Proc. WSNA 2002, September 2002, Atlanta, GA, USA.

14. Bhaskar Krishnamachari, Networking Wireless Sensors, Cambridge University Press, ISBN-13 978-0-521-83847-4, 2005.

15. C.S. Raghavendra, Krishna M. and Taieb Znati, Wireless Sensor Networks, Springer Science+Business Media Inc., ISBN 1-4020-7883-8, 2004.

16. Pontie, G.J and Kaiser, W.J., ”Wireless Integrated Network Sensors”, Communicatoins of the ACM, vol. 43, no. 5, pp.551-8, May 2000.

17. Rabaey, J. M., Ammer, M. J., L. da Silva Jr., J., Patel, D., and Roundy, S., ”PicoRadio Supports Ad Hoc Ultra-Low Power Wireless Networking”, IEEE Computer Magazine,

vol. 33, pp. 42-48, July 2000.

18. V. Heesch, D. 2004. Doxygen. http://www.doxygen.org.

19. http://www.xbow.com

20. http://www.moteiv.com

21. TinyOS: a Component-Based OS for the Networked Sensor Regime, http://webs.cs.berkeley.edu/tos/, Sep 2003.

22. Moore, Gordon E., ”Some Personal Perspectives on Research in the Semiconductor Industry,” in Rosenbloom, Richard S., and William J. Spencer (Eds.). Engines of Innovation (Boston: Harvard Business School Press), pp. 165-174, 1996.

23. P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A self-regulating algorithm for code maintenance and propagation in wireless sensor networks. In Proceedings of the First USENIX/ACM Symposium on Network Systems Design and Implementation (NSDI), 2004.

24. J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In Proceedings of the second International Conference on Embedded Networked Sensor Systems (SenSys 2004), Baltimore, Maryland, 2004.

25. T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical report, UCLA, 2003.

26. J. Kulik, W. R. Heinzelman and H. Balakrishnan,”Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks”, Wireless Networks, Vol.8(2-3), pp. 169 185, 2002.

27. Charles Perkins and Pravin Bhagwat, ”Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers”, ACM SIGCOMM’94 Conference on Communications Architectures, Protocols and Applications, 234 - 244, 1994