POWER-BENEFIT ANALYSIS OF ERASURE ENCODING WITH REDUNDANT ROUTING IN SENSOR NETWORKS

Roopa Vishwanathan, B.E.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2006

APPROVED:

Stephen R. Tate, Major Professor
Robert Akl, Committee Member
Ram Dantu, Committee Member
Krishna M. Kavi, Chair of the Department of Computer Science and Engineering
Oscar N. Garcia, Dean of the College of Engineering
Sandra L. Terrell, Dean of the Robert B. Toulouse School of Graduate Studies

Vishwanathan, Roopa. Power-benefit analysis of erasure encoding with redundant routing in sensor networks. Master of Science (Computer Science), December 2006, 66 pp., 5 tables, 28 figures, references, 57 titles.

One of the problems sensor networks face is adversaries corrupting nodes along the path to the base station. One way to reduce the effect of these attacks is multipath routing. This introduces some intrusion-tolerance in the network by way of redundancy but at the cost of a higher power consumption by the sensor nodes. Erasure coding can be applied to this scenario in which the base station can receive a subset of the total data sent and reconstruct the entire message packet at its end. This thesis uses two commonly used encodings and compares their performance with respect to power consumed for unencoded data in multipath routing. It is found that using encoding with multipath routing reduces the power consumption and at the same time enables the user to send reasonably large data sizes.

The experiments in this thesis were performed on the Tiny OS platform with the simulations done in TOSSIM and the power measurements were taken in PowerTOSSIM. They were performed on the simple radio model and the lossy radio model provided by Tiny OS. The lossy radio model was simulated with distances of 10 feet, 15 feet and 20 feet between nodes. It was found that by using erasure encoding, double or triple the data size can be sent at the same power consumption rate as unencoded data. All the experiments were performed with the radio set at a normal transmit power, and later a high transmit power.

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

CHAPTER 1. INTRODUCTION 1

1.1. Definition and Uses of Sensor Networks 1

1.2. Constraints in Sensor Networks 1

1.3. Security Issues in Sensor Networks 3

1.3.1. Security Primitives 3

1.3.2. Common Attacks and Defenses 4

1.4. Scope of the Thesis 6

1.5. Organization of the Thesis 8

CHAPTER 2. OVERVIEW OF SENSOR NETWORK SECURITY 9

2.1. Cryptography 9

2.1.1. Symmetric Key Ciphers 9

2.1.2. Public Key Cryptography 13

2.2. Secure Routing Protocols 14

2.3. Introduction to Encoding Schemes 19

2.4. Platform and Language Details 22

2.4.1. TinyOS 22

2.4.2. TOSSIM 24

2.4.3. PowerTOSSIM 25

CHAPTER 3. DESIGN OF THE EXPERIMENTS 27

3.1. Design of Test3Paths 28
3.2. Test4Paths 29
3.3. TestAgg 29
3.4. PowerTOSSIM Power States 30
3.5. Radio Models 31
CHAPTER 4. SIMULATION RESULTS 33
4.1. Simple Radio Model 34
4.1.1. Test3Paths 34
4.1.2. Test4Paths 37
4.2. Lossy Radio Model 39
4.2.1. Test3Paths 40
4.2.2. Test4Paths 43
4.3. Varying the Power of the Radio 46
4.3.1. Test3Paths 48
4.3.2. Test4Paths 50
4.4. Aggregation 52
CHAPTER 5. CONCLUSION 58
5.1. Summary of Results and Analysis 58
5.2. Future Work 60
BIBLIOGRAPHY 62

LIST OF TABLES

1.1 Comparison of encoding cost of different encoding schemes. 8

5.1 Power consumption of different encodings for different radio models. 59

5.2 Power consumption of different encodings for radio current set at high. 59

5.3 A comparison of energy consumed on a per-node basis for the 3 applications. 60

5.4 Power savings obtained by using erasure encoding for various simulations. 60

LIST OF FIGURES

3.1 Topology of Test3Paths. 29

3.2 Topology of Test4Paths. 30

3.3 Topology of TestAgg. 31

4.1 Test3Paths: Source node power consumption. 34

4.2 Test3Paths: Remaining nodes power consumption. 36

4.3 Test3Paths: Base station power consumption for 3 paths. 37

4.4 Test4Paths: Source node power consumption. 38

4.5 Test4Paths: Remaining nodes power consumption. 39

4.6 Test4Paths: Power consumed by the base station. 39

4.7 Test3Paths: Power consumption of source node with lossy distance = 20 feet. 41

4.8 Test3Paths: Power consumption of source node with lossy distance = 15 feet. 41

4.9 Test3Paths: Power consumption of source node with lossy distance = 10 feet. 42

4.10 Test3Paths: Power consumption of remaining nodes with lossy distance = 20 feet. 43

4.11 Test3Paths: Power consumption of base station. 44

4.12 Test4Paths: Power consumption of source node with lossy distance = 20 feet. 45

4.13 Test4Paths: Power consumption of source node with lossy distance = 15 feet. 45

4.14 Test4Paths: Power consumption of source node with lossy distance = 10 feet. 46

4.15 Test4Paths: Power consumption of remaining nodes with lossy distance = 20 feet. 47

4.16 Test4Paths: Power consumption of base station. 47

4.17 Test3Paths: Source node power consumption with radio set at high. 49

4.18 Test3Paths: Power consumed by remaining nodes with radio set at high. 49

4.19 Test3Paths: Base station power consumption for radio set at high. 50

4.20 Test4Paths: Power consumed by source node for radio set at high. 51

4.21 Test4Paths: Power consumed by remaining nodes with radio set at high. 51

4.22 Test4Paths: Power consumed by base station with radio set at high. 52

4.23 TestAgg: Power consumed by source node. 53

4.24 TestAgg: Power consumed by aggregating node. 54

4.25 TestAgg: Power consumed by base station. 55

4.26 TestAgg: Power consumed by the source node for radio set at high. 55

4.27 TestAgg: Power consumed by the remaining nodes for radio set at high. 56

4.28 TestAgg: Power consumed by the base stationfor radio set at high. 57

CHAPTER 1

INTRODUCTION

1.1. Definition and Uses of Sensor Networks

Sensor networks can consist of thousands of low-power, low-cost nodes deployed in either mobile or fixed locations to monitor activities. Among the many uses of sensor networks are environmental habitat monitoring (e.g., Great Duck Island deployment), traffic monitoring, military operations and geological applications. These nodes might be controlled by a processing center known as the base station. The base station can be a more powerful computing device such as a laptop or a gateway to another network. It can either disseminate data to the network or can extract data from it or both.

Individual sensor nodes consist of a central processing unit (CPU), sensing unit, a transceiver and a power unit. The transceiver unit connects the node to the network, the sensing unit takes the signals produced by the sensors and converts them to digital signals. The power unit keeps track of the total power consumed and remaining power. The processing unit might perform some in-node computataion of the data collected, extract the relevant parts and send that.

Routing in sensor networks provides a challenge due to the fact that adversaries can easily capture and corrupt nodes. Some authors suggest using multiple, disjoint paths to counter denial of service attacks. In this thesis, I examine these techniques and power usage characteristics, and explore encoding techniques that reduce power usage while retaining robustness.

1.2. Constraints in Sensor Networks

Sensor nodes typically have limited memory (4KB RAM), a low power radio and a 4MHz CPU. The device typically is powered by one or more batteries running at 3V which might vary across different vendors. These values are for the mica motes developed at UC Berkeley,

but since the nodes work in a resource constrained environment, power and energy usage must be optimized. Due to these constraints, the challenges faced by sensor networks become a lot different from those faced by ad-hoc wireless networks. This is because the design goals and routing architecture of sensor networks are somewhat different from ad-hoc wireless networks. Some of the differences as outlined in [1] are:

- Sensor networks typically are deployed with a much larger number of nodes than wireless networks.
- The sensor network topology can change much quicker than a wireless ad-hoc network.
- The communication model of sensor networks is either many-to-one (all sensor nodes to a base station), one-to-many (base station to sensor nodes), or localized communication wherein sensor nodes send messages to neighbors or broadcast a message to a group of neighbors. Sensor networks typically do not support any-to-any routing in the network.
- Neighboring nodes in sensor networks might have highly similar data or readings, hence to optimize bandwidth and power, aggregation of data is an important consideration. Thus there might be a need to establish trust relationships betweeen nodes that might not be necessary in ad-hoc networks.

Message routing is the most affected by these constraints. There are various kinds of routing that can take place in a network. Some of them are:

- Static vs. dynamic routing: In static routing, the routing table is precomputed and all nodes have to refer to it to find the next node on the path. In dynamic routing, each node decides the next node based on certain heuristics like hop count, signal strength, transmission capacity of that node. Dynamic routing might be a better option in some cases to optimize shared radio usage.
- Clustered vs. flat hierarchy: Some networks might have one or more levels of clustering wherein some nodes act as aggregators leading to a hierarchical network.

The aggregator nodes can then collect a threshold amount of data and perform in-node processing and relay the data to the base station. Hence the number of packets that get transmitted to the base station gets reduced. This is useful in networks where readings of two nodes are highy similar, e.g., temperature sensing networks, light sensing networks, etc.

- Stateful vs. stateless networks: In stateful networks, each node can remember its past routing decisions and can possibly base its new decisions on the past ones. In stateless networks, routing decisions are not remembered. Stateful protocols might be memory-intensive, but over a period of time might reduce the computational time (to make routing decisions) by a considerable extent.

In each of these routing topologies, the following problems can occur:

- Due to low power radio, messages may get dropped.
- Effects of noise and collision are more pronounced as the radio channel is shared.
- Problems seen in regular wireless networks like multipath, fading and signal attenuation also affect routing in sensor networks.

1.3. Security Issues in Sensor Networks

In this section, I look at some common attacks in sensor networks and ways to defend against some of them. I also take a look at how security primitives can be applied to sensor networks.

1.3.1. Security Primitives

The radio links in sensor networks are inherently insecure thus enabling adversaries to eavesdrop on transmissions, perform replay attacks and inject spurious data packets. Nodes can be captured too and used for malicious purposes. Also the attacker need not be a sensor node: it can be a laptop or a more powerful device. Providing all three traditional security primitives (confidentiality, integrity and availaibility) at the same time may not be possible in this scenario. To this end some assumptions have to be made. First, the base station will in most cases have to be assumed to be non-malicious. Aggregation points or clustering

nodes in the network if any will have to be trusted by other nodes. Confidentiality, as such, can be obtained but managing encryption results in a lot of overhead. Also, the key sizes have to be as small as possible. Integrity of the data in the network can be maintained with a message authentication code (MAC), and availability of the network can in most cases be acheived. In addition to this, there is the possibility of an insider node being corrupted by an adversary. In this scenario, it might not be possible to guarantee any of the three primitives, and the most that can be hoped for is graceful degradation of the network. In some networks, there might be hierarchical grouping of nodes where all nodes in one group report to one aggregator node. In such networks, one might have to think about protection of the aggregator node, secure admission of nodes in the group, intrusion detection, prevention of adversaries performing traffic analysis, etc.

1.3.2. Common Attacks and Defenses

Attacks can take place at any layer of the network stack. Some of them are as follows:

(i) Physical layer: An attacker can jam the frequencies at which some or all of the nodes are transmitting thus performing denial of service. One way to combat this attack is spread-spectrum communication or switching to lower duty cycle when the jamming occurs. Another defense includes transmitting at infra-red or optical wavelengths. Another kind of attack is physical tampering or interrogation of nodes. Defenses for this may include tamper-proof packaging or hiding of nodes.

(ii) Data link layer: In this an attacker can introduce collisions in a MAC packet, thus forcing the receiver to request retransmisisons. This might lead to exhaustion of resource at both ends. Using error correction codes might reduce the effect of this but not to a considerable extent and at the cost of a high overhead.

(iii) Network layer: There might be a lot of attacks in this layer. Some of the more common ones as outlined in [26] are:

- Spoofing routing information in the network: By doing this, an attacker can create a routing loop, attract traffic towards them, divert traffic away from the base station, etc.
- Sinkholes: In this, a compromised node can attract all traffic towards itself by advertising a high-quality route to the base station. By doing this, all traffic in the network is channeled to the adversary who can then suppress messages from other nodes or mount other forms of attacks. This can be a serious problem in many-to-one networks, and is hard to defend against. One way of defending against sinkholes is to contruct or use protocols where traffic is not routed to one centre, but rather topology is constructed on the fly by localized interactions between nodes, e.g., GPSR [27].
- Selective forwarding: In this the attacker puts itself on a data flow path and refuses to forward data from certain nodes in the network. It can also modify the data originating from a certain node and forward it. One way to reduce the effect of this is multipath routing where source nodes can forward data along multiple disjoint paths to the base station.
- Wormholes: This occurs when an attacker performs a replay attack of messages received in one part of the network in another part using an out-of-bound channel. This can be used along with sinkholes to great effect. This is very hard to defend against.
- HELLO floods and acknowledgement spoofing attacks: In some protocols nodes are required to authenticate or announce themselves to each other before starting communication. In acknowledgement spoofing, an attacker replays an overheard acknowledgement message to convince the node that it is the node's neighbour. In HELLO floods, an attacker broadcasts to every node in the network that it is its neighbour, thus creating a wormhole. One defense against

these attacks is to verify the bidirectionality of the link before starting any message communication.

- Sybil: In this the attacker assumes different identities to the other nodes in the network thus convincing every node that it is the node's neighbor. One defense is to force a node to only communicate with a set of neighbors (determined by the base station) with whom it shares keys.

(iv) Transport layer: This layer manages end-to-end communication. An adversary can perform the classic TCP-SYN style flood attack by sending many connection requests to a single node, thereby causing exhaustion of resources at that node's side. Also it can repeatedly forge connection request and connection status messages between two nodes, thus causing desynchronization between them. One way to prevent against this is to perform authentication of all packets exchanged. Also, client puzzles can be generated and verified by each server node to whom a connection request has been sent.

Erasure encoding is a technique used in multicast routing to send redundant data packets along different paths in the network so that the receiver, after having received a sufficient number of packets, can reconstruct the original data packets from a subset of the original packets. This is to ensure speedy downloads. This can be applied to sensor networks with many-to-one routing so that when nodes transmit data to the base station, they can send encoded packets along multiple paths so that the base station can reconstruct the data from a subset of the packets.

1.4. Scope of the Thesis

In this thesis I use the idea of multipath routing as used in INSENS [10] to redundantly route messages along different paths to the base station, thus reducing the effect of DoS attacks or selective forwarding/modification attacks. Instead of one message being duplicated across different paths, an attempt was made to explore encoding the message using erasure encoding and sending the message pieces along different paths. Thus if the base station

gets a sufficient number of messages out of the message pool, it can reconstruct the data at its end. If the network is sufficiently dense (there can be many disjoint paths between a node and the base station), the presence of adversaries along some paths might not affect the data reaching the base station. The usage of erasure encoding reduces the bandwidth consumption. In the original multipath scenario, the channel's bandwidth is used for sending the same data along different paths. In this setting, the channel bandwidth can be more efficiently used to send double or triple the data size depending on the kind of encoding used. Also the overall power consumed is reduced as the data size is halved (or even divided by three) when using erasure encoding. The different encoding schemes that were experimented with are:

- Encoding two different data packets and sending them along three different routes such that recovery of any two out of three messages will guarantee recovery of the complete data set - this is called 2-of-3 encoding.
- Encoding three different data packets and sending them along four different routes such that recovery of any three messages out of four will guarantee recovery of the complete data set - this is called 3-of-4 encoding.

A comparative analysis was done on the different radio transmit powers and different radio models and their effect on power consumption. The results presented are in terms of data size vs. overall power consumed and message count. For each different simulation, the results are presented for 3 sets of nodes: the source node, the base station and the remaining nodes (which perform packet forwarding).

In this thesis, there is a trade-off between power consumed and the resilience of the network. Less power has been used for a fixed data size as compared to unencoded data but at the cost of decreasing the network resilience. In INSENS [10], the data is duplicated along different paths which gives greater resilience, but consumes more power for the same data size as compared to the encoding scheme used in this thesis. One more kind of encoding which has not been experimented with is 1-of-2. In this, the same resilience as 2-of-3 can be

TABLE 1.1. Comparison of encoding cost of different encoding schemes.

Encoding	Resilience	Communication Cost
1-of-3	2	$3n$
1-of-2	1	$2n$
2-of-3	1	$1.5n$
1-of-4	3	$4n$
3-of-4	1	$1.33n$

acheived, but at a higher communication cost.

If n bytes of data need to be sent from the source to the base station, 1-of-3 would require $3{\cdot}n$ bytes of data sent into the network, 2-of-3 would require $1.5{\cdot}n$ bytes of data and 1-of-2 would require $2{\cdot}n$ bytes of data to be sent into the network. Table 1.1 shows the communication requirements of different encoding schemes for a data size of n bytes. Resilience means the number of disjoint paths an attacker can corrupt in the network while the complete data set still gets to the base station.

The erasure encoding technique used here is very simple. More complex codes like Reed-Solomon codes, Tornado codes, Online codes etc. can be adapted to the multipath scenario, and this is left to future work.

1.5. Organization of the Thesis

Chapter 2 presents a review of related work in sensor network security, usage of cryptography, some key distribution schemes and a view of TinyOS (the operating system designed at the University of California at Berkeley on which the application is based), nesC (the language in which it is written), TOSSIM (the TinyOS simulator), and finally a brief review of erasure encoding. Chapter 3 describes the design of the experimental application and design considerations. Chapter 4 presents the simulation and experimental results in PowerTOSSIM. Chapter 5 summarizes the work done and looks at future research possibilities.

CHAPTER 2

OVERVIEW OF SENSOR NETWORK SECURITY

2.1. Cryptography

In this chapter, I review prior work in sensor network security. This is mostly in the area of cryptography and secure routing. I also provide some background on erasure encoding schemes.

A major technique of combating security problems is seeing how to apply cryptography efficiently and then designing protocols which can use these algorithms (trying to minimize memory, bandwidth, and the computation required at the sensor nodes end). Public key cryptography is highly taxing on the limited memory and computation power of the sensor nodes, although there have been some public key schemes designed for use in lightweight sensor networks. Carman, Kruss and Matt [6] show that for an encryption key size of 1024 bits, the Rivest-Shamir-Adleman algorithm (RSA) consumes about 42 mJ whereas Advaned Encryption Standard (AES) consumes 0.104 mJ. Also, using digital signatures for authentication would be too expensive. Hence symmetric key ciphers have to be used.

2.1.1. Symmetric Key Ciphers

Symmetric key cryptography is done by key pre-distribution. Here again you cannot have a single shared key throughout, because if one node gets compromised, then the entire network gets compromised. Therefore techniques of pair-wise key-sharing between nodes have to be used. In a network of n nodes, you have to distribute $(n-1)$ keys to each node. Each node having pair-wise keys with others poses different problems: If nodes arrive dynamically, then it might be difficult assigning keys to them. Also, node capture necessitates re-keying of the nodes whom the captured node had shared keys with, and the capture event has to be broadcast throughout the network to stop other nodes from dealing with that node.

There might not be enough memory on chip to do all of this, so random key pre-distribution has been used to overcome these problems. Here you can generate and distribute a random subset of keys taken from a large pool of keys to each node before network deployment. Two nodes can then interact with each other to find out whether they have a shared key. If they do not, they can establish a path of nodes between them, with each node along the path sharing keys with each other. Eschenauer and Gligor [13] present such a scheme. They give a 3-step protocol:

(i) Key Distribution

(ii) Shared key discovery

(iii) Path establishment

Initially the nodes are loaded with some keys taken from a pool. This pool is generated offline to reduce the computation required at the sensor node's end. Some subset of these keys is drawn without replacement and loaded into the key ring of each sensor node. Then during the second phase, the network topology is established, i.e., the nodes interact with each other to see whether they have a shared key. One way of doing this would be to broadcast the key IDs. (There has to be a secure way to do this or else adversaries can spoof that node's key ring). In the third phase, the nodes that are not connected directly attempt to establish a indirect link between each other. This is because every node's key ring has some extra keys which it doesn't share with any other node, so they can be reserved for new connections. If a path of nodes sharing keys exists between two nodes who do not share a key, then the nodes can use that path to exchange a key that establishes a direct link. Also, when a key needs to be revoked, the revoke event is broadcast and all nodes who have that key in their rings remove that key.

The process can be viewed as constructing a random graph, the nodes being vertices and an edge exists between 2 nodes if there exists a link (direct or otherwise) between them. A random graph has a high probability of being connected if the degree of its nodes is high or at least above a certain threshold. Eschenauer and Gligor showed that in a random graph

of 10,000 nodes only 250 keys needed to be preloaded to each node's key ring so that the probability of the network being fully connected will be 0.9999.

Perrig and Song [7] extend this concept in three different ways. First, instead of two nodes having one pair-wise key, they can have q keys which are taken from a key pool S. They call this the q-composite technique. Unless two nodes have q keys in common, they cannot communicate with each other. This increases the resilience against node capture in the sense that an adversary needs to know all q keys to impersonate another node. But an adversary can get hold of a large subset of keys by breaking fewer nodes' key set. Hence the key pool has to be large enough to prevent this from happening. But if it is too large then the number of keys shared between two nodes might be less, thus increasing the number of pairs of nodes that might not have any connections with each other. Also if the key pool is small, the adversary can build up a critically large set of keys, so a tradeoff has to be achieved. Perrig and Song mathematically show that the smaller the number of captured nodes, the greater is the resilience of the network. So if a few nodes' key rings have been compromised, the attacker cannot use those keys to compromise other nodes. Hence it will have to attempt large-scale attacks in which a lot of nodes get compromised which are more expensive to mount and easier to detect. If a large-scale attack has been mounted though, the network becomes more vulnerable in the q-composite scheme than other schemes.

The second mechanism that they present is multi-path key reinforcement. This is basically a technique to improve the earlier random mechanism (not q-composite). After key establishment has taken place sometimes the keys on a node's key ring might be shared by other nodes. If two nodes A and B share one pair-wise key, say k, and if k resides in the key ring of some other node who got compromised, then the link between A and B is compromised. Multi-path key reinforcement presents a way in which, after key establishment, the key is immediately updated. A computes a new key (this might have to be done in-node) and sends it to B via multiple paths. Bits of the new key travel along multiple paths. Only if B gets all the bits, it will be able to reconstruct the entire key (done by XORing all the

bits received). Now if an adversary compromised some (but not all) nodes along the path, he will not be able to reconstruct the entire key. This is used in conjunction with the basic scheme and not the q-composite scheme. They have experimentally shown that in a network of 10,000 nodes the basic scheme with multi-path routing gives a boost to the performance of the basic scheme, but the q-composite scheme with multi-path routing has little effect.

The main reason for this happening is that there are certain weaknesses in both the schemes: the weakness of the q-composite mechanism is that if it has a small key pool size, then if an adversary is able to compromise a number of keys, he would have gotten a good number of keys from the key pool. The weakness of the multi-path scheme is that finding multiple (completely disjoint) paths between two nodes is a bit difficult if the nodes along the path all have different key rings. This would increase network overhead. So we would be forced to make the key pool size small. This reduces the overall effectiveness of combining both these techniques.

The third technique Perrig and Song present is that of random pairwise keys. In the previous two techniques, each node shared a pairwise key with some other node and could communicate. But there was no way a node could ascertain the identity of the node it was communicating with: More than one node can have the same key in its key ring. So if node A is communicating with node B, even node C can have the same key as B in its key ring, so A should be certain that he is communicating with B and not C. Even though C might have a pairwise key with A, it might have been compromised by malicious parties. They present a mode of authentication wherein instead of just the pairwise key, the ID of all the nodes that have it in their key ring is also loaded into the key-ring of each node. This technique has been shown to be best of all the three they developed.

Another scheme is presented by Ning and Liu [34] wherein a pool of bivariate polynomials are used as a key pool. Nodes which have shares in the same polynomial can communicate with each other. This is a 3-step protocol where first a pool of bivariate polynomials is generated, then key establishment takes place wherein nodes look for others who have shares

of the same polynomial as them. They do this by broadcasting the polynomial ID's. Lastly path key establishment take place, where sensors who don't have a direct link between them will try to find other nodes and will establish a path through these nodes.

These random schemes have high memory overhead and have a probabilistic model in which the probability of a random graph being highly connected increases the chances of a secure link existing between 2 nodes. Moreover, all these schemes assume that the network is dense. If the network is sparse, then it could result in a disconnected graph where some nodes do not have any links at all with some others. Hence they wouldn't be able to do key exchanges at all.

Perrig and Chan [43] give a scheme, PIKE, where two nodes who do not share a direct link can find some intermediary who shares a pair-wise key with both of them. They also provide for dynamic node arrival by deploying the network in such a way that nodes have sequential IDs. They assume that there exists a two-dimensional lookup table where all node IDs are stored. A matrix is developed where nodes (IDs) lying on a row or column share a pair-wise key with all others on the same row/column. If two nodes lying on different rows/columns want to find an intermediary, all they have to do is look for nodes that lie at the intersection of the two nodes IDs. Nodes lying on the same row/column do not need to perform any key establishment as they already share a key. They also extend this to three dimensions. They simulated this using the GPSR algorithm and a geographic hash table (GHT) was used to provide a list of node IDs.

2.1.2. Public Key Cryptography

Memory and processor constraints limit the use of public key cryptography in sensor networks. There have been some schemes designed though in which authentication is done. Watro et al. present TinyPK, where authentication and key agreement takes place between an external party and a sensor node [54]. The external party is typically a more powerful device like a laptop. Here the external party authenticates itself to a sensor node but the sensor node does not authenticate, so even if there is any malicious activity from the sensor

node's end, the external party has no option but to trust it. In this there is a certification authority (CA) somewhere in the picture and the sensor nodes are loaded with the public key of the CA. Now there is a challenge response protocol:

(i) The external party submits it own public key signed by the CA's private key.

(ii) The sensor node will then use the preloaded CA's public key to verify the external party's public key.

(iii) External party will send a nonce and a checksum signed with its private key which the sensor node will be able to decrypt using the public key extracted in the second step. If the nonce and the checksum are validated, then the external party has been authenticated.

(iv) Now the sensor node will send across a session key and the nonce which will be encrypted with the external party's public key stored in the sensor node. When the external party receives it, he will decrypt it using his private key. Thus a communication link can be established. This was implemented on Mica motes with 4 KB RAM and 1024-bit RSA took about 14.5 seconds.

Here the sensor node operates with the public keys of the CA and the external party. If it had used the private keys of both, it would have required more time and memory.

2.2. Secure Routing Protocols

There have been several routing protocols that have been optimized for use in sensor networks, including SPIN [44], TinySec [25] and Directed Diffusion [23]. Details of them are provided below:

1. SPIN: SPIN stands for Sensor Protocols for Information via Negotiations. There are two categories of SPIN. The first is SPIN1 and the other is SPIN2. SPIN1 is the basic protocol and SPIN2 makes optimum use of energy. SPIN1 has three steps:

- ADV: This is the first step where any node that has new information will advertise information about it to its neighbors. Here energy consumption is minimized by not forwarding the entire data but just a brief description of it: metadata.

- REQ: In this step, all the nodes that are interested in the metadata will respond to that node with a data request message. Even if the nodes are not interested, they will inform their immediate neighbors about the availability of the data. Thus, soon the entire network will get to know of the existence of the new data.
- DATA: In this last step, the originator node will give copies of the full data to all those nodes who sent a REQ request back.

SPIN2 performs the same steps but nodes conduct energy checks at regular intervals. If a node knows it will not be able to complete the entire protocol (after some other node gives it a DATA request) it will not even advertise the presence of new data that it has.

Perrig et al. [44] experimentally showed that SPIN outperforms traditional sensor network protocols like Flooding and Gossiping and disseminates 60% more data per unit energy than these protocols.

This protocol mainly focuses on energy efficiency, reducing the computation required at the sensor node's end (nodes do not require any information about network topology, only about their immediate neighbors) and reducing bandwidth consumption. SPIN by itself does not have any cryptographic operations, so an attacker might be able to forge the identities of nodes, send spurious data, perform denial of service (DoS) attacks, etc. SPIN is mainly targeted for the Link/MAC layer.

Perrig et al. [44] present two protocols which build on SPIN, viz. **SNEP** and μ**TESLA** which provide confidentiality and authentication.

(i) SNEP: It stands for Secure Network Encryption Protocol. This provides confidentiality and integrity of the data. In SNEP the nodes send encrypted messages to each other with a Message Authentication Code (MAC). This uses a block cipher (DES) in counter mode which has the property that it is resilient against known ciphertext attacks. Basically, both nodes share a counter which is used along with the encryption key, and a MAC is also sent which includes the value of the counter.

This is to prevent replay attacks. The following notation is used to describe communication between two parties A and B:

C is the counter to be incremented.

k_m is the MAC key.

k is the encryption key.

$\{Plaintext\}_{(k,C)}$ denotes the encryption of the plaintext with the encryption key, k and the counter C.

A -> B : $\{Plaintext\}_{(k,C)}$, MAC(k_m,C | $\{Plaintext\}_{(k,C)}$)

Here every node shares a master secret key with the base station, and k_m, the MAC key, is derived from that using a pseudo random function.

(ii) μTESLA: This stands for Micro Timed Efficient Streaming Loss Tolerant Authentication protocol. This provides authentication of nodes. The base station and nodes are loosely time synchronized with each other and initially the base station sends across a MAC on the packet to each node (which is computed using a secret key). The nodes can verify that the MAC has come from the base station (by verifying the MAC). After a particular period of time, the base station sends across the key used to compute the MAC. The keys are part of a key chain which is generated by a publically known one-way function. The sender selects the last key in the sequence and repeatedly applies the one-way function to compute all other keys. The keys in the chain are generated in a few time intervals of each other. They are generated using a one-way hash function (e.g., MD5), and they have the property that they cannot be computed backward, i.e., if you have key k_i, k_{i+1} can be computed by the function $k_i = F_{k_{i+1}}$

The symmetric algorithm used for key generation is RC5.

Hence every node need have only the first key in the sequence to verify the entire sequence.

2. TinySec: TinySec is a link layer encryption protocol which uses the symmetric cipher Skipjack. There are two modes provided: Authentication only and authentication with encryption (auth-encryption). In Authentication only, the nodes send packets with a MAC where the payload is not encrypted. In auth-encryption mode, the node sends a data packet and MAC with the data payload encrypted. At the other end, the receiver can validate the message using the MAC and decrypt it.

Here a block cipher in CBC mode was chosen in order to prevent (or reduce the effects) of the same Initialization vectors (IV's) being reused despite the fact that a stream cipher would have been a faster choice. If an IV is ever repeated for 2 plaintexts, then an adversary can gather knowledge about both the plaintexts. At the same time, generating an unlimited set of different IV's means they have to be really long which is too memory and bandwidth intensive for sensor nodes. With a block cipher, even if the same IV gets used twice, the effect will not be as bad as it is with a stream cipher. The mode to guarantee this is CBC which has a fairly good performance in the face of repeated IVs. This is further strengthened by encrypting the IV's. TinySec was shown to add at the most 10% overhead in terms of latency.

3. INSENS [10]: Intrusion Tolerant Routing in Wireless Sensor Networks (INSENS) is a routing protocol which uses redundant routing between the sensor nodes and the base station. In INSENS, there are 3 steps:

1. Route request: This is when the network has just been established and the base station sends out routing topology requests to all the nodes. The integrity of this is protected by a one-way-sequence number. All subsequent communication from the base station will use a one way hash function to compute the next number in the one way sequence. This is similar to μTESLA. In μTESLA, the base station sends out a MAC, here it sends out a one-way sequence number.

2. Feedback: In this step, on receipt of the routing request from the base station the nodes will send feedback on the same route back. Each node will send the information back

to the parent node which sent it the request information. The integrity of this is protected by a keyed MAC which includes neighbour information and the current one way sequence. In this way, the base station can verify and compare the topology information with all the other nodes' information, and even if an adversary manages to spoof a message he will get the neighbours list wrong.

3. The base station computes routing tables and forwards them to all the nodes.

This protocol uses the idea of one-way hash chains as used in μTESLA and keyed MAC's as used in SPINS. The major contribution of this protocol is that it incorporates redundancy in the routing paths: it constructs multiple paths between the base station and nodes. So if a malicious node resides on a path between a node and a base station, there are enough additional paths between them to ensure that the data reaches the destination safely. Of course, the network has to be dense enough to enable this.

Aggregation: One way of minimizing overhead is to make some nodes aggregators: They gather data from the surrounding nodes, process them and propagate the result to the base station. For example, in a temperature monitoring system, a node can gather data from all its neighbors, calculate their average value and send it to the base station. Some security issues here are that the sensor readings may be wrong due to the sensors getting compromised or the aggregators themselves getting compromised.

One way of doing this is presented in SIA [46]. Here the aggregators share a key with the sensor nodes, thus they can verify that the data came from the correct nodes. Then the aggregator uses the data to compute the aggregate value of it. It does this by constructing a binary hash tree keyed at the root using a collision resistant hash function and then transmits the root to the base station. The base station can the check whether the aggregated values are close to the actual data values by random sampling mechanisms and interactive proofs.

Deng et al. in [12] present a method to perform secure in-network processing where the base station delegates authority to certain nodes (aggregators) for some time to perform data gathering and processing. The tools for doing this (member nodes' keys, one way hash chains,

topology information, etc.) are all provided by the base station. The aggregator needs just to utilize this information, gather data, perform processing and deliver the information to the base station.

2.3. Introduction to Encoding Schemes

Encoding schemes are used in some peer-to-peer networks wherein one sender transmits data to one or multiple receivers who then upload their own content to the network. In this scenario, one has to make sure that adversaries do not choke up the bandwidth by inserting spurious data. Traditional forward error correction codes like Reed-Solomon codes require a fair amount of decoding time. Among all the encoding schemes available, erasure encoding has been found to be reliable and low-cost. In erasure encoding, the sender divides a file F into n blocks and then maps this set of n blocks onto a larger set of k blocks which then get transmitted. The receiver has to receive any sufficiently large subset of those k blocks to be able to reconstruct the entire data. There are two kinds of erasure encoding:

1. Erasure codes of rate k: In this a block of n blocks is mapped onto a larger set of n/k blocks where $0 < k < 1$.

2. Rateless erasure codes: In this, a set of n is mapped onto a set of blocks whose size is exponential in n. There are many classes of erasure codes including Tornado codes [5], Online codes [38], LT codes [35] and Raptor codes [52]. Details of them are provided below.

- Online codes: Online codes map a set of n message blocks onto a set of $n \cdot k$ auxiliary blocks where k is a small constant between 1 and 3, a constant, $\delta = 0.005$ and n is the number of message blocks. It has three stages: a precoder, an encoder and a decoder. The precoder adds a set of n message blocks onto a randomly chosen set of k auxiliary blocks. It then sums up the message blocks and auxiliary blocks as a complete file, F' whose size is $n(1+\delta k)$ which is given to the encoder. The encoder then specifies a probability distribution d_i for each check block i. It then outputs a pair $< x_i, c_i >$ where x_i is meta-data that describes which blocks were chosen and c_i is the sum of those blocks. The decoder maintains a pool of unrecovered

check blocks; whenever a message block is received, it subtracts it from the pool of unrecovered check blocks. Maymounkov and Mazieres [39] propose a scheme to implement Online codes for multi-source downloads wherein the each block has an ID.

Gkantsidis and Rodriguez [18] present a scheme wherein one can use network encoding, i.e., each node in the network can encode blocks and transmit them as and when they receive them. This scheme has been tested against different topologies like dynamic node (server node) departures and heterogenous client populations and has been shown to outperform other error correction techniques in all cases. Based on this, they implemented *Avalanche* a real time system that used network encoding. Luby and Mitzenmacher in [4] present a scheme where they use erasure codes to construct a *digital fountain*. The *digital fountain* injects a set of distinct encoded blocks into the network, if the receiver recives any subset of those encoded blocks equal in length to the source data, irrespective of the specific blocks received, it can construct the source data. They implement this by using a class of erasure codes known as *Tornado codes*.

- LT codes [35]: This class of codes also uses the *digital fountain* approach. In LT codes every encoded block has a degree associated with it chosen from a degree distribution. The degree distribution has to be evenly spread out over the set of encoding blocks. It then chooses a set of random neighbours of the encoding symbol same as the degree. When all the neighbours are XOR'd, we get the encoding symbol. A major consideration of this encoding scheme is that the degree distibution should be good. Input symbols are added as soon as they are processed and the redundancy in the encoding symbols is as low as possible, i.e., the number of encoding symbols covering the same input symbol should be minimum. LT presents

a distribution called **Solitron** distribution to this effect. It also discusses variations of this distribution called **Ideal Solitron** distribution and **Robust Solitron** distribution.

- Raptor Codes: Raptor codes [52] are a subclass of the digital fountain codes and are based on LT codes with the advantage that they are linear time codes. Just before an LT-code is applied to the input symbols, they are pre-coded with some redundant symbols. On the decoder's side, this means that only a *constant fraction* of the input symbols need to be recovered.
- Tornado codes [5]: Tornado codes are similar to Reed-Solomon codes in that they have a system of equations where the variables are the unreceived packets. An XOR operation is performed between the received packets to get the missing packet. The equations are of the form: $y_3 = x_1 \oplus x_4 \oplus x_7$

  If any three of the four unknowns in the above equation, are received, the fourth unknown can be reconstructed. When a packet arrives, it can result in solving for unknowns in a cascade of equations, thereby recovering all the source data packets. This is mainly geared toward multicast (one-to-many) downloads.

Problems with erasure codes: Erasure codes provide security against loss of packets and not against injection of spurious data into the network or corruption of packets. Security can be provided against this by having each packet have an index; the receiver then checks for duplicate index and drops both the packets. But this can get difficult for packets with large number of invalid indices. Karlof et al. [24] present a scheme in which they implement an improvement over erasure codes: distillation codes. In this the valid symbols are "distilled" from invalid ones. A set of valid symbols is constructed and each received symbol is tested for membership by way of *one-way accumulators*. The accumulator is a Merkle hash tree [40]. If the one-way accumulator is broken, then an adversary can pollute the network. Krohn et al. [28] give a scheme where each block of data to be transmitted is hashed and the receiver

need only obtain the hash value to verify the integrity of the check blocks as and when they arrive.

2.4. Platform and Language Details

The platform used for the experiments described in this thesis is TinyOS-1.1.15 [33], using programming language nesC [16] and simulated in TOSSIM [32]. I next provide an overview of these technologies.

2.4.1. TinyOS

TinyOS was developed at the University of California at Berkeley and is a popular operating system for sensor networks. It supports multiple platforms: mica, mica2, micaZ, rene and telos. TinyOS can execute only one program at a given time. It has two threads of execution: **Tasks** and **Hardware Event handlers**. Below is a brief discussion of them.

- Tasks: Tasks are functions which cannot be preempted by anything except by hardware event handlers. They are typically short and run to completion. Tasks are basically deferred functions. Once a program posts a task, it does not know when it is going to be executed. The TinyOS scheduler executes it when it is free. Hence tasks do not have return values; they return *void*. They also do not take any parameters. For example:

```
task void SendMessages(){
statement 1;........
statement 2;........
...}
```

  This is because tasks are executed within the scope of a module, so any variables passed to the task would go out of scope outside the module. A component posts a task to the scheduler by using the *post* keyword. This queues up the task in the task queue. A task can post itself too.

- Hardware event handlers: Hardware event handlers are executed in response to interrupts which can preempt tasks and each other. Commands and events that have to be executed as part of a Hardware Event Handler are declared with the **async** keyword. Tasks, that can be preempted are declared with the **sync** keyword. Hence there might be race conditions when a hardware event handler is being executed. These can be prevented using the **atomic** keyword. When this keyword is used, it enforces exclusive access to shared code/data. This is usually used with variables and/or small statements. If a possible race condition needs to be avoided or disregarded, there is a keyword **norace**. If a variable is declared with this keyword, then in case of a race condition involving that variable, the ncc compiler just ignores it. Tasks are by default **sync**. The posting of a task by a component is **async** however. Any command that calls an **async** command or event has to be **async** itself. For example:

```
interface Leds{
 async command void LedsOn();
 async command void LedsToggle()
 ....}
```

TinyOS has various components that control different node activities like sensing, EEPROM, ADC, radio, etc. The radio stack used is the mica2 stack.

The TinyOS networking model is made up of 3 layers: *Physical layer* which sends the bits over the radio, *Link Layer* which employs a BMAC protocol, *Transport Layer* which controls actual communication between nodes: by swapping and routing buffers to and fro between nodes. This layer performs a buffer swap with the lower layers. The link layer has Active Message ID's (up to 256) for each message. This is like a port number associated with each message.

2.4.2. TOSSIM

TOSSIM is a discrete event simulator for TinyOS networks. The same code that is compiled for actual motes can be compiled for TOSSIM by the change of a compiler option. In TOSSIM, each simulated node boots up at a random time avoiding synchronization with other nodes. It simulates TinyOS network stack at bit-level granularity. TinyOS follows a radio model wherein the sender transmits a start symbol at a rate of 10Kbps, the receiver listens at a rate of 20Kbps and then the actual data is transmitted at a rate of 40Kbps. TOSSIM simulates this radio model by sampling the bit-rates at regular intervals and if there is a change in the bit-rate, changes the period between radio clock events. TinyOS provides two radio models: the **simple** radio model and the **lossy** radio model. The **lossy** model is based on empirical data collected from actual sensor node deployment. In the actual sensor deployment, packet loss rate was measured, a probability disribution was determined for the packet loss rate over distance, and this was mapped onto independant bit-error rates. TOSSIM can simulate both the simple radio model and the lossy model. In the simple model, all nodes are placed in a cell and each node can hear each other perfectly with no transmission errors, and in the lossy model we can specify the packet loss rate between two nodes in terms of probability of the packet being lost or bit being flipped, for bit-level and packet-level simulations respectively. With TOSSIM, it is possible to examine the output with different modes at run time, e.g. radio, route, cyclic redundancy check (CRC) bits, active message (AM) packets, etc. TOSSIM takes in one required parameter, the number of nodes to simulate, along with optional debug options. It has an user interface called **TinyViz** which allows the user to control and interact with the simulation as it runs providing capabilities like setting breakpoints, packet injection, manipulating radio links, position of nodes, etc. TinyViz provides plugins for doing all of this, and the user can write her own plugins as well.

The only difference between TinyOS and TOSSIM is that in TOSSIM tasks are not preempted by hardware interrupt handlers. This is beacuse TOSSIM is a discrete event simulator.

Applications can be compiled to run under TOSSIM by compiling with **make pc**, using a standard TinyOS make file which generates an executable located in build/pc/main.exe. This can then be run using various run-time options. This is an example of the way it is run:

build/pc/main.exe -t=30 -r=lossy -rf=Lossy.nss -p 9

TOSSIM provides a list of debug modes to use such as USR1,USR2, route, AM, crypto, sensor, etc., depending on what debug messages are required to be displayed at run-time. USR1 - USR4 are for the user to print out debug messages. AM is for displaying the AM ID's of messages sent, crypto is for printing out TinySec messages if used and so on. Two or more of these can be combined too.

2.4.3. PowerTOSSIM

PowerTOSSIM is an application that measures the power consumed by TinyOS applications simulated in TOSSIM. TOSSIM does not model the CC1000 mica2 radio stack which PowerTOSSIM uses as a microbenchmark. Hence PowerTOSSIM ports this stack to TOSSIM and uses it as an energy model. It logs the energy consumed by various components: radio, CPU, analof to digital converter (ADC), sensor boards and EEPROM into a file and then applies a code transformation technique to it every time there is a change in power. Each simulated component makes calls to a new component called *PowerState* which does the logging. An excerpt of this log file for Test3Paths (one of our experimental applications) is shown below:

2: POWER: Mote 2 RADIO_STATE OFF at 7815908

2: POWER: Mote 2 RADIO_STATE ON at 7815908

2: POWER: Mote 2 RADIO_STATE RX at 7815908

2: POWER: Mote 2 ADC SAMPLE PORT 0 at 7921602

2: POWER: Mote 2 ADC DATA_READY at 7921602

2: POWER: Mote 2 RADIO_STATE TX at 7922402

2: POWER: Mote 2 RADIO_STATE RX at 8252002

2: POWER: Mote 2 ADC SAMPLE PORT 0 at 8346402

PowerTOSSIM has a detailed power model of the mica2 node which is used with the logged power state transition messages to generate a detailed energy consumption estimate of each component of the node. PowerTOSSIM also models the CPU cycle count of each node, but this was not used by the applications in this thesis. This is done by using CIL (C Intermediate Language) which maniputales the C source file generated by the nesC compiler, breaks it into blocks and inserts an execution counter into each block.

In addition to energy consumption, PowerTOSSIM also provides a detailed current draw for each node outlining the current consumed in different states: idle state, radio receiving state, transmitting state and radio off state.

CHAPTER 3

DESIGN OF THE EXPERIMENTS

I designed experiments to test different data encoding schemes by creating an encoding module that is designed to send and receive messages using the mica2 radio stack. The experiments were performed through simulations that were run on TOSSIM, and power consumption analysis was done on PowerTOSSIM. The basic idea was to apply erasure encoding schemes to sensor networks to reduce the power consumption and explore the cost-effectiveness of these techniques. Erasure encoding schemes have been used in multicast networks for a long time, but have not been tried in sensor networks until now. Applying these codes will reduce redundancy in the network as compared to basic redundant routing schemes like INSENS [10]. But this also reduces the resilience of the network as compared to INSENS [10]. I implemented three applications for my experiments: Test3Paths, Test4Paths and TestAgg.

Test3Paths: This sends three packets along three different paths from the source node to the base station. All the remaining nodes along the multihop path forward the packet until it reaches the base station. Depending on whether encoding is applied or not, Test3Paths either sends the original data set along three different paths or splits the data set and sends the pieces along with the encoded data along three different paths.

Test4Paths: This sends four packets along four different paths from the source node to the base station. All the nodes in the network forward the packet until it reaches the base station. If encoding is applied, Test4Paths splits the data packet and sends the pieces along with the encoded data along four different paths. If not, it sends the original data packet along four paths.

TestAgg: This sends three packets along three different paths from the source node to the base station. On one of the paths, there is an aggregating node which takes all the packets

from other nodes, suppresses them until it collects all data, decodes it and then routes it to the base station.

3.1. Design of Test3Paths

Test3Paths was designed as a simple multi-hop application which sends data along three different paths to the base station. One execution of Test3Paths sends packets simultaneously along three paths. If erasure encoding is to be used, Test3Paths takes a data packet and splits it into two different data packets, applies erasure encoding on these two packets which forms the third packet and sends the three packets along three different paths to the base station. As shown in Fig. 3.1, the three packets are sent from node 2, which is the source node, to node 0, which is the base station.

Message structure and power modes: A TinyOS message packet has 3 parts: a preamble, a header and a payload. The preamble is 28 bytes long, the header 7 bytes and the payload can be up to 29 bytes. The payload is provided by the application and the multihop suite of applications takes 7 bytes off it for the multihop header. Hence the final payload can be up to 21 bytes long. The preamble is provided by the CC1000 radio stack. TinyOS supports two radio stacks: CC1000 and CC2420. While TOSSIM simulates both, PowerTOSSIM simulates the power consumption for only the CC1000 stack, which has an operating frequency of 300-1000 MHz with data rates of up to 76.8 Kbps.

Different power modes: In this application, the values used are 0 for the highest duty cycle (100%) at a data rate of $\sim$42 Kbps and the lowest is 3 for the lowest duty cycle (7%) at a data rate of $\sim$1.7 Kbps. Duty cycle is the time period for which a node is functional. Transmitter and receiver should be at the same duty cycle and data rate to be able to communicate with each other. Here node 2 will have the highest data rate while transmitting and its receiving power mode is set at 3 as it does not have to receive anything. Node 0 has transmit set at 3 since it does not have to transmit anything and receive set at 0. The remaining nodes have receive set at 0 but adjust their transmit accordingly as and when they need to transmit.

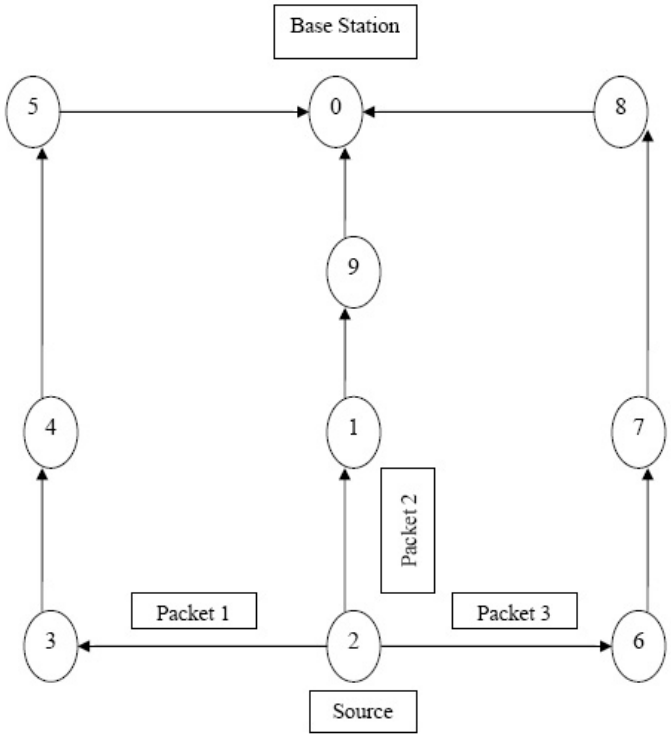

FIGURE 3.1. Topology of Test3Paths.

3.2. Test4Paths

Test4Paths is an application that routes data along four different paths to the base station. If erasure encoding is performed, Test4Paths takes a data packet, splits it into three different packets up to a maximum size of 21 bytes and then computes the erasure encoding of them which forms the fourth packet. It then sends the four packets along four different paths to the base station. The nodes are put into low power modes in the same order as Test3Paths. The network topology is as shown in Fig. 3.2

3.3. TestAgg

TestAgg is an application that routes data along three paths to the base station with data aggregation along a path. On one path to the base station, a node will intercept all the messages, wait till it receives data from all the three paths, decodes the data packets received, assembles the fragments and then routes the result to the base station. Here Node

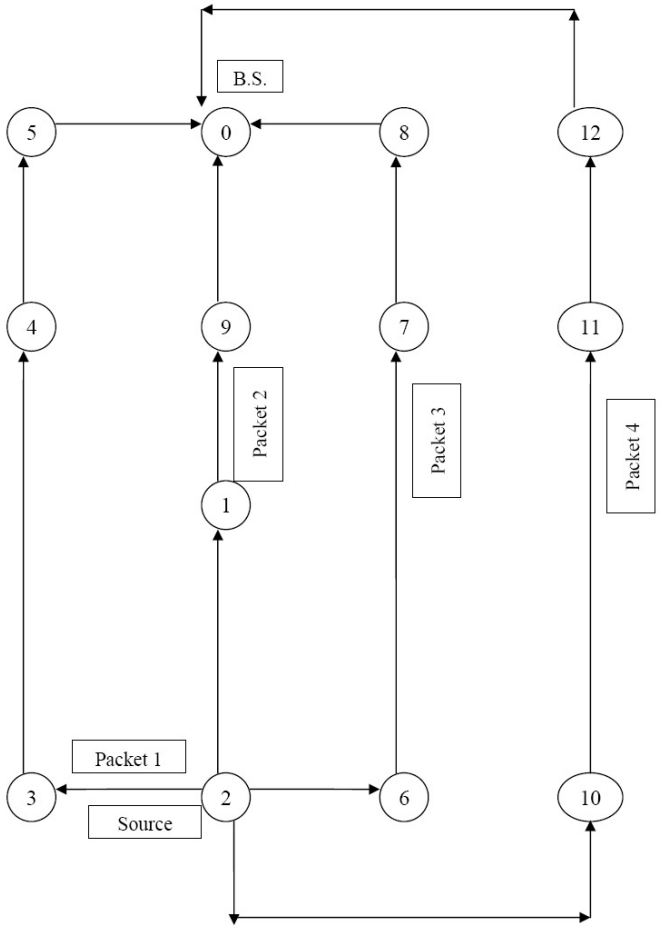

FIGURE 3.2. Topology of Test4Paths.

2 transmits in high power mode but listens at the lowest duty cycle possible since it does not have to receive anything. All the remaining nodes will receive in a high power mode but will adjust their transmit power in a way similar to that described in Test3Paths and Test4Paths. The network topology is as shown in Fig. 3.3

3.4. PowerTOSSIM Power States

PowerTOSSIM provides different CPU power states at different current rates: active, idle, standby and power saving state. The current consumed at each of these states varies from 8.0 mA at active state, 3.2 mA at idle state to 103 μA at power-down state. It uses a table of actual measured radio transmission current values (10 in all) for different modes

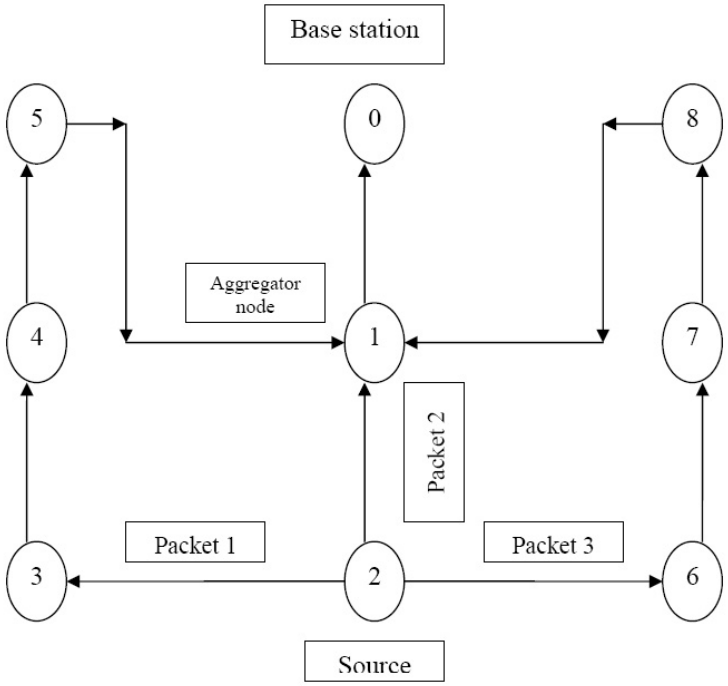

FIGURE 3.3. Topology of TestAgg.

ranging from 3.7 mA to 21.48 mA. The power drawn by all the components (radio, sensors, ADC, EEPROM, etc.) are stored in an energy model file. The receiving current drawn for the radio is fixed at 7.0 mA. The default transmission current is 8.47 mA which was changed to full transmission power at 21.48 mA and the lowest at 3.7 mA. PowerTOSSIM also has values of the current drawn for the sensor board, LED's, EEPROM and ADC.
For measuring the power, the power consumed by each component of the application is computed using these measured values.

3.5. Radio Models

TOSSIM provides a directed graph of independant bit error rates between two nodes. It does not model radio propogation as such. There are two radio models in TOSSIM: simple radio model and lossy radio model. The simple radio model assumes that all the nodes are in a single cell and can hear each other perfectly, hence there are no bit transmission errors. In the lossy model, the network is modeled as a directed graph, where nodes represent vertices

and an edge is the radio link between them. Every edge has a weight that represents the probability of bits getting flipped during transmission. A probability of 0 means that there will be no errors. There are some files representing loss topologies collected from empirical data for grids of 20x20 nodes with varying distances which can be used in simulations. In my experiments, the lossy model was simulated with spacings of 10 feet, 15 feet and 20 feet between nodes.

CHAPTER 4

SIMULATION RESULTS

Objective of the experiments: The goals of the experiments were determining the following:

(i) Determining power requirements for routing packets along 3 and 4 paths as the data size varies;

(ii) Explore the effect of different power modes of the radio when doing multipath routing;

(iii) Determine the relative efficiencies of different encoding schemes, e.g., 1-of-3, 2-of-3, 1-of-4 and 3-of-4;

(iv) Examine the power consequences of doing in-network aggregation;

(v) Compare the power consumed by the simple and lossy radio models.

To this end, the following kinds of experiments were run with increasing data sizes from 0 to 255 bytes for 1-of-3, 2-of-3, 1-of-4 and 3-of-4 encoding schemes:

(i) The simple radio model was simulated with different power modes set on nodes. In this there are no errors in transmission. This was done with all the nodes booting up at the same time.

(ii) The network was simulated with a lossy radio model with spacings of 10 feet, 15 feet and 20 feet.

(iii) The transmit radio current was varied from normal to high.

(iv) In-network aggregation was implemented and the power consumption pattern was studied.

From the experiments run, it was noticed that as the data size increases, the percentage of power difference between encoded and unencoded data increases. This is more pronounced

in larger data sizes. Hence I present the results separately for smaller data sizes (data sizes from 40-140 bytes) and larger data sizes (data from 140-240 bytes).

4.1. Simple Radio Model

The simple radio model has no errors in transmission. Each node can hear all the other nodes perfectly well. There are no bit transmission errors. The simulation was run with all the nodes booting up at the same time. The radio was set at the normal transmit current: 8.7 mA. The power drawn by the source node is shown in the graph below.

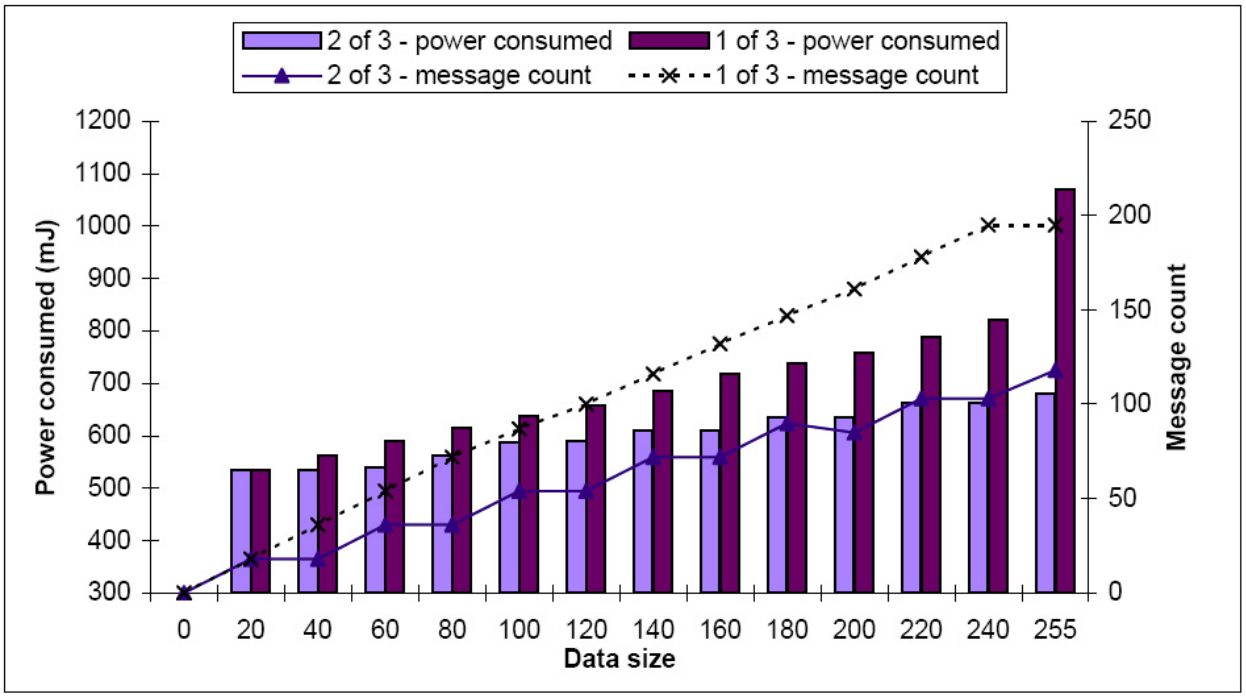

FIGURE 4.1. Test3Paths: Source node power consumption.

4.1.1. Test3Paths

1-of-3: This sends the same data along three different paths. 2-of-3: This sends half the data along two paths each, and the encoding of the data along the third path. Fig. 4.1 shows the results for the source node for 1 of 3 and 2 of 3 for Test3Paths. In 2 of 3, for transmitting the same total data size, the data along each path will be half of that along each path in 1 of 3. The dark columns in the graph represent the power consumption for 1

of 3 and the lighter columns represent the power consumption for 2 of 3. The results show that the power consumed for 1 of 3 is higher than that for 2 of 3. For small data sizes (up to 140 bytes), there is a power savings of 8.46% and for data sizes 160-240, we get a power savings of 16.08% when using erasure encoding. The graph also shows the number of messages transmitted by the source node. The dotted line shows the message count for 1 of 3 and the solid line shows that for 2 of 3. The messages are transmitted by the source node every 5 seconds for a time period of 30 virtual seconds ($\sim$ 70 real seconds in the simulator).

I have used fragmentation of the data packets which can help in transmitting larger packets than the maximum payload size TinyOS allows per packet. Packets are sent in fragments of at most 21 bytes each. The maximum number of fragments transmitted is 13 for 1 of 3 at a data size of 255 and 7 for 2 of 3 at a data size of 255. The TinyOS header is 7 bytes per packet and the header added by the multihop module is 7 bytes. In addition to this, there is a 28 byte preamble added by the radio stack. Hence counting the header and preamble bytes, the total overhead transmitted by the source node per fragment is 42 bytes.

In 2 of 3, the power consumed by the source node increases very gradually, often staying the same for 2 consecutive data values as 2 consecutive data sizes require the same number of fragments to be transmitted. In 1 of 3, the power drawn by the source node increases with each data size. Since total transmission size in 2 of 3 is half that of 1 of 3, the encoded data grows to require additional fragments only after total data size increases by 42 bytes, hence the increase in power is only seen at every second 20-byte total size increase. For a data size of, say, 120 bytes, the source node transmits only 9 fragments, whereas in 1 of 3, it transmits 18 fragments (packets). This reduces the power consumption rate in 2 of 3.

The power consumed by the remaining nodes (other than the source node and base station) is shown in Fig. 4.2. The dark bars represent 1 of 3 and the light ones represent 2 of 3. This varies from 1000 mJ - 1007 mJ. At a data size of 255 bytes, the power consumed by the remaining nodes peaks at $\sim$ 1007 mJ. The dotted line shows the number of messages transmitted in 1 of 3. The solid line shows the number of messages transmitted in 2 of 3.

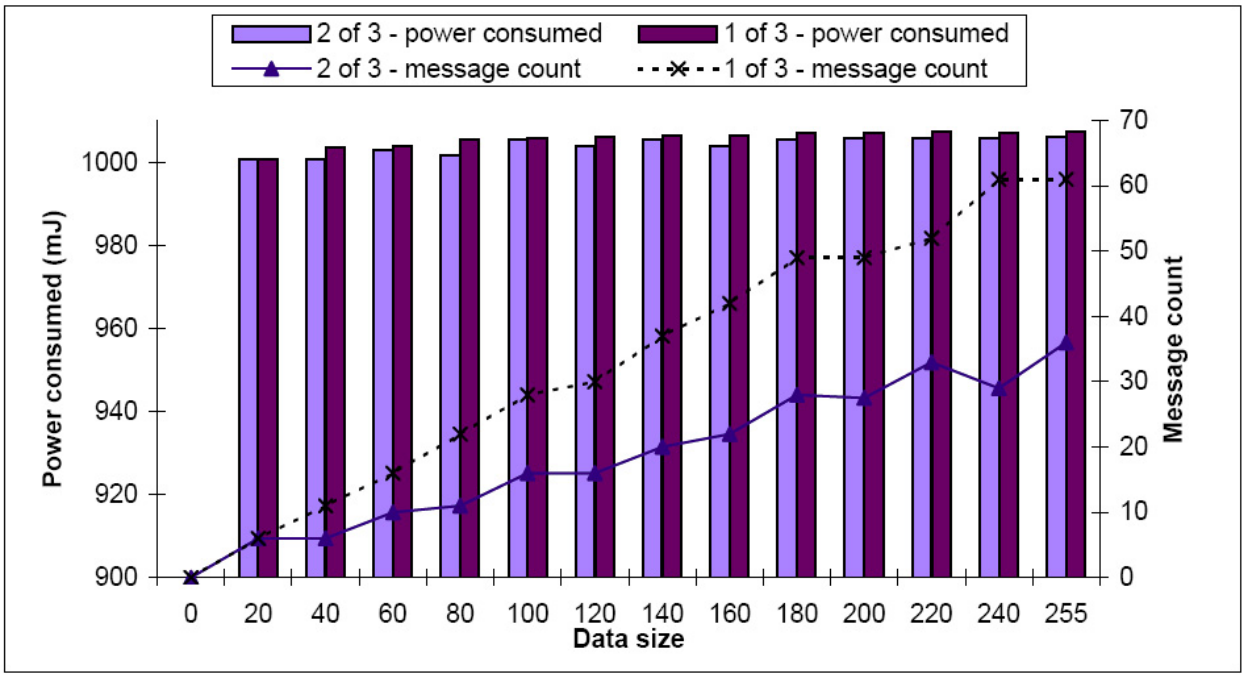

FIGURE 4.2. Test3Paths: Remaining nodes power consumption.

The power drawn by the remaining nodes does not increase as much as the source node as all the nodes always consume significant power due to having to listen for incoming messages and forwarding them. The power drawn for listening remins constant all through and the slight increase in power is due to forwarding messages of larger data sizes. It was seen that at smaller data sizes, each node gets exactly one-third of the transmitted packets, but as data size increases, packets begin to get lost. For example, if the total no. of packets transmitted by the source node is 18, nodes along each of the paths should get 6 packets each. The packet loss may be due to the fact that sometimes nodes along the forwarding path may be transmitting at the same time that the previous node is sending a packet (which they are supposed to be receiving). Hence packets might get dropped along the path. Also, it was observed that the nodes closer to the source node get almost all the tranmsmitted packets but as the number of hops from the source node increases, the number of packets dropped increases.

The power consumed by the base station for 1 of 3 and 2 of 3 is shown in Fig. 4.3. The dotted line respresents 1 of 3. It remains constant as the base station just has to spend time listening for messages which is constant for all data sizes. It is almost same for 1 of 3 and 2 of 3. It was seen that although the base station has to decode the fragments, the CPU power required to decode and re-assemble the fragments is negligible.

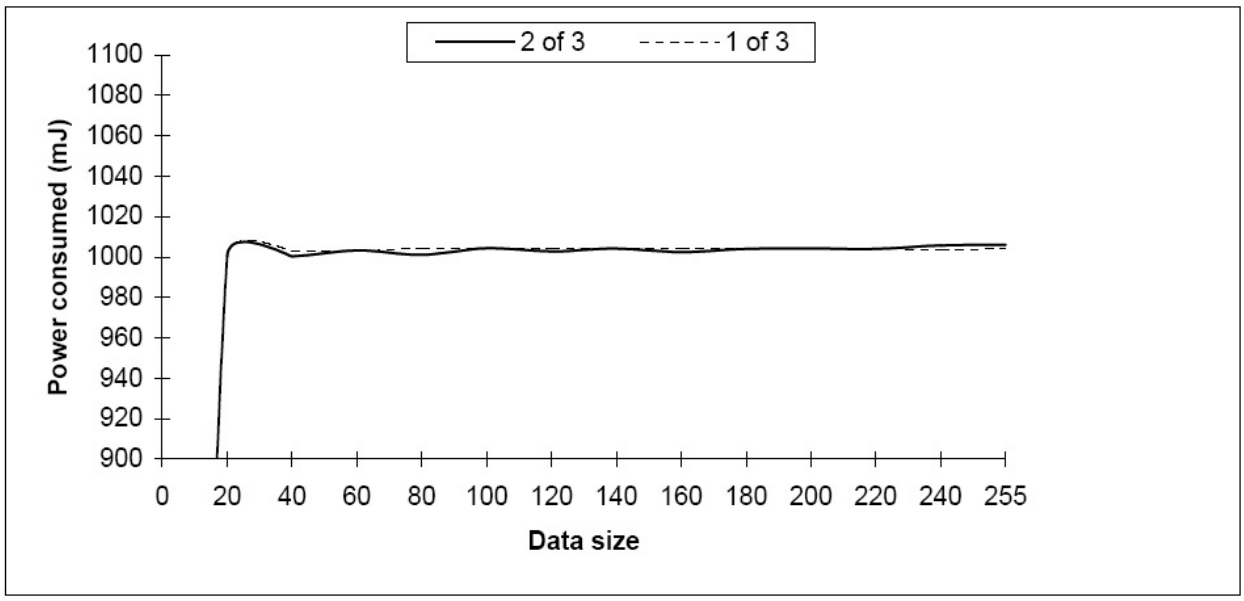

FIGURE 4.3. Test3Paths: Base station power consumption for 3 paths.

4.1.2. Test4Paths

1-of-4: This sends the same data along 4 paths. 3-of-4: This sends one-third of the data along each of three paths and the encoded data along the fourth path.

For Test4Paths, the power consumed by the source node is shown in Fig. 4.4. The dark columns represent the power consumed by 1 of 4 and the lighter ones represent the power consumed by 3 of 4. From the graph, it can be seen that the power consumed by the source node is much higher for 1 of 4 than 3 of 4. For Test4Paths, the simulations were run for 30 virtual seconds with packets being transmitted every 10 seconds.

For 3 of 4, there is a power savings of 6.84% for small data sizes up to 140 bytes and for larger data sizes, we get a power savings of 13.13% when using erasure encoding. Here too,

as in Test3Paths, data is fragmented into packets of at most 21 bytes. As before, the header overhead for each fragment is 14 bytes. The dotted line represents the message count of 1 of 4, the solid line represents that of 3 of 4. Here too as in Test3Paths, it was seen that as the number of hops from the source node increases, the number of messages dropped increases and this also increases with increase in data size.

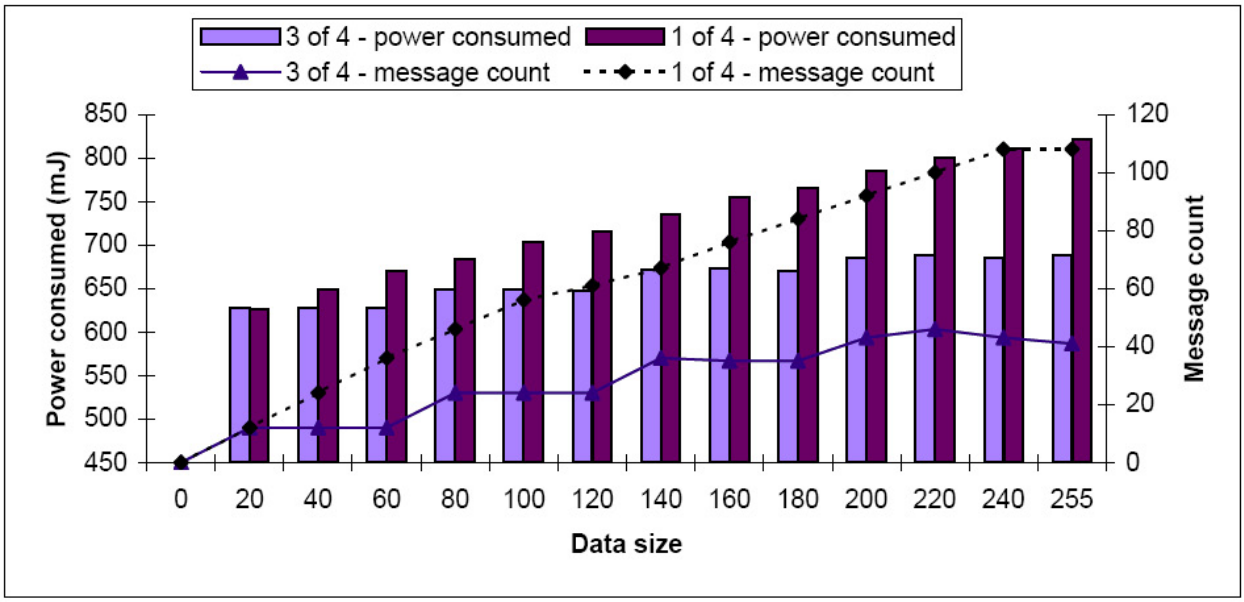

FIGURE 4.4. Test4Paths: Source node power consumption.

For the remaining nodes (other than the source node and the base station), the power consumed is shown in Fig. 4.5. The dotted line shows the number of messages sent in 1 of 4, the solid line shows the number of messages sent in 3 of 4. The power consumed does not vary much: it stays around 1004 mJ throughout for 3 of 4 and varies slightly from 1003 mJ - 1005 mJ in 1 of 4. This is due to the fact that, as in Test3Paths, the remaining nodes spend most of the time listening for messages.

For the base station, the power consumption graph is shown in Fig. 4.6. The base station just listens for messages and hence the power consumption is constant. The dotted line represents 1 of 4 and the dark line 3 of 4. In both cases, the base station consumed almost the same amount of power.

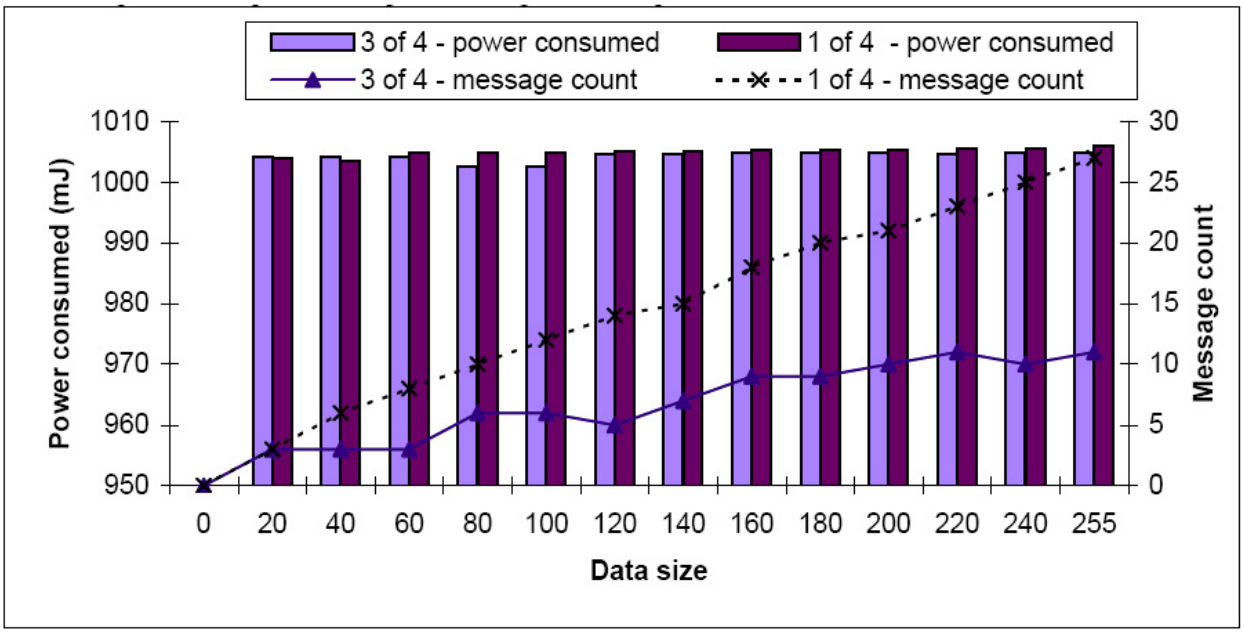

FIGURE 4.5. Test4Paths: Remaining nodes power consumption.

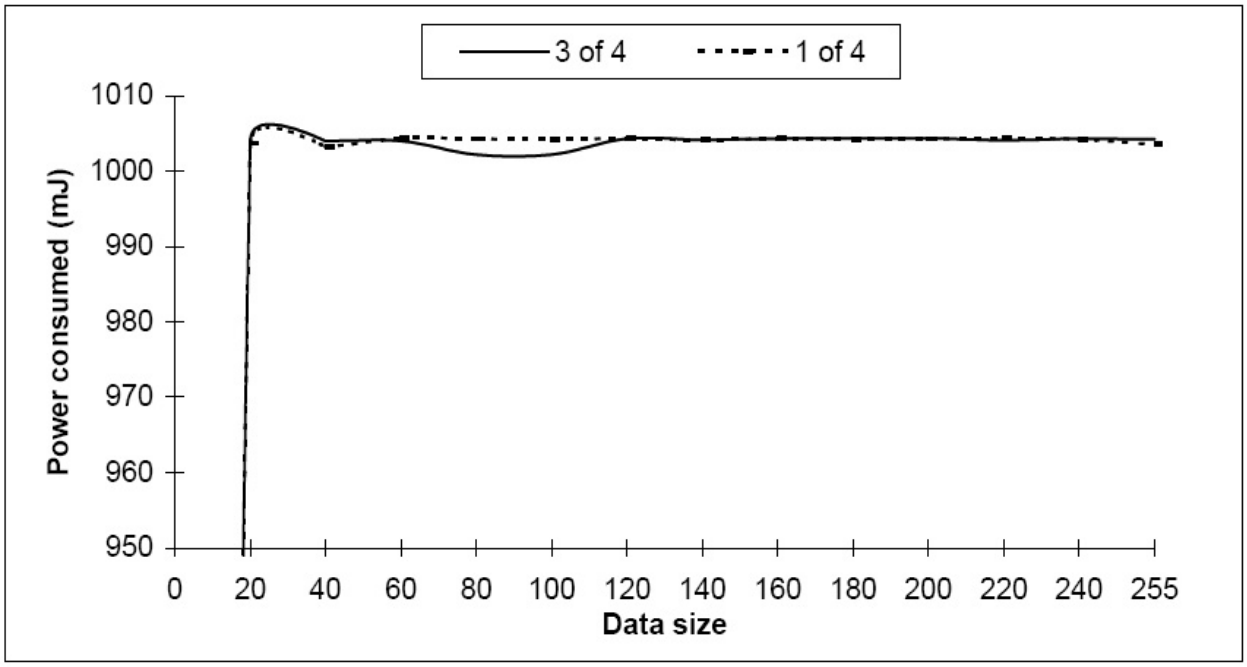

FIGURE 4.6. Test4Paths: Power consumed by the base station.

4.2. Lossy Radio Model

In the lossy model, bit error rates are simulated. TinyOS has a few files which contain a probability distribution for packet loss. These files are based on empirical data collected

over a 20x20 grid with variable distances of 5 feet, 10 feet, 15 feet and 20 feet between them. These files were used for simulating the loss probability, and the simulation was done with re-transmissions in case of loss. Since the nodes transmit in a radius of 50 feet and as the distance from the centre (and the distance between nodes) increases so does the probability of loss, the number of re-transmissions were based on distance: for a distance of 20 feet, there were 3 re-transmissions, for 15 feet, 2 re-transmissions, for 10 feet, 1 re-transmission.

4.2.1. Test3Paths

In Test3Paths, a network of 10 nodes was simulated and the lossy files were used with distances of 10, 15 and 20 feet. For the source node, the power consumed for 2 of 3 is far smaller than 1 of 3. For smaller data sizes, there is not much difference, but for larger data sizes, the difference is significant and ranges from 200 to 300 mJ. For smaller data sizes, the power savings is 8.82% and for larger data sizes, it is 14.96% The results for the source node for 20 feet are shown in Fig. 4.7. The dark columns indicate 1 of 3 and the lighter ones, 2 of 3. The dotted line is for message count of 1 of 3 and the solid line, 2 of 3. The simulations in this section (for Test3Paths) were run for a time period of 60 virtual seconds with the source node transmitting every 20 seconds.

The results for 15 foot spacing for the source node are shown in Fig. 4.8. It was seen that the power consumption for 15 feet is smaller than that of 20 feet. This would be due to the smaller number of re-transmissions done by the source node. The power savings we get for smaller (40-140) and larger data sizes (160-240) is 8.2% and 14.99% respectively. It can be seen that the message count in 15 feet is smaller than in 20 feet. At 20 feet, the source node does up to 40 more transmissions than at 15 feet.

For 10 foot spacing, the results for the source node are shown in Fig. 4.9. The power savings obtained was 4.74% for data sizes 40-140 and 10.4% for data sizes 160-240. This is lesser than the savings obtained for 20 and 15 feet. This would be due to the fact that the source node does just 1 re-transmission as compared to 3 for 20 feet. The number of

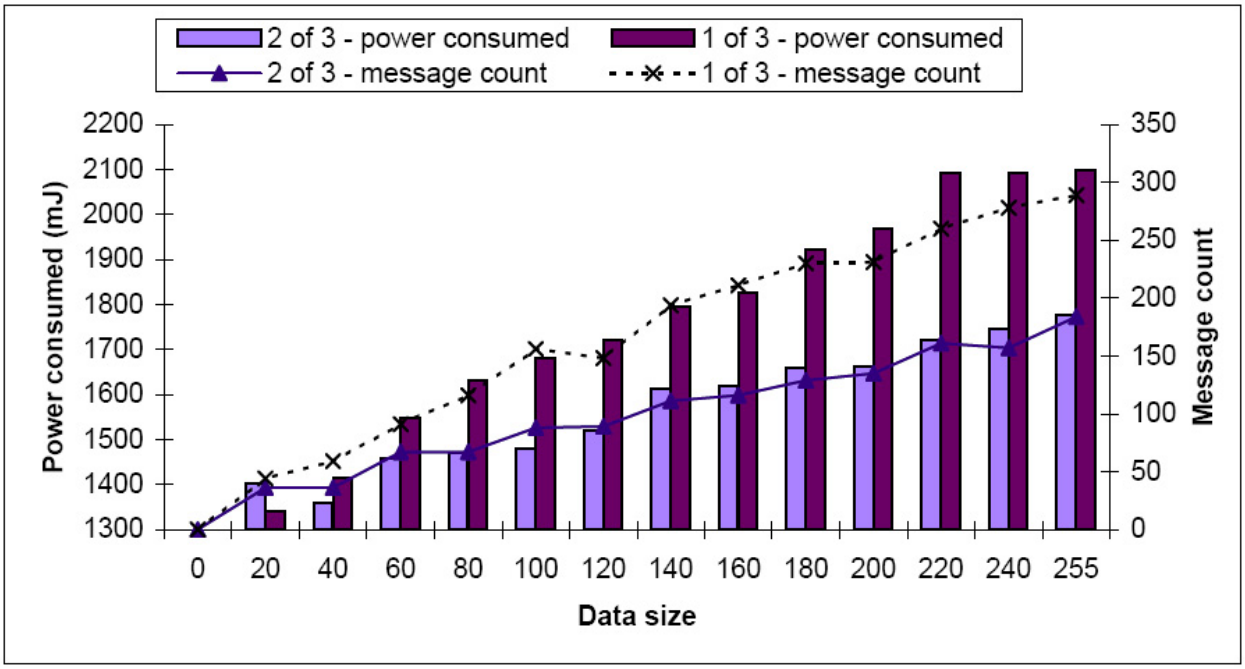

FIGURE 4.7. Test3Paths: Power consumption of source node with lossy distance = 20 feet.

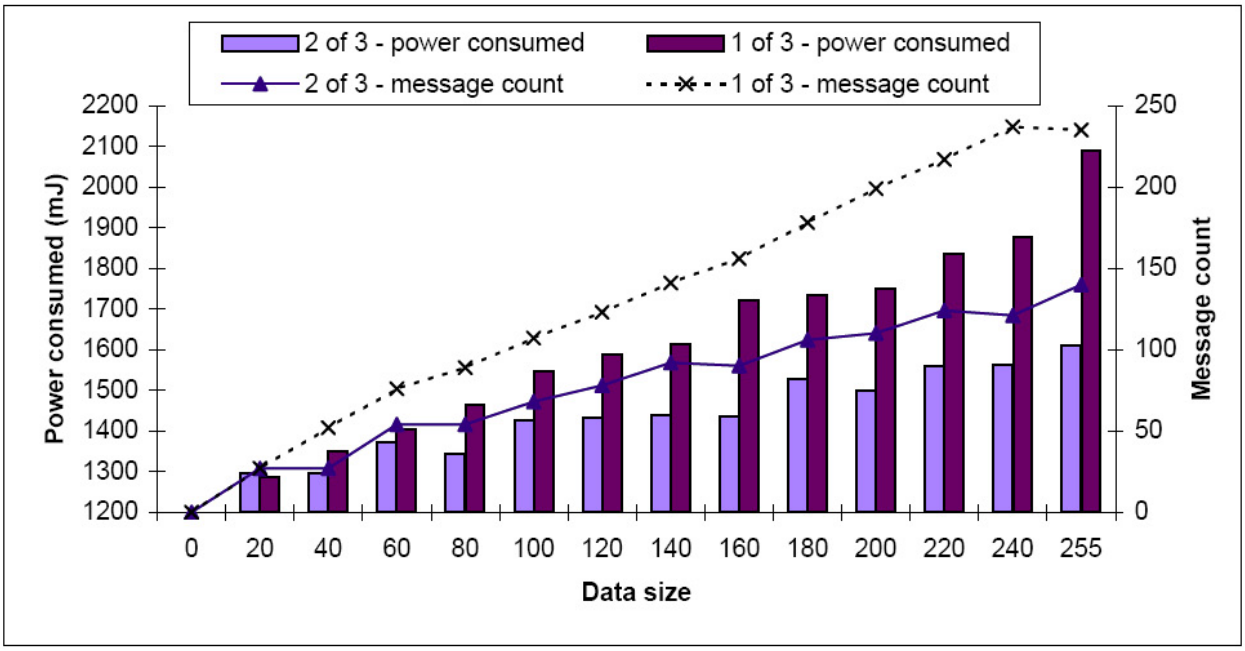

FIGURE 4.8. Test3Paths: Power consumption of source node with lossy distance = 15 feet.

messages transmitted is up to 100 smaller than at 20 feet. This applies for 1 of 3 as well as 2 of 3. For 1 of 3, the difference in power consumed is higher.

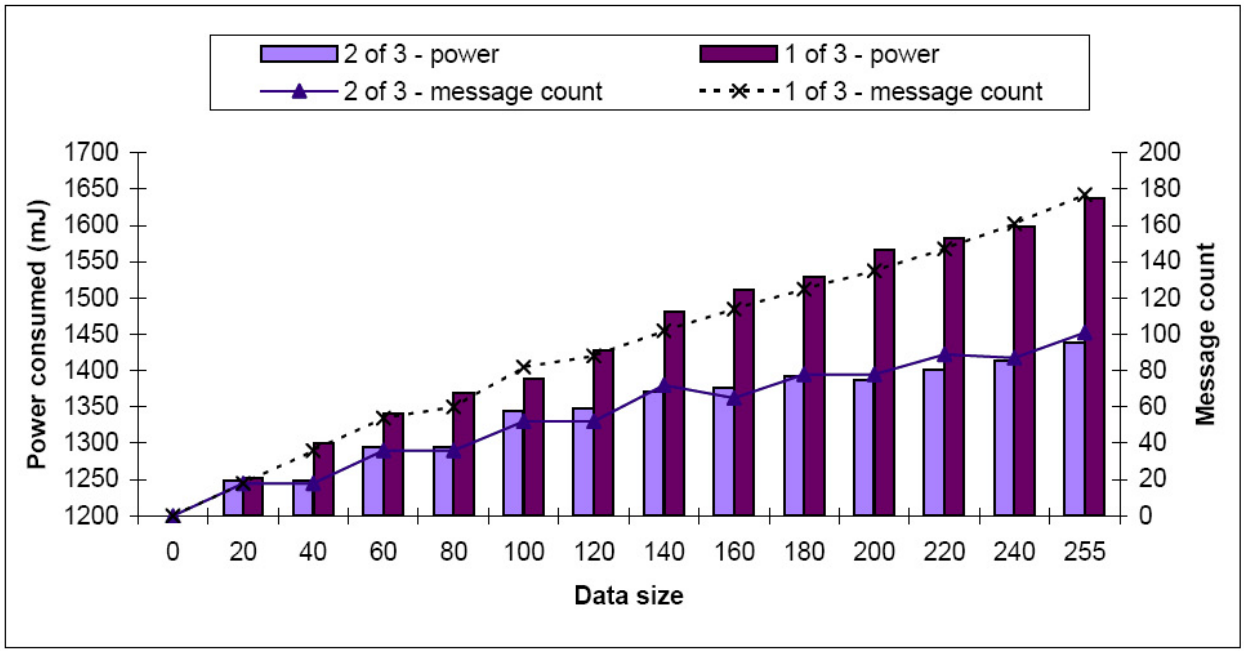

FIGURE 4.9. Test3Paths: Power consumption of source node with lossy distance = 10 feet.

For the remaining nodes, for 20 feet, the results are shown in Fig. 4.10. The difference between 2 of 3 and 1 of 3 is not very high although 1 of 3 consumes marginally more power than 2 of 3. The dark columns show the results for 1 of 3 and the lighter ones for 2 of 3. The dotted line shows the message count for 1 of 3 and the dark line for 2 of 3. It was seen that in 1 of 3, the remaining nodes forwarded almost 100 more messages than 2 of 3.

For 15 feet and 10 feet, the remaining nodes' power consumption decreased by 5-10 mJ. The graphs for them are not shown since this difference is so small. This would be because the remaining nodes spend most of their time listening and waiting for messages and transmit in short bursts. Hence an increase in number of re-transmissions (due to an increase in distance) will not affect their power consumption too much. For the base station, the power consumption remained constant across the data sizes from 0 to 255. Also the power consumption of the base station in 1 of 3 and 2 of 3 remained the same. This was

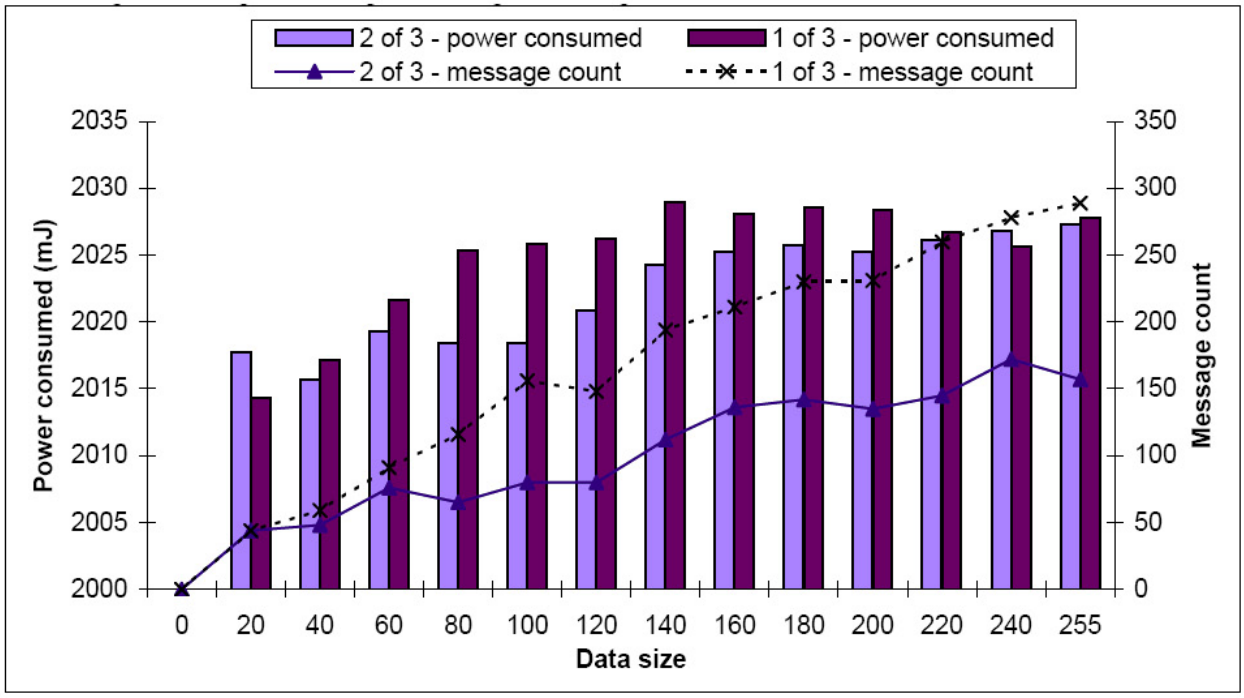

FIGURE 4.10. Test3Paths: Power consumption of remaining nodes with lossy distance = 20 feet.

similar to the results obtained in the simple model. The results for the base station for Test3Paths are shown in Fig. 4.11. The dotted line represents 1 of 3 and the dark line, 2 of 3.

The base station power was almost the same for 20 feet, 15 feet and 10 feet. Hence the results for 15 feet and 10 feet are not shown. This would be because the base station only listens for messages and does not trasmit anything. The differences in power consumption for different nodes for different distances is due to the difference in the number of re-transmissions. This increases/decreases the total power consumed. Hence for the base station, this does not make any significant difference.

4.2.2. Test4Paths

For Test4Paths, the simulations were run for 10 feet, 15 feet and 20 feet. The simulations for Test4Paths were run for 60 seconds with the source node transmitting data every 30 seconds. The results for the source node for 20 feet are shown in Fig. 4.12. The dark

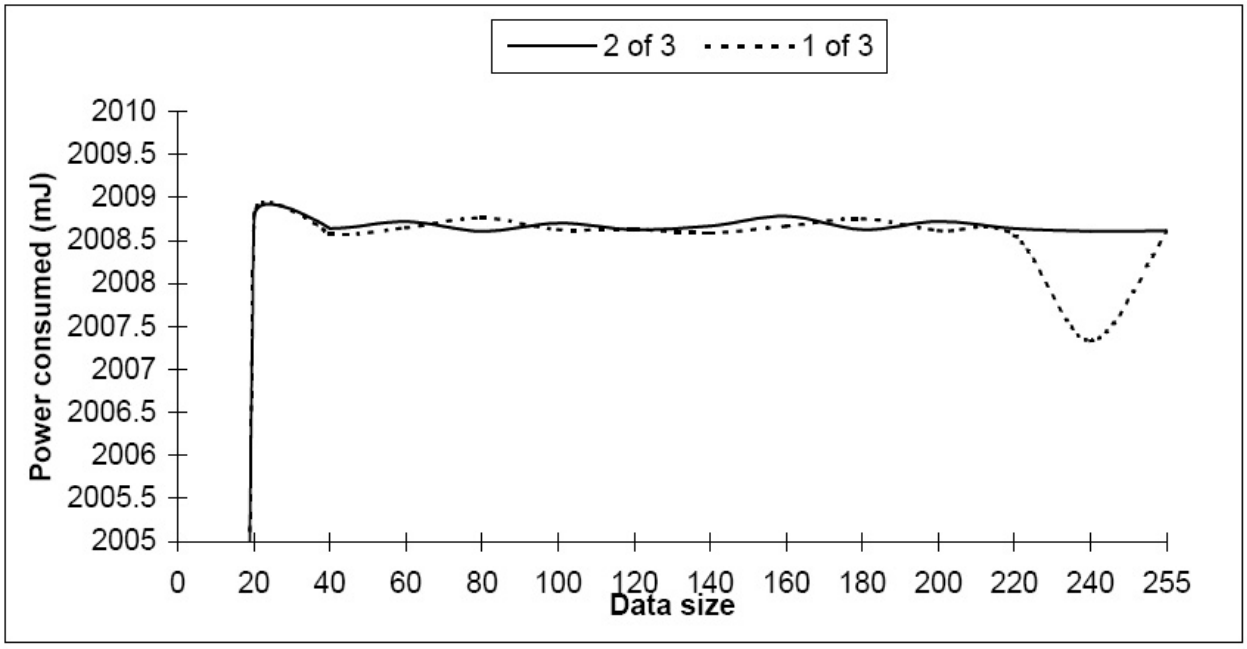

FIGURE 4.11. Test3Paths: Power consumption of base station.

columns represent 1 of 4 and the lighter ones, 3 of 4. From the graph, it can be seen that the power consumed by 1 of 4 is far higher - up to 400 mJ more than 3 of 4. Using erasure encoding results in a 6.84% power savings for data sizes 40-140 and 13.13% savings for data sizes 160-240. In 3 of 4, the power consumed remains the same for 3 consecutive data sizes. This is because 3 consecutive data sizes require the same number of fragments to be transmitted.

For 15 feet, it was seen that the power consumption decreased by up to 45 mJ (3 of 4) and 100 mJ (1 of 4). Since the number of re-transmissions is smaller in 15 feet spacing, the number of transmissions by the source node (message count) went down by almost 50. The results are shown in Fig. 4.13. The dotted line shows the message count for 1 of 4 and the solid line for 3 of 4.

For 10 feet, with 1 re-transmission per lost message, the power consumed by the source node was approximately 100 (3 of 4) - 300 (1 of 4) mJ smaller than 20 feet. It was seen that the differences in power for different spacings are more in 1 of 4 and 1 of 3 than in 3 of 4 and 2 of 3 respectively. The results for the source node are shown in Fig. 4.14.

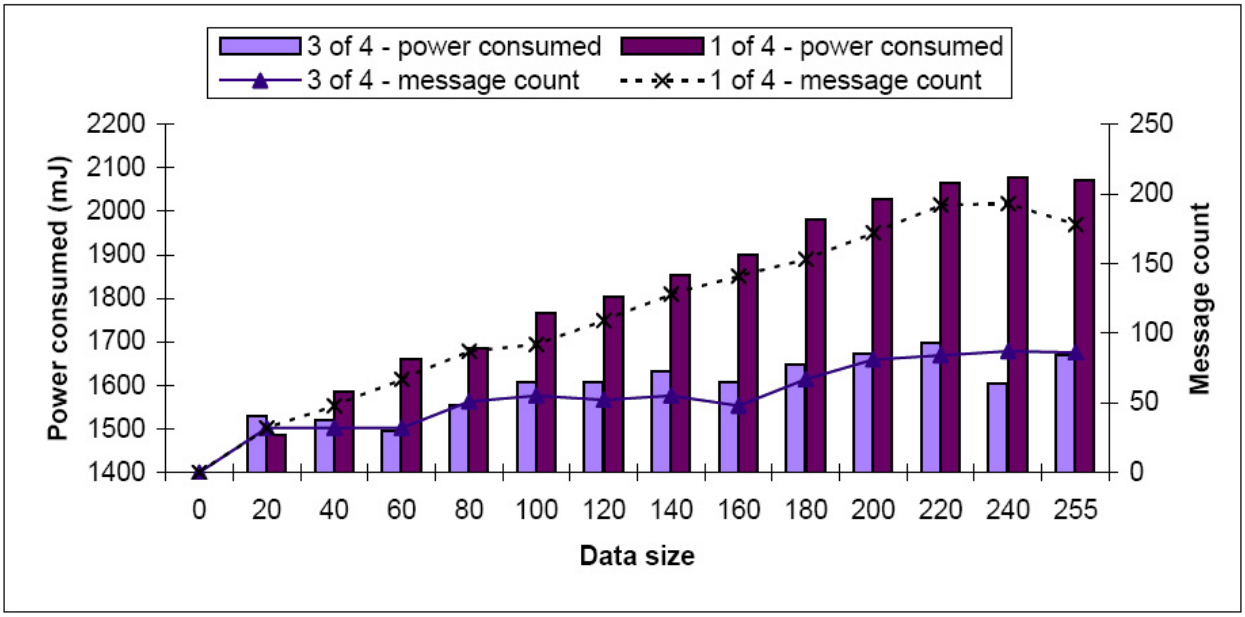

FIGURE 4.12. Test4Paths: Power consumption of source node with lossy distance = 20 feet.

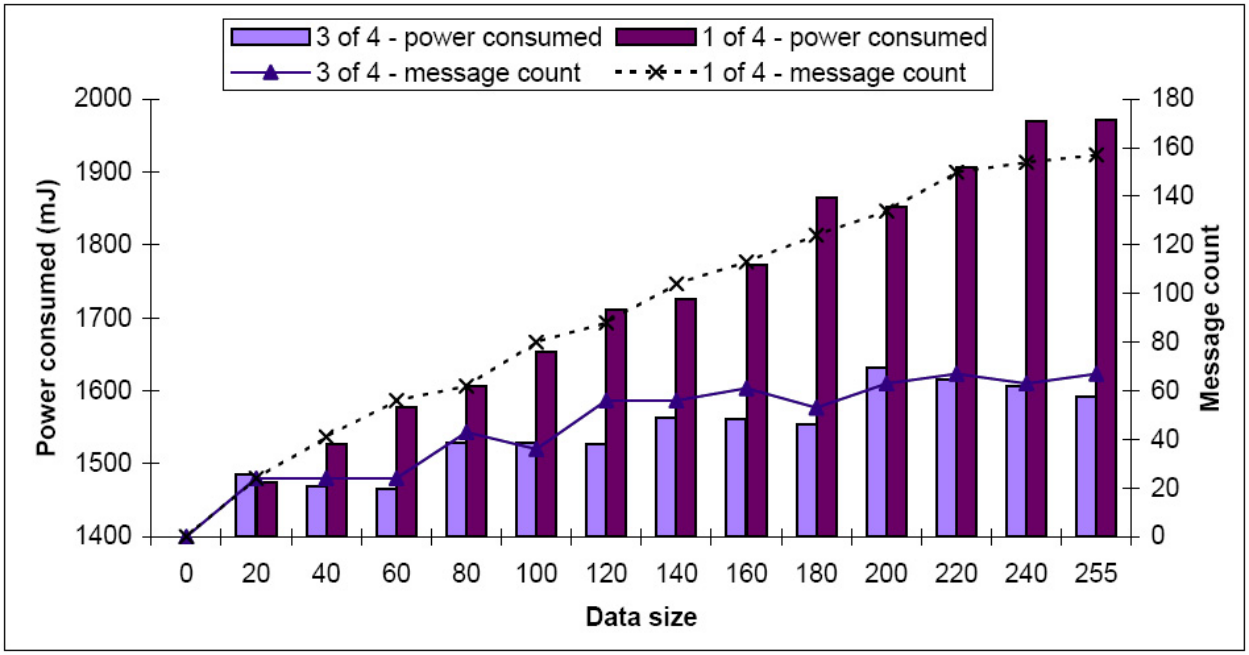

FIGURE 4.13. Test4Paths: Power consumption of source node with lossy distance = 15 feet.

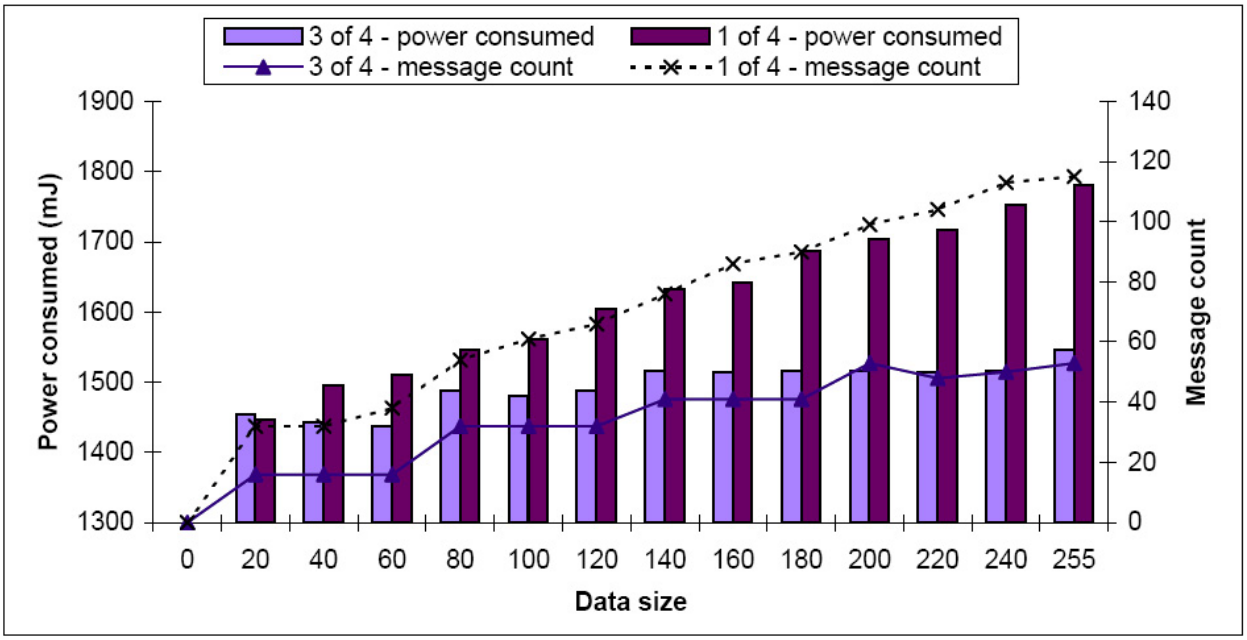

FIGURE 4.14. Test4Paths: Power consumption of source node with lossy distance = 10 feet.

For the remaining nodes, the power consumption, as in Test3Paths, did not vary much for 20, 15 and 10 feet distances. The power consumed by 1 of 4 was marginally more than 3 of 4. The results for 20 feet are shown in Fig. 4.15. This would be again due to the fact that the remaining nodes spend most of their time waiting for and listening to messages and transmit occasionally. Hence a few re-transmissions more would not affect their power consumption too much.

The power consumed by the base station for Test4Paths is shown in Fig. 4.16. This did not vary across the different distances and even for 1 of 4 and 3 of 4, it remained somewhat the same. Hence the graphs for 15 feet and 10 feet are not shown. The base station spends all its time listening for and receiving messages. Hence an increase in the number of transmitted mesages will not affect its power consumption pattern.

4.3. Varying the Power of the Radio

The above simulations were done with the power modes of individual nodes being set individually to different levels: The base station (node 0) had receive set at a high duty cycle

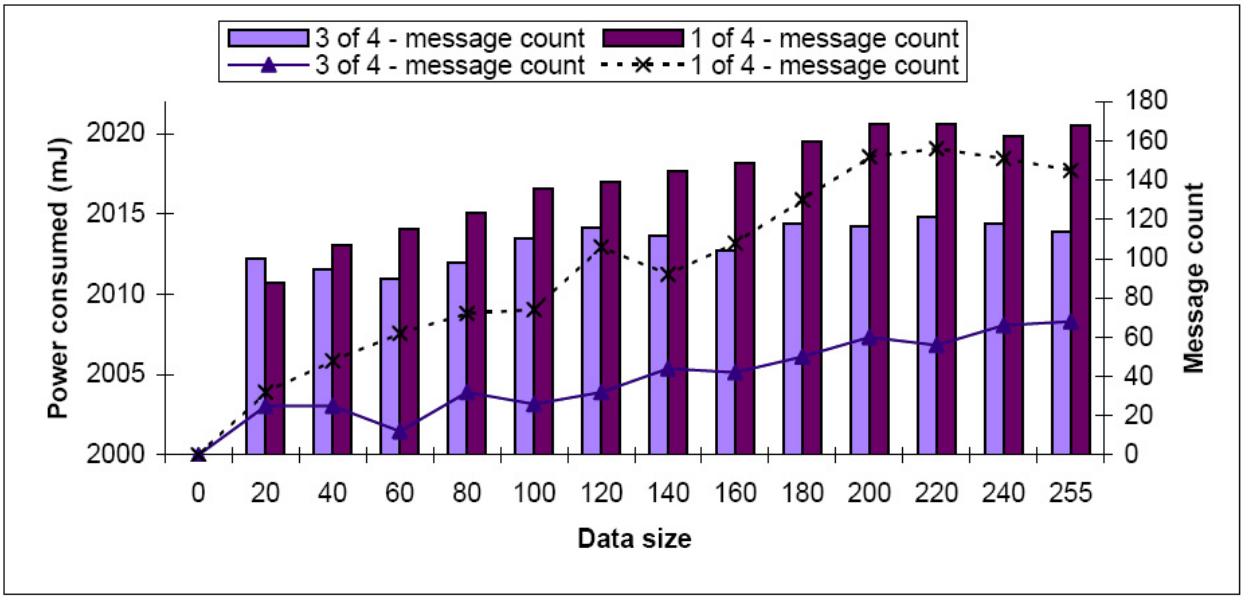

FIGURE 4.15. Test4Paths: Power consumption of remaining nodes with lossy distance = 20 feet.

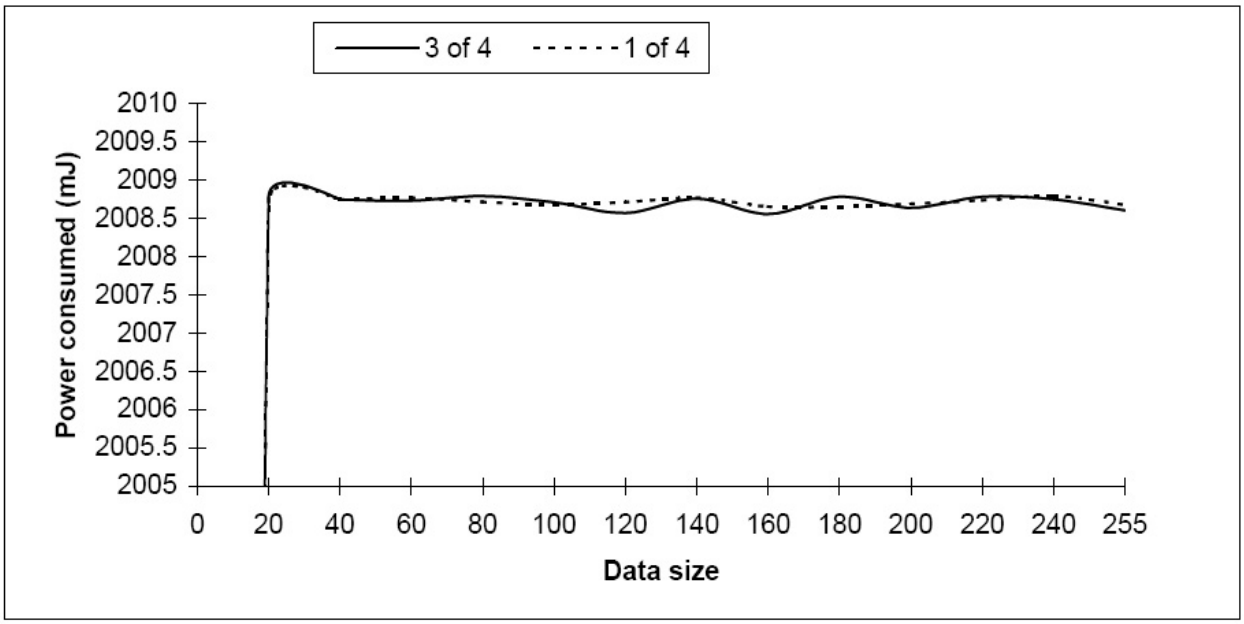

FIGURE 4.16. Test4Paths: Power consumption of base station.

(0) but transmit set at a low duty cycle (3). The source node (node 2) had transmit set at a high duty cycle (0) but receive set at the lowest duty cycle (3). When any of the remaining nodes want to transmit, they can set the power high just before transmission and as soon as

they finish, the power is set to a low duty cycle. The receive power of the remaining nodes is kept high throughout as the exact time when the messages will arrive at the forwarding nodes cannot be predicted beforehand. The transmit current of the radio was till now set to the normal transmit power: 8.7 mA. For the simulations in this section, the transmit current was set to the highest possible value: 21.47 mA.

4.3.1. Test3Paths

The results for power consumption of the source node for Test3Paths are shown in Fig. 4.17. When the radio is kept at the highest transmit current, the power consumption of the nodes increases to almost two times that of the radio at normal trasmit current. In the graph, the dark columns represent 1 of 3 and the lighter ones represent 2 of 3. The graph shows that the power consumed by 2 of 3 is far lesser than that of 1 of 3. Using erasure encoding results in a power savings of 11.66% for data sizes 40-140 and 21.45% for data sizes 160-240. The simulations were run for 30 virtual seconds with the source node transmitting data every 5 seconds.

Comparing the results for the radio transmit current set at normal to the radio set at high, we can see that the power consumed by the source node increases significantly by about 200 mJ (2 of 3) and 600 mJ (1 of 3).

The power consumed by the remaining nodes with the radio set at high is shown in Fig. 4.18. For 1 of 3, the power consumed by the remaining nodes goes from 1000 mJ - 1038 mJ. This is a bigger range as compared to the radio set at normal transmit current. As the radio current increases, the transmit power increases and the ratio of transmit current:receive current increases. Hence the power consumed for forwarding packets also increases. For 2 of 3, the power consumed varies from 1004 mJ - 1023 mJ. The number of messages transmitted and received remains the same as before.

The power consumed by the base station for Test3Paths for the radio set at high is shown in Fig. 4.19. The dotted line represents the power drawn by 1 of 3. The power consumed for 2 of 3 and 1 of 3 increase very slightly from 1000 - 1004 mJ. Comparing the results for the

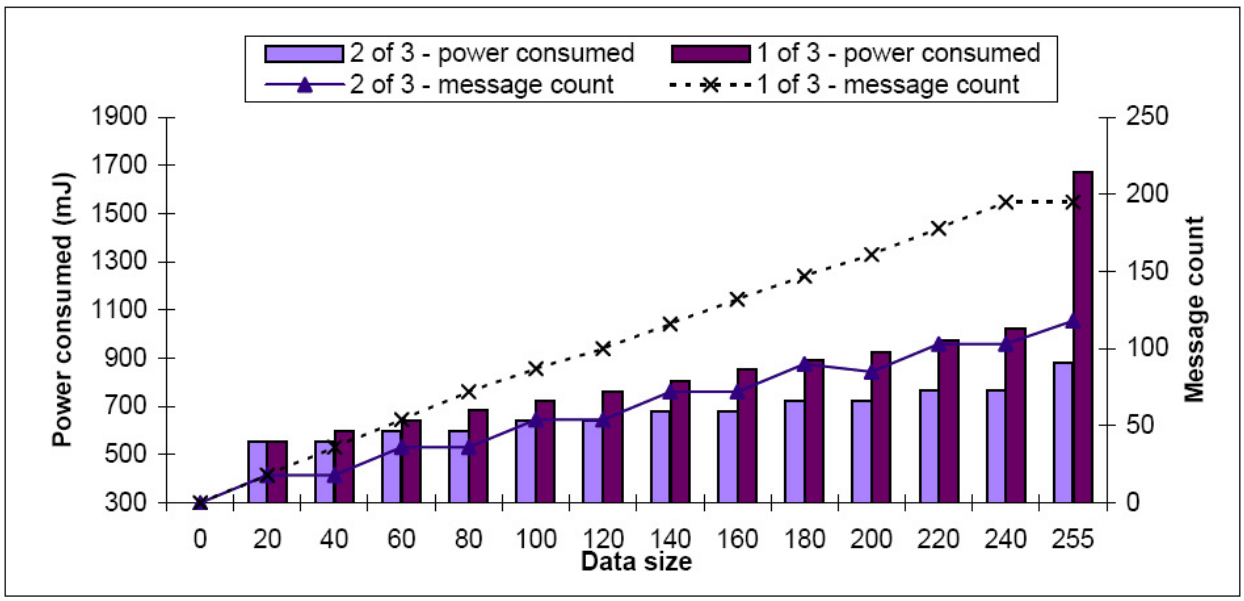

FIGURE 4.17. Test3Paths: Source node power consumption with radio set at high.

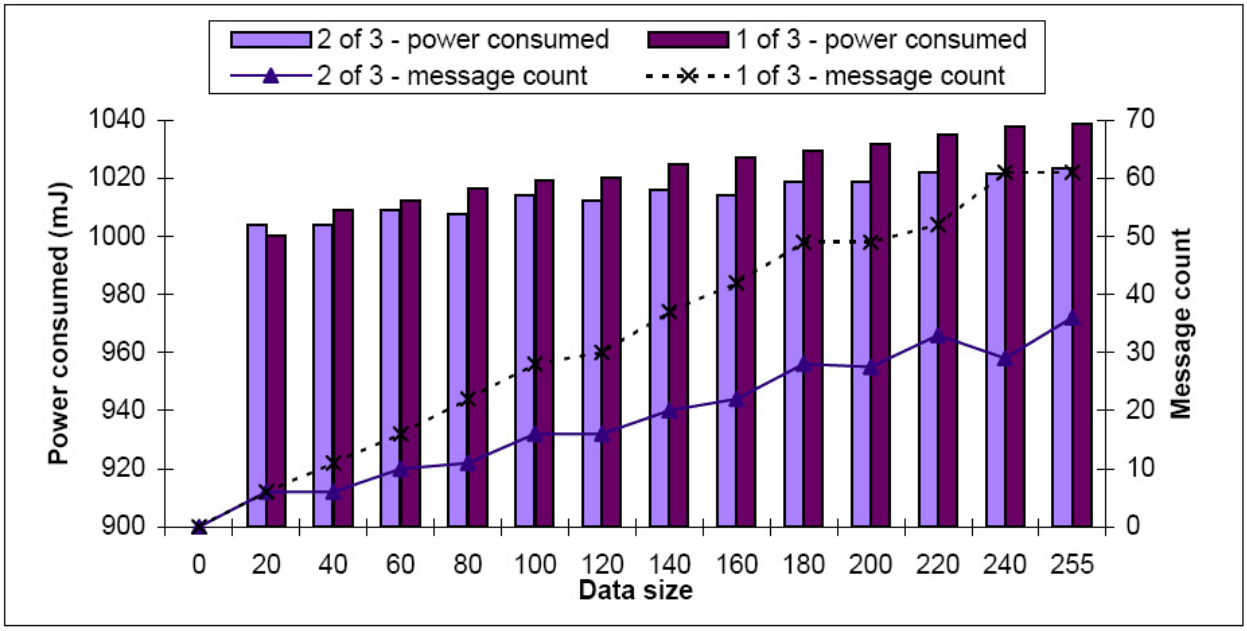

FIGURE 4.18. Test3Paths: Power consumed by remaining nodes with radio set at high.

radio set at high to the radio set at normal transmit current, it can be seen that the power consumed by the base station is relatively unaffected. This is because the base station only

receives messages and does not transmit anything. Hence any changes in the radio transmit current will not affect the power consumption of the base station.

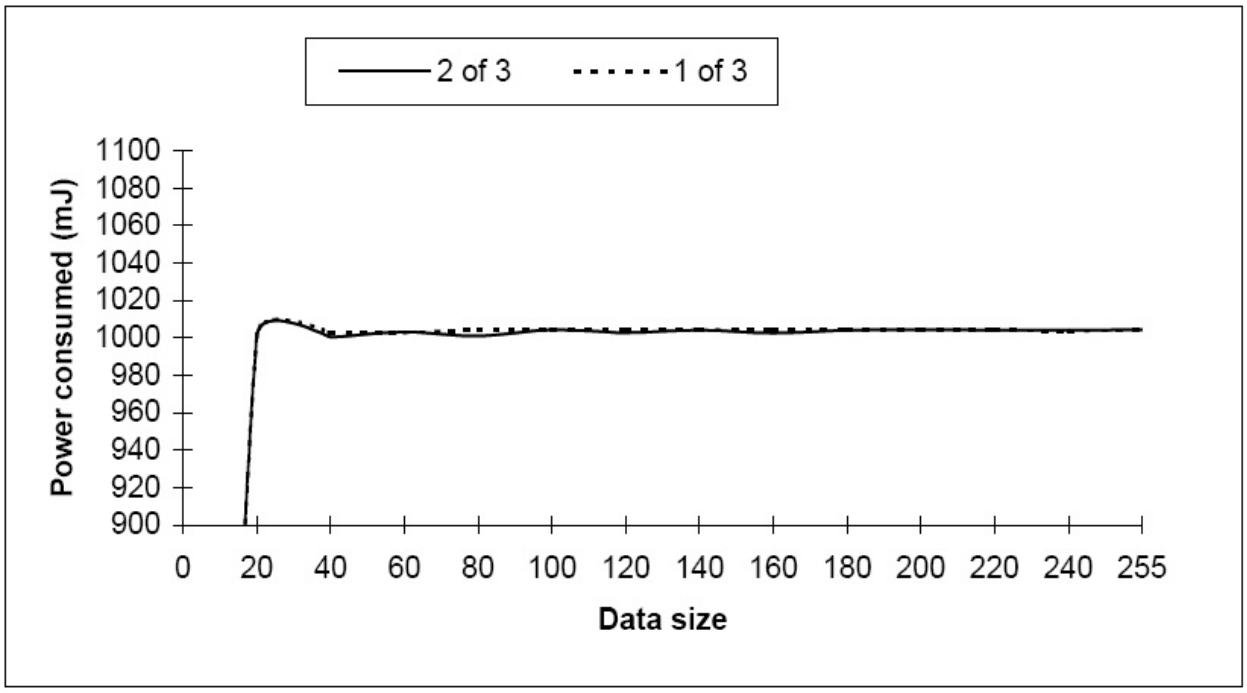

FIGURE 4.19. Test3Paths: Base station power consumption for radio set at high.

4.3.2. Test4Paths

Test4Paths was also run with the radio transmit power set to high. The results for the source node are shown in Fig. 4.20. The dark columns and dotted lines represent the power consumption and message count of 1 of 4 and the lighter columns and solid lines represent 3 of 4. It can be seen that the power consumed by the source node is more for 1 of 4 than for 3 of 4. The power consumed as compared to the radio set at normal transmit current increases by about 100 mJ for both 1 of 4 and 3 of 4. Using erasure encoding, we get a power savings of 11.6% for data sizes 40-140 and 21.15% for data sizes 160-240.

The simulations were run for 30 virtual seconds with the source node transmitting data every 10 seconds.

The power consumed by the remaining nodes for the radio set at high for Test4Paths is shown in Fig. 4.21. The power consumption for the remaining nodes increases a bit more

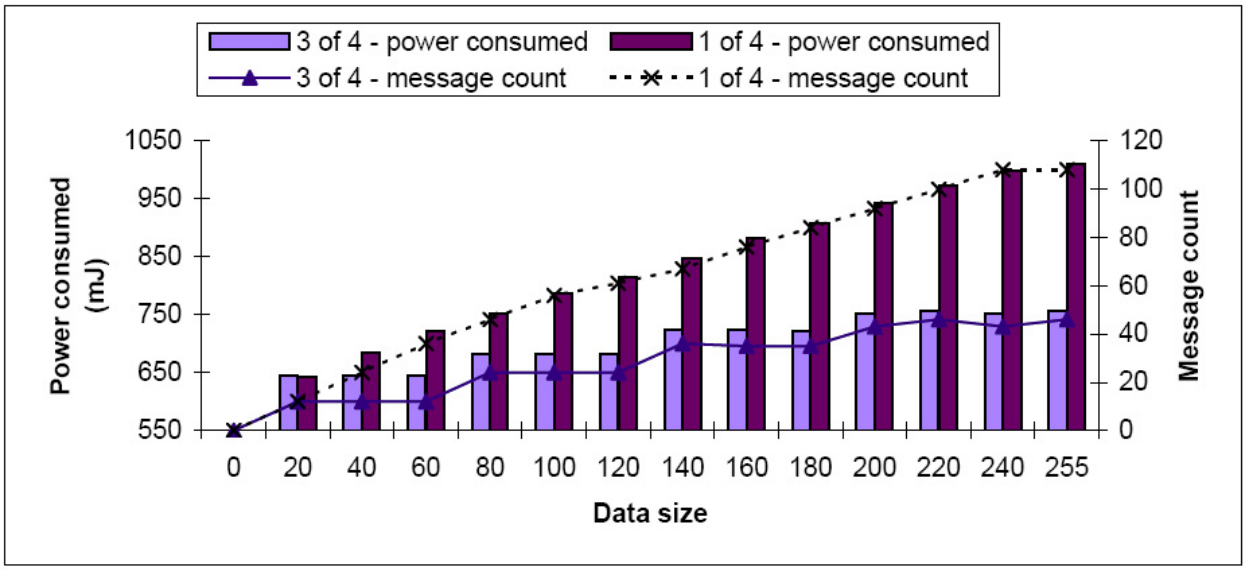

FIGURE 4.20. Test4Paths: Power consumed by source node for radio set at high.

than that of the radio set at normal ($\sim$ 4-5 mJ more). The power consumption is higher for 1 of 4 than 3 of 4. This is consistent with that observed in Test3Paths.

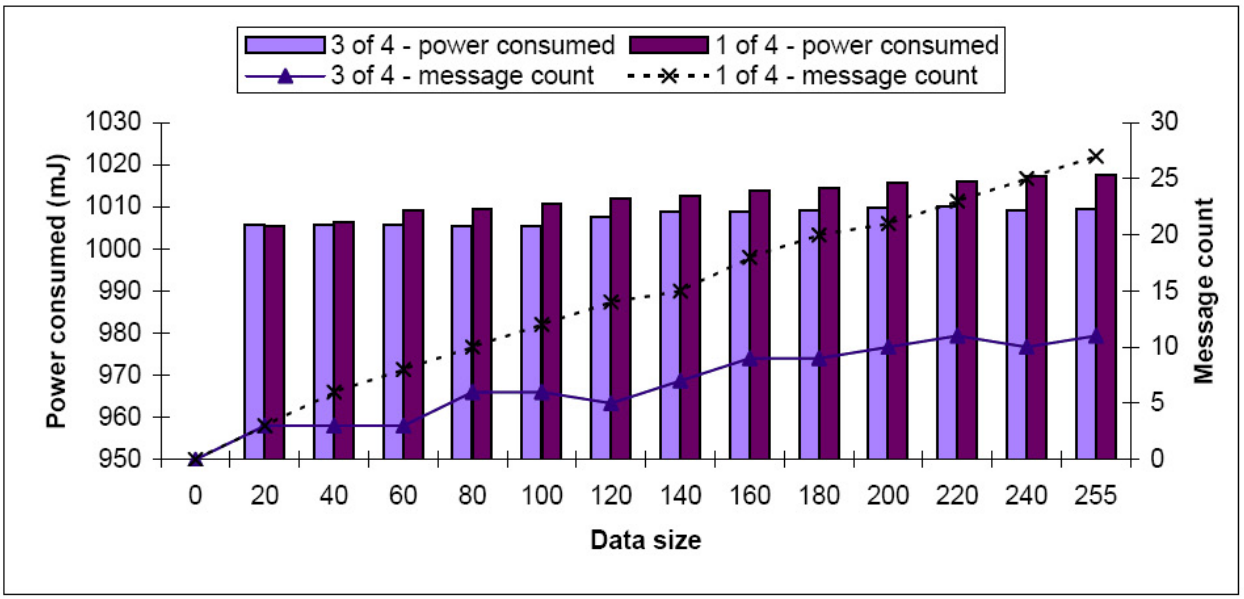

FIGURE 4.21. Test4Paths: Power consumed by remaining nodes with radio set at high.

For the base station, the power consumption graphs are shown in Fig. 4.22. The dotted lines represent 1 of 4. The power consumption for the base station does not increase as

compared to the results for the radio set at normal for both, 1 of 4 and 3 of 4 as the base station just has to listen for incoming messages and a change in the transmit current does not change its power consumption pattern.

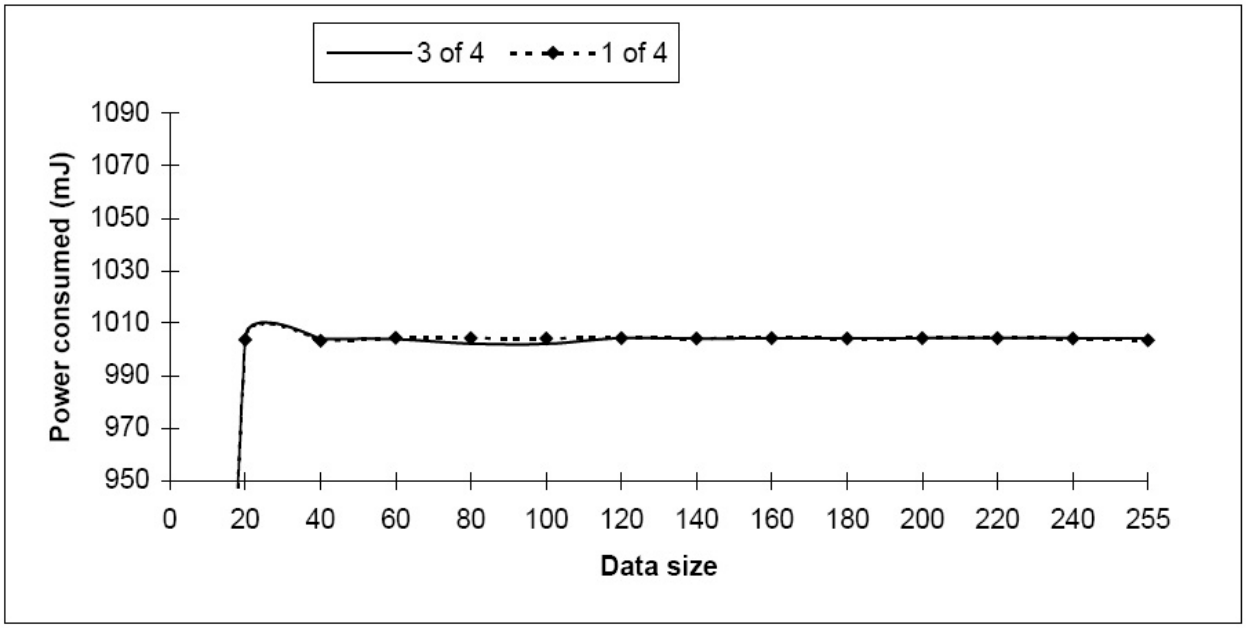

FIGURE 4.22. Test4Paths: Power consumed by base station with radio set at high.

One thing that could be observed when the radio was set at a higher transmit current was that the number of dropped messages decreases. Even nodes that are a few hops away from the source node get all the packets correctly for most of the data sizes and even at higher data sizes the number of dropped messages decreases.

4.4. Aggregation

In a sensor network, each node transmitting data to the base station may be too power consuming if all of them are transmitting similar kinds of data. One way of reducing this might be to designate one node to collect all relevant data, process it and send it to the base station. Networks can have hierarchies where clusters of nodes report to their respective aggregator nodes. The processing of the data might be application specific: in a network sensing temperature or light, the aggregator might just take the average of all the sensor

readings and route it to the base station. This reduces unnecessary redundant or duplicate messages to the base station, as in such networks a lot of readings might be similar. In this module, there is a node in the network which collects messages from all the three different paths and sends them to the base station. The aggregating node is node 1. It intercepts each of the messages going along the three paths, supresses them till all the three messages have been received, decodes the message, and sends it to the base station. The power consumed by node 1 is just 25-30 mJ more than that of the other nodes. This would be the power overhead for collecting and processing all the three messages. The simulations were run for 30 virtual seconds and the source node transmits every 5 seconds. Fig. 4.23 shows the total power consumed by the source node. The power consumed by the source node is the same as that of Test3Paths.

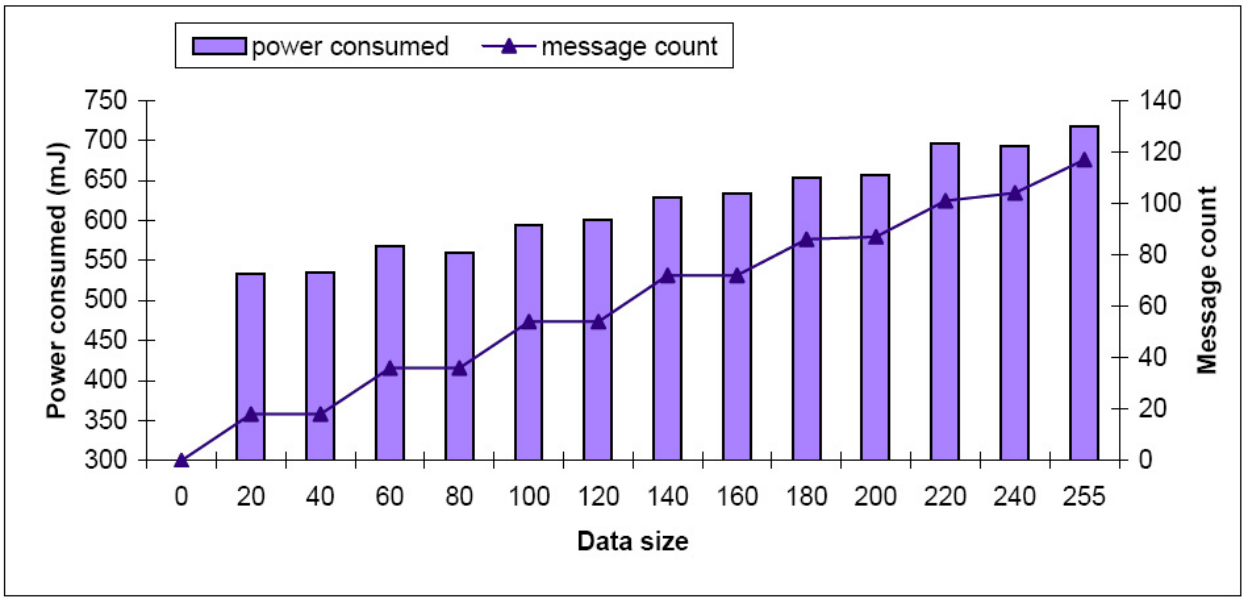

FIGURE 4.23. TestAgg: Power consumed by source node.

The power consumed by the aggregating node (node 1) is shown in Fig. 4.24. The maximum power drawn by the aggregating node at a data size of 255 bytes is 1030 mJ. The dark bars represent the power consumption of the aggregating node and the lighter ones represent the power consumption of the remaining nodes (other than the source node and

base station). It can be seen that the aggregating node draws almost the same power as the other nodes. The difference in the power consumption of the aggregating node increases as the data size increases.

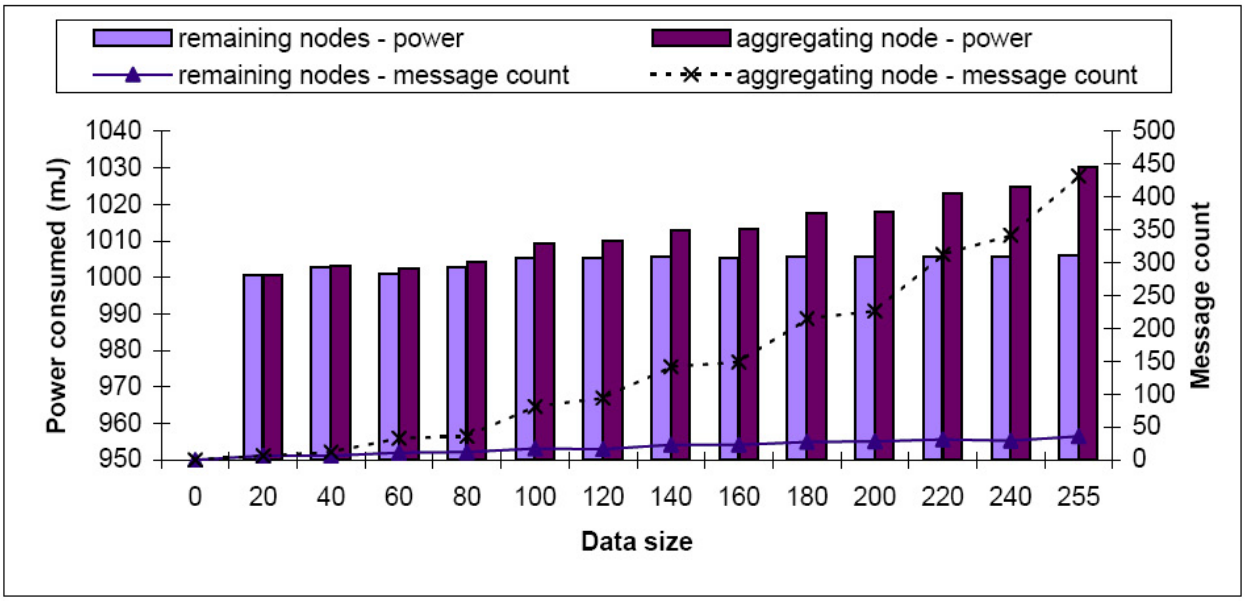

FIGURE 4.24. TestAgg: Power consumed by aggregating node.

The power consumed by the base station is shown in Fig. 4.25. The base station consumes about the same power as Test3Paths and Test4Paths. In TestAgg, the base station receives only one set of messages and does not have to decode them, unlike Test3Paths and Test4Paths, but still its power consumption graph remains the same.

The radio transmit current for TestAgg was also varied from normal to high. The results for the source node are shown in Fig. 4.26. The power consumed by the source node increases by $\sim$100 mJ.

The power consumed for the remaining nodes with the radio set at high is shown in Fig. 4.27. The power consumed by the remaining nodes (represented by the lighter columns) increases by about 18 mJ. This is consistent with what was observed in Test3Paths and Test4Paths: as the radio transmit current increases, the power consumption of the remaining nodes increases. Here the power consumed by the aggregating node (dark columns) is far

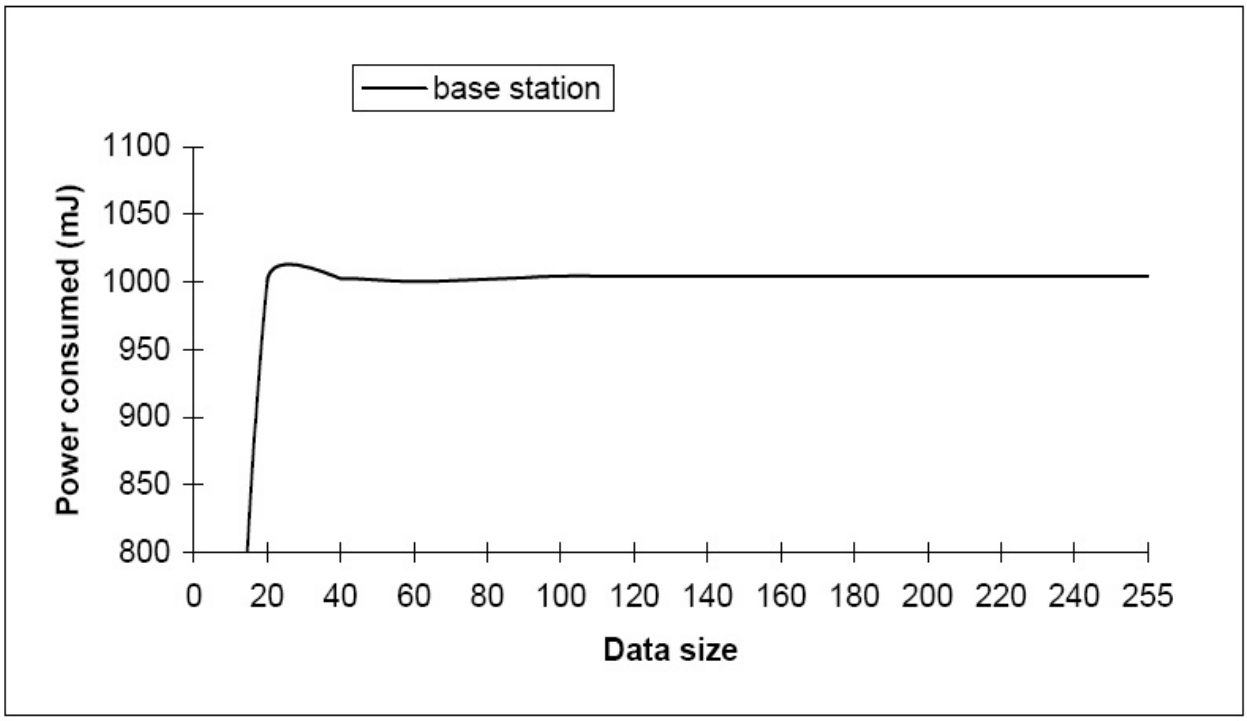

FIGURE 4.25. TestAgg: Power consumed by base station.

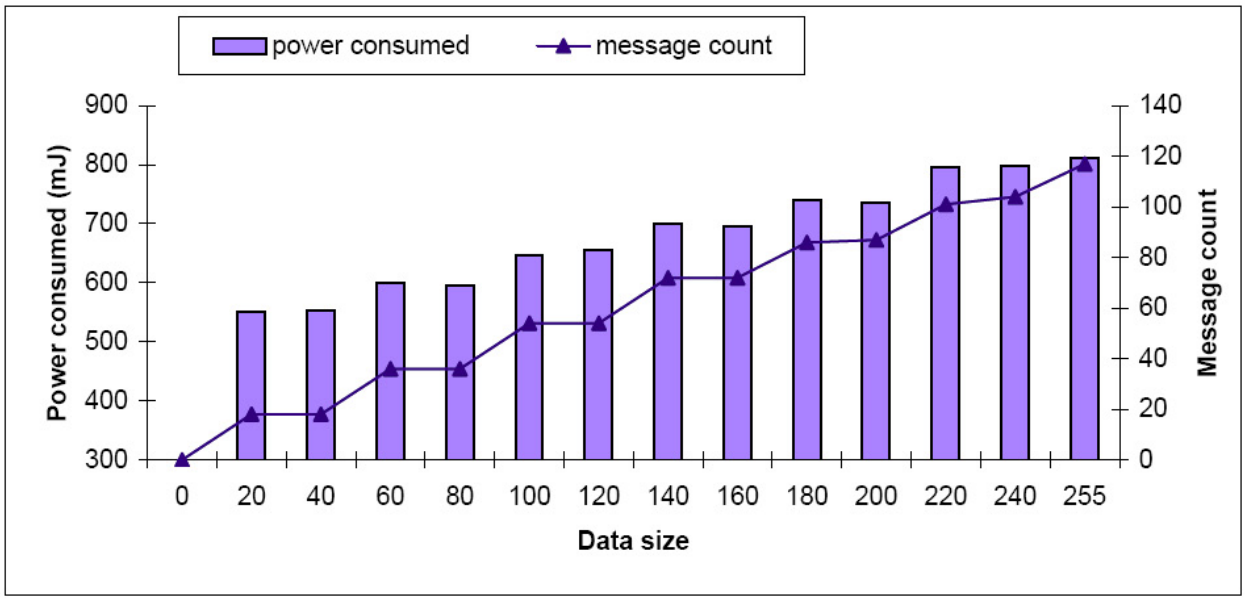

FIGURE 4.26. TestAgg: Power consumed by the source node for radio set at high.

higher than the remaining nodes. Here the power difference between the aggregating node and the remaining (forwarding) nodes is more pronounced as compared to the radio set at normal.

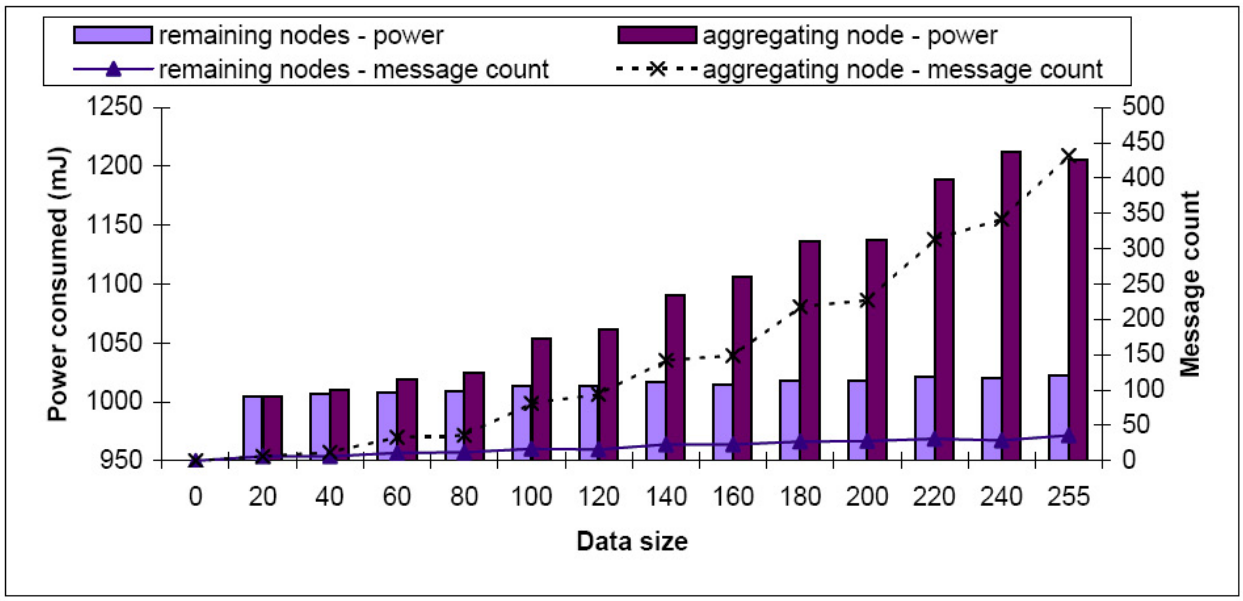

FIGURE 4.27. TestAgg: Power consumed by the remaining nodes for radio set at high.

The power consumed by the base station for the radio set at high is shown in Fig. 4.28. Here again the power consumed by the base station does not increase much and stays almost the same as that of the radio set at normal as the base station is in listen mode most of the time.

The overhead required for encoding and decoding the messages is the CPU cycles taken by nodes 2 and 0, i.e., the source and base station. This is negligible, as the current drawn by the CPU is just 4.13 mA. The power drawn by the CPU component stayed the same for all the nodes even if some of the nodes performed more activity, such as node 2 transmitting three times the number of messages, node 0 (base station) decoding and assembling the fragments and node 1 aggregating the data. Also it was observed that the power drawn by the CPU increases gradually from a data size of 0 bytes to 255 bytes - by about 2-3 mJ.

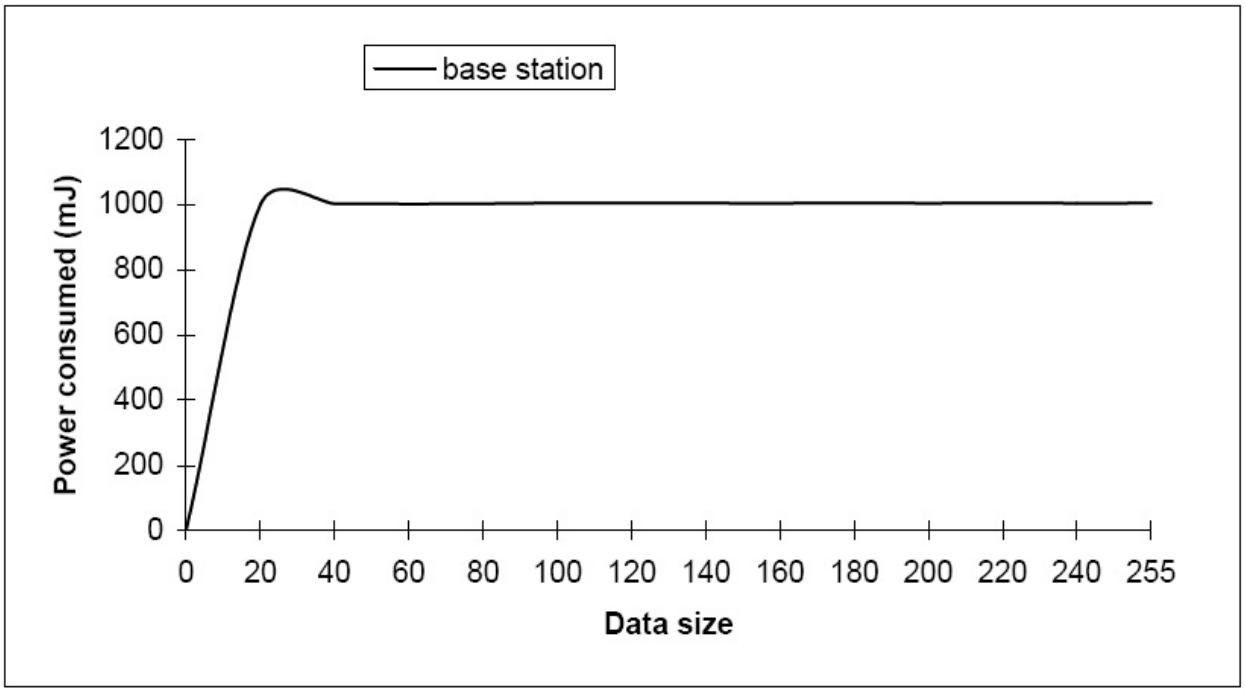

FIGURE 4.28. TestAgg: Power consumed by the base stationfor radio set at high.

CHAPTER 5

CONCLUSION

5.1. Summary of Results and Analysis

From the results obtained in Chapter 4 the following could be concluded:

(i) You can send more data for the same power consumed in 2-of-3 encoding and 3-of-4 encoding as compared to 1-of-3 and 1-of-4 respectively.

(ii) The power consumed by the lossy model is more than twice that of the simple model (about 700 - 800 mJ higher). This is almost 50-60% more than the power consumed by the simple model.

(iii) The power consumed by the lossy model increases as the distance between the nodes increases. This trend is visible in Test3Paths and Test4Paths.

(iv) Encoding with in-network aggregation consumes about 20 to 30 mJ more power compared to the one without it.

(v) The energy consumed by the source node increases with data size at a much greater rate then the remaining nodes.

The results showing the power consumption of the source node for both the radio models for different encodings are shown in Table 5.1. These are for the radio set at a normal transmit current. The values shown are the lowest and highest power consumed. All units are in mJ. It can be seen that the power consumed by 2-of-3 is less than that of 1-of-3 and the power consumed by 1-of-4 is greater than 3-of-4. Also the power consumed by the lossy model increases as the distance increases. The ranges shown here are for the maximum and minimum data sizes.

TABLE 5.1. Power consumption of different encodings for different radio models.

	Simple model	Lossy 10 feet	Lossy 15 feet.	Lossy 20 feet.
1-of-3	550 - 1100	1250 - 1640	1280-2090	1340 - 2100
2-of-3	530 - 690	1248 - 1440	1290 - 1610	1403 - 1773
1-of-4	625 - 822	1446 - 1780	1475 - 1972	1485 - 2073
3-of-4	626 - 687	1455 - 1547	1485 - 1592	1529-1670

TABLE 5.2. Power consumption of different encodings for radio current set at high.

	Normal (8.7 mA)	High (21.48 mA)
1-of-3	550 - 1100	551 - 1670
2-of-3	530 - 690	551 - 881
1-of-4	625 - 822	642 - 1009
3-of-4	626 - 687	644 - 755
Aggregation	532 - 717	550 - 810

The results showing the power consumption of the network when the radio current is set at high are shown in Table 5.2. The ranges shown here are the maximum and minimum power consumed.

The results on a per-node basis are summarized in table 5.3. These results are the average power consumed from a data size of 0 bytes to 255 bytes. The results show that for Test3Paths and Test4Paths, the energy consumed by all nodes other than node 2 (source node) is somewhat the same. For TestAgg, the power consumed by node 1 and 2 vary while that of node 0 is minimum. These results are for 2-of-3 and 3-of-4.

The total power savings obtained from using erasure encoding for various simulations is shown in table 5.4. This table shows the power savings of the source node only as the source node gets the maximum power savings.

TABLE 5.3. A comparison of energy consumed on a per-node basis for the 3 applications.

	Test3Paths		Test4Paths		TestAgg	
	Normal	High	Normal	High	Normal	High
Base Station	1004	1003	1002	1004	1004	1003
Source node	603	677	660	702	621	644
All remaining nodes	1004	1014	1004	1007	1004	1014

TABLE 5.4. Power savings obtained by using erasure encoding for various simulations.

	40-140 bytes	160-240 bytes
Simple radio model - 3 paths	8.46%	16.08%
Simple radio model - 4 paths	6.84%	13.13%
Lossy model 20 feet - 3 paths	8.82%	14.96%
Lossy model 15 feet - 3 paths	8.20%	14.99%
Lossy model 10 feet - 3 paths	4.74%	10.40%
Lossy model 20 feet - 4 paths	8.93%	18.05%
Lossy model 15 feet - 4 paths	7.23%	14.82%
Lossy model 10 feet - 4 paths	5.26%	10.84%
Radio at high - 3 paths	11.66%	21.45%
Radio at high - 4 paths	11.6%	21.15%

5.2. Future Work

From the results it is clear that by using 2-of-3 encoding and 3-of-4 encoding, you can send more data at the same power consumption rate. For 1-of-3 and 1 -of-4, the encoding proves to be costly. For the same power consumed, you can send twice or thrice the data size in bytes for 2-of-3 and 3-of-4 respectively. Conversely, for a fixed data size, less power is consumed by 2-of-3 and 3-of-4 encodings which leads to longer battery life. Other kinds of encodings similar to the ones in this thesis can be experimented with. An example of this being 1-of-2. In 1-of-2, you get the same resilience as 2-of-3, but have to transmit more

data into the network for a fixed data size as compared to 2-of-3. More elaborate encodings like Reed-Solomom codes [48], Tornado codes [5] etc. can be experimented with, and if the power required for performing computations is not too high, they will provide a means to send packets of much larger data sizes than the ones in this test module.

For future work, the applications described in this thesis can also be ported to TinyOS-2.0 which has a bigger set of multihop protocols and algorithms to which different encoding techniques can be applied. Also, the routing used in this thesis was many-to-one: all nodes communicate with the base station. Erasure encoding can also be applied to any-to-any routing or one-to-many routing. In this thesis, the routing table was hand-coded, but erasure encoding can also be applied in conjunction with different routing algorithms, e.g., directed diffusion, SPINS, etc., to increase the power efficiency.

BIBLIOGRAPHY

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. In IEEE Communications Magazine, pages 102–114, 2003.

[2] N. Alon and M. Luby. A linear time erasure-resilient code with nearly optimal recovery. In IEEE Transactions on Information Theory (special issue devoted to coding theory), volume 42, pages 1732–1736, November 1996.

[3] R.E. Blahut. Theory and Practice of Error Control Codes. Addison Wesley, 1984.

[4] John W. Byers, Michael Luby, and Michael Mitzenmcher. A digital fountain approach to reliable distribution of bulk data. In SIGCOMM, pages 56–67, 1998.

[5] J.W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: using tornado codes to speed up downloads. In Proc. of the IEEE INFOCOMM '99, pages 275–283, 1999.

[6] D.W. Carman, P.S. Kruss, and B.J. Matt. Constraints and approaches for distributed sensor network security. Technical report, NAI Labs, 2000.

[7] Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In IEEE Symposium on Security and Privacy, page 197, 2003.

[8] Chee-Yee Chong and Srikanta Kumar. Sensor networks: Evolution, opportunities and challenges. Proc. of the IEEE, 91:1247–1256, August 2003.

[9] B. Dahill, B.N. Levine, E. Royer, and C. Shields. A secure routing protocol for ad-hoc networks. Technical Report UM-CS-2001-037, Electrical Engineering and Computer Science , University of Michigan, August 2001.

[10] Jing Deng, Richard Han, and Shivakant Mishra. A performance evaluation of intrusion-tolerant routing in wireless sensor networks. In Proc. of the I.E.E.E. 2nd International Workshop on Information Processing in Sensor Networks, IPSN'03, LNCS 2634, pages 349–364, 2003.

[11] D.Liu and P.Ning. Efficient distribution of key chain commitments for broadcast authentication in distributed sensor networks. In Proc. of the 10th Annual Network and distributed System Security Symposium, pages 263–276, February 2003.

[12] W. Du, J. Deng, Y. Han, and P. Varshney. A pairwise key pre-distribution scheme for distributed sensor networks. In Proc. of the tenth ACM Conference on Computer and Communications Security (CCS 2003), pages 42–51, October 2003.

[13] Laurent Eschenauer and Virgil D. Gligor. A key management scheme for distributed sensor networks. In Proceedings of the 9th ACM Conference on Computer and Communication Security, pages 41–47, November 2002.

[14] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable co-ordination in sensor networks. In 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pages 263–270, 1999.

[15] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin. Highly-resilient energy-efficient multipath routing in wireless sensor networks. Mobile Computing and Communications Review, 4(5), October 2001.

[16] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In Proc. of the ACM SIGPLAN 2003 Conference on Programming Language Design and Inplementation, pages 1–11, New York, NY, USA, 2003. ACM Press.

[17] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Cooperative security for network coding file distribution. Technical Report MSR-TR-2004-137, Microsoft Research, 2004.

[18] Christos Gkantsidis and Pablo Rodriguez Rodriguez. Network coding for large scale content distribution. IEEE Infocomm 2005, 4:2235–2245, March 2005.

[19] J. Hill and D. Culler. System Architechture for Wireless Sensor Networks. PhD thesis, University of California, Berkeley, May 2004.

[20] Y.C. Hu, D.B. Johnson, and A Perrig. SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. In Proc. of the 4th IEEE Workshop on Mobile Computing Systems and Applications (WMCSA 2002), pages 3–13, June 2002.

[21] Y.C. Hu, A. Perrig, and D.B. Johnson. Ariadne: A secure on-demand routing protocol for ad-hoc networks. Technical Report TR01-383, Department of Computer Science, Rice University, December 2001.

[22] Y.C. Hu, A. Perrig, and D.B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, June 2002.

[23] C. Intanagonwiwat, R. Govindan, Deborah Estrin, J. Heidemann, and F. Silva. Directed diffusion for wireless sensor networking. IEEE/ACM Trans. Networking, 11:2–16, February 2002.

[24] C. Karlof, N. Sastry, Y. Li, A. Perrig, and J. Tygar. Distillation codes and applications to DoS resistant multicast authentication. In Proc. of the 11th Network and Distributed Systems Security Symposium (NDSS), San Diego, CA., February 2004.

[25] C. Karlof, N. Sastry, and D. Wagner. TinySec: A link layer security architechture for wireless sensor networks. In (to appear), Proc. of the second ACM Conference on Embedded Networked Sensor Systems (SenSys 2004), Baltimore, MD, November 2004.

[26] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. In Proc. of the First IEEE International Workshop on Sensor Network Protocols and Applications, pages 113–127, 2003.

[27] Brad Karp and H.T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In Proc. of the Sixth International Conference on Mobile Computing and Networking (MobiComm 2000), pages 243–254, August 2000.

[28] Maxwell N. Krohn, Michael J. Freedman, and David Mazieres. On-the-fly verification of rateless erasure codes for efficient content distribution. In IEEE Symposium on Security and Privacy, pages 226–241, Berkeley,California, 2004. IEEE.

[29] J. Kulik, W.R.Heinzelman, and H. Balakrishnan. Negotiation-based protocols for disseminating information in wireless sensor networks. Wireless Networks, 8(2-3):169–185, 2002.

[30] Joanna Kulik, Wendi Rabiner, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In Proc. of the Fifth Annual IEEE/ACM International Conference on Mobile Computing and Networking (MobiComm '99), 1999.

[31] Philip Levis. TinyOS Programming, February 2006.

[32] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In Proc. of the First ACM Conference on Embedded Networked Sensor Systems (Sensys), 2003.

[33] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Ambient Intelligence, chapter TinyOS: An Operating System for Wireless Sensor Networks. W. Weber and J. Rabaey E. Aarts, 2005.

[34] D. Liu and P. Ning. Establishing pairwise keys in distributed sensor networks. In Proc. of the Tenth ACM Conference on Computer and Communications Security (CCS 2003), pages 52–61, October 2003.

[35] M.G. Luby. LT codes. In The 43rd Annual IEEE Symposium on Foundations of Computer Science, pages 271–282, 2002.

[36] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman. Efficient erasure correcting codes. IEEE Trans. on Information Theory, 47(2):585–598, February 2001.

[37] M.G. Luby, M. Mitzenmacher, M.A. Shokrollahi, D.A. Spielman, and V. Stemann. Practical loss-resilient codes. Annual ACM Symposium on Theory of Computing, pages 150–159, 1997.

[38] P. Maymounkov. Online codes. Technical Report Tech.Rep. 2002-833, NYU, November 2002.

[39] P. Maymounkov and D. Mazieres. Rateless codes and big downloads. In Proc. of the 2nd International workshop on Peer-to-Peer Systems (IPTPS), February 2003.

[40] R. Merkle. Protocols for public key cryptosystems. In In Proc. of the IEEE Symposium on Research in Security and Privacy, pages 122–134, April 1980.

[41] J.M. Park, E.K.P. Chong, and H.J. Siegel. Efficient multicast stream authentication using erasure codes. ACM Trans. Inf. Syst. Secur., 6(2):258–285, 2003.

[42] S. Park, S. Savvides, and M. Srivastava. Simulating networks of wireless sensors. In Proc. of the 2001 Winter Simulation Conference, pages 1330–1338, 2001.

[43] A. Perrig and H. Chan. Pike: Peer intermediares for key establishment in sensor networks. In Proc. of the IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies., volume 1, pages 524–535, March 2005.

[44] A. Perrig, R.Szewczyk, V. Wen, D. Culler, and J.D. Tygar. SPINS: Security protocols for sensor networks. Wireless Networks Journal (WINET), pages 8(5):521–534, 2002.

[45] Adrian Perrig, John Stankovic, and David Wagner. Security in sensor networks. Communications of the ACM, 47:53–57, June 2004.

[46] B. Przydarek, D. Song, and A. Perrig. SIA:secure information aggregation in sensor networks. In ACM Sensys 2003, November 2003.

[47] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In Proc. of the 9th Annual International Conference on Mobile Computing and Networking, pages 96–108, 2003.

[48] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. J. Soc. Indust. Appl. Math., 8:300–304, 1960.

[49] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. In Communications of the ACM, volume 21, pages 120–126, 1978.

[50] L. Rizzo. Effective erasure codes for reliable computer communication protocols. ACM Computer Communication Review, 27(2), April 1997.

[51] V. Shnayder, M. Hempstead, B. rong Chen, G.W. Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In Proc. of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys), pages 1–12, New York, NY, USA, 2004. ACM Press.

[52] M. Amin Skokrollahi. Raptor codes. Technical Report DF2003-06-001, Digital Fountain Inc., June 2003.

[53] D. Spielman. Linear-time encodable and decodable error-correcting codes. In IEEE Transactions on Information Theory (special issue devoted to coding theory), volume 42, pages 1723–1731, November 1996.

[54] Ronald Watro, D. Kong, Sue fen Cuti, J. Mulligan, Charlie Gardiner, and Dan Coffin. TinyPK: securing sensor networks with public key technology. In Proc. of the 2nd ACM workshop on Security of ad-hoc and sensor networks, pages 59–64, 2004.

[55] A. Woo and D.E. Culler. A transmission control scheme for media access in sensor networks. In Proc. of the 7th International Conference in Mobile Computing and Networking (MobiComm 2001),ACM Press, pages 221–235, 2001.

[56] A. Woo, T. Tong, and D. Culler. Taming the underlying challeges of reliable multihop routing in sensor networks. In Proc. of the First ACM Conference on Embedded Sensor Systems (SenSys), pages 14–27. ACM Press, 2003.

[57] A. Wood and J.A. Stankovic. Denial of service in sensor networks. IEEE Computer, 35(10):54–62, October 2002.