A Sleep-Scheduling-Based Cross-Layer Design Approach for Application-Specific Wireless Sensor Networks

by

Rick Wan Kei Ha

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2006

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

ABSTRACT

The pervasiveness and operational autonomy of mesh-based wireless sensor networks (WSNs) make them an ideal candidate in offering sustained monitoring functions at reasonable cost over a wide area. To extend the functional lifetime of battery-operated sensor nodes, stringent sleep scheduling strategies with communication duty cycles running at sub-1% range are expected to be adopted. Although ultra-low communication duty cycles can cast a detrimental impact on sensing coverage and network connectivity, its effects can be mitigated with adaptive sleep scheduling, node deployment redundancy and multipath routing within the mesh WSN topology. This work proposes a cross-layer organizational approach based on sleep scheduling, called Sense-Sleep Trees (SS-Trees), that aims to harmonize the various engineering issues and provides a method to extend monitoring capabilities and operational lifetime of mesh-based WSNs engaged in wide-area surveillance applications. Various practical considerations such as sensing coverage requirements, duty cycling, transmission range assignment, data messaging, and protocol signalling are incorporated to demonstrate and evaluate the feasibility of the proposed design approach.

Keywords – wireless sensor network, energy efficiency, sleep scheduling, cross-layer design, Sense-Sleep Trees, implicit acknowledgements, integer linear programming, network flow model, transmission range assignment

ACKNOWLEDGEMENTS

There are a number of individuals and institutions that I would like to express my deepest gratitude in making my endeavour in graduate studies such a relished experience. They are:

- My supervisors, Dr. Pin-Han Ho and Dr. Sherman Shen, for providing me with valuable advice and superb guidance in the research process,
- My committee members, Dr. Liping Fu, Dr. Sagar Naik, Dr. Ajit Singh, and Dr. Hossam Hassanein, for contributing their valuable time and effort in perfecting my work and serving in my thesis committee,
- My supervisor of the master program, Dr. Weihua Zhuang, for recommending me for PhD studies at University of Waterloo,
- University of Waterloo and her many fine educators and staff, for enabling me to pursue top-quality undergraduate and graduate education at such an enlightening environment,
- And finally, family, friends and colleagues, for showing me a life outside of books, Internet and research.

DEDICATION

To Karen (Ah Wai), my soul (sole) mate in life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Recent technological advances in wireless sensor networks (WSNs) give inspiration to the development of a new breed of monitoring systems for various physical phenomena that can offer additional capability and flexibility over existing options at reasonable costs. In the past several years, WSNs have received considerable attention by the research community and a large number of related research proposals have been put forth [1] [3] [4] [6] [70]. In some cases, WSNs are envisioned to be deployed in dynamic situations such as military operations and disaster relief, where the emphasis is on effective self-coordination and survivability over a random topology. On the other hand, some WSNs may be used in more static applications such as environmental monitoring and surveillance, where the focus is more on maximizing system lifetime over a pre-determined network configuration. Because of their flexibility in topological organization, WSNs are often compared to mobile ad hoc networks (MANETs), which are also attracting intense research interest in recent years.

Although both MANETs and WSNs are based on wireless technologies with dynamic topologies, there exist quite a number of differences between the two as listed below:

1. MANET applications such as military commando operations and disaster zone search teams physically alter the network topology as they tend to involve a lot of mobility. As WSN sensor nodes are generally static for recording environmental data within a specific area, topological changes are generally caused by nodes that have entered sleep mode for energy conservation, or have ceased functioning due to drained battery or some other system failure.

2. While energy consumption concerns are the focal point for both types of networks, the application lifetime for MANETs seems to be much shorter than those of WSN nature. For example, a MANET for emergency response communication may be in operation for several hours to several days, while a WSN for environmental monitoring may last for a number of years. Also since MANET devices have closer ties with human operators, energy replenishment is more readily available.

3. The direction of data flow within WSNs is simple; for the most part, any data generated by the sensor nodes will be forwarded directly to the data sink, whereas control packets from the data sink to the rest of the sensor nodes could be of unicast, multicast or broadcast types. In contrast, individual nodes within a MANET may initiate data connections to any of the other nodes.

4. WSNs typically have a much higher nodal density that is several orders of magnitude over MANETs. Because of the large number of sensor nodes, WSNs may not implement unique global identification as the overhead in assignment and management would be stupendous.

Since WSNs are often deployed in remote or hostile areas where provisions for energy replenishment and sensor node replacement are tremendously difficult to realize, the primary concern in WSN design is their collective *energy efficiency*. As the operational reliability of each node is completely compromised once its energy is depleted, the highest priority in WSN design is to prolong system lifetime as much as possible through the use of energy-efficient hardware components and power management techniques. Also, since each WSN is expected to be comprised of thousands of nodes or more, per unit *cost* hence becomes a major factor in sensor node design and component selection. While sensor nodes are forecasted to cost pennies each to manufacture as technology advances and scale of production increases, any minute increase in production and deployment cost will aggregate into a sizeable sum that lays impact to the overall budget due to the sheer number of the nodes involved. A balance must be struck between choosing low-cost yet perhaps inferior components and maintaining sensing application reliability without making WSNs excessively expensive.

On another front, reliability has always been one of the top concerns in communications system design, and WSNs are no exception. In light of the influences of energy efficiency and cost, a few primary implications of reliability in WSN design are as follows:

- *Hardware Reliability* - This measure is related to the propensity of the onboard hardware components in succumbing to failure during normal WSN operations. While the specifics

on WSN hardware design for maximizing reliability are beyond the scope of this work, engineering intuition suggests that it is a good idea to select hardware components that are as simple in architecture as possible.

- *Sensing Reliability* - In WSN applications, all sensor nodes cooperate together to monitor physical phenomena of interest across the sensing field. As individual nodes can sense the appropriate physical phenomena within their sensing range only, important events may be missed by all sensor nodes because of possible inadequate sensing coverage. Therefore, providing comprehensive sensing coverage requires meticulous network planning and node deployment strategies. Redundant nodes may be introduced to the sensing field to offer additional reliability for sensing coverage, though trade-offs must again be weighed with respect to per node costs and network management complexity.

- *Communication Reliability* - In most WSN applications, the overall traffic profile is very simple as packets only flow from sensor nodes to the data sink and vice versa, with very few inter-node exchanges. Despite this simplicity in traffic flows, the WSN is still expected to deliver sensor data and network control messages with high fidelity in a timely fashion. Aside from packet loss effects presiding over unstable wireless links, the inherent multihop nature of WSN communications present additional uncertainty in guaranteeing packet transport reliability that the common protocols used in the Internet paradigm are inadequate to handle. In particular, it remains a considerable challenge to preserve the network connectivity in conjunction with low duty cycle sleep scheduling strategies intended for maximum energy conservation.

1.1 SUMMARY OF RELATED WORK

Without enacting any energy saving technique during WSN operations, the radio transceiver would typically consume more energy than any other hardware component onboard a sensor node. Aside from avoiding the use of energy-inefficient and complex hardware components, sizable energy savings can be achieved through aggressive power management strategies in devising adaptive sleep schedules at the MAC layer to minimize the amount of energy lost due to needless transceiver idle listening [2] [16]. While the main implication of sleep scheduling at the

MAC layer is the shortening of the time the radio transceiver is engaged in idle listening, incidences of overhearing can also be reduced as sleeping nodes are no longer eavesdropping on the wireless medium.

For routing algorithms, however, link table entries would expire prematurely if an intermediate node sleeps and shuts off all links to its neighbours without prior notification, thereby forcing frequent packet reroutes. In addition, network maintenance functions such as neighbourhood discovery and time synchronization must also operate within the short time frame that the transceiver is active, thereby competing for bandwidth and processing power with other data reporting functions. In the application layer, real-time data reporting functions are subject to constant and debilitating routing path breakages due to random sleeping nodes, while the spontaneity and variability of data generation in event-driven monitoring applications place undue demand on sleep scheduling effectiveness. Because of these far-reaching effects, a cross-layer perspective should be taken in devising sleep schedules such that the various inter-layer issues can be tackled collectively [13].

Besides sleep scheduling, precious sensor node battery power can be saved by topology control measures that can dynamically adjust the radio transmission range of individual sensor nodes to balance energy efficiency, while maintaining adequate network connectivity [33] [54]. However, the energy saving benefits of such topology control techniques can be offset by the messaging and processing overhead in determining the minimum transmission range per node, especially in the event-driven WSN operating environment characterized by low data rates and long sleep periods. A third approach in reducing energy use for WSN communications is through power-aware routing where packets are routed onto paths with the most residual energy [76] [77]. This is to keep the best network connectivity and to increase the total transmission capacity. Other cost metrics, such as inter-node distances, transmission delays, channel conditions and route hop count, can also be taken into account when determining the minimum cost route across the WSN. One major drawback of this approach is the additional overhead required in disseminating the residual energy or cost information across the WSN for routing updates, which can be much reduced by exploiting the simple traffic profile and low data rates inherent in WSN applications.

For example, a legitimate organizational method for WSNs to minimize upstream and downstream communication costs is to interconnect all the nodes with a large spanning tree structure that is rooted at the data sink. Forwarding messages in a shortest path spanning tree has the advantage of locating a lowest cost path between each of the nodes and the data sink, which enables minimum cost forwarding and source routing to perform effectively [17] [18]. Also, junction points are ideal locations for performing data aggregation and in-network processing to reduce upstream traffic volume. Clustering is the best known example of utilizing the tree structure for WSN formulation [11] [21], though it requires denser connectivity for proper cluster hierarchy formation. The star topology is also an example of a tree structure where all of the nodes are leaves connected to the data sink via 1 hop, which clearly is not applicable to all types of WSN topologies. A linear chain, used in chain-based routing protocols [5] [9], is a special case of a tree where the number of descendents per node is 1.

Because sleep scheduling is an integral part of WSN design, compatibility issues of spanning tree management and sleep scheduling should be investigated with prudence. Random sleep scheduling is not recommended because it would exert a detrimental effect on network connectivity and topology maintenance efficiency under an ultra-low duty cycle (i.e. less than 1% active time per sleep cycle). On the other hand, while implementing a global coordinated sleep schedule for all the nodes is feasible on a spanning tree structure, a network-wide communication blackout exists during the long sleep periods where none of the nodes would be active for packet forwarding. This lull in communication will adversely impact the monitoring effectiveness in temporal coverage (i.e. timely reporting of emergency events). One possible solution is to reduce the time scale of sleep schedules such that the length of each sleep cycle is shortened while maintaining the same transceiver duty cycle (e.g. 1 millisecond active time per 1 second vs. 1 second active time per 1000 seconds for a duty cycle of 0.1%). However, additional control overhead and hardware cost may be incurred in keeping time synchronization tighter [56]. Also, a shorter sleep cycle may cause multihop packet exchanges to span over multiple active periods, thereby complicating routing procedures and lowers communication reliability.

Another way to shorten the communication blackout period while maintaining the percentage of sleep time is via forming a spanning tree with a large number of leaves such that the non-leaf

nodes, often referred to as the *connected dominant set* (CDS), form a virtual backbone [57]. In the graph theory terminology, a connected dominating set in a graph is a set of connected vertices such that every vertex in the graph either is in the set or has a neighbour in the set. By finding the minimum CDS (MCDS), the entire WSN can theoretically assign the fewest number of active nodes as the virtual backbone and groups of leaf nodes can be then turned on and off successively to provide interleaved coverage while minimizing energy usage through regular sleep scheduling. The main concern with this approach is that it requires the nodes belonging to the MCDS to remain in active mode for longer periods of time to accommodate the varying sleep schedules of the leaf nodes, thereby depleting their battery reserves much sooner.

Given the prevalence of mesh-based topologies in WSN applications, multiple disjoint CDSs (DCDS) can be identified so that through rotating the sleep and active times of all DCDSs, only one DCDS is needed to remain active at any given time to shoulder the communication and sensing responsibilities for the entire WSN. The base problem of partitioning all the nodes into disjoint dominating sets, not necessarily connected, is called a *domatic partition*, and the maximum number of disjoint dominating sets for a given topology is called is *domatic number*. Through maximizing the domatic number for a given WSN topology, each node can enjoy the most sleep time and therefore maximum energy efficiency while both network connectivity and sensing coverage can be preserved. The work in [59] showed that every graph has a domatic partition with $(1-o(1))(\delta+1)/\ln n$ dominating sets, where δ and n denote the minimum degree and number of vertices, respectively. The authors in [31] applied this concept to WSNs and proposed approximation algorithms to maximize system lifetime in a clustered environment.

Even though this new perspective on combining sleep scheduling and connectivity is promising, a couple of critical issues deserve further investigation. First, the original problem of maximum domatic partition (MDP) for a given graph does not require the nodes within each dominating set to be connected. Therefore, the results of prior work are only applicable to WSN design if the entire network is fully connected or arranged in a star topology around the data sink, both of which are not practical in most cases. Secondly, extending the MDP results for a connected topology into a maximum connected domatic partition (MCDP) solution is much more difficult than constructing a MCDS out of a minimum dominating set. Since the complexity of the MDP

problem is already known to be NP-Complete [80], solving the MCDP problem would prove to be even more complicated.

No matter how many nodes are to remain active at any given time in a sleep schedule, a minimum level of sensing coverage should be maintained for reliable sensing according to the application requirements. Many prior studies on the WSN connectivity, especially those stemmed from MANET research such as the work in [74], have paid little attention on the importance of guaranteeing sensing coverage. On the other hand, redundant nodes can be deployed onto the sensing field to compensate for the sleeping nodes as well as to offer additional sensing reliability against node failures [50]. The problem of determining the minimum number of active sensor nodes to provide a certain degree of sensing coverage alludes to the classic set cover problem and its variants [47], though network connectivity should also be considered to form a minimum connected sensor cover [23]. Therefore, both sensing coverage and network connectivity should jointly be considered when formulating an optimal sleep schedule that can balance all the application requirements [29] [79].

1.2 Research Objectives and Thesis Highlights

Summarizing all the design concerns pertinent to guaranteeing WSN operability and reliability, the real engineering challenge henceforth is to devise a comprehensive yet manageable network organization and communication paradigm that can harmonize all of the criteria without creating significant conflicts in optimization objectives. The prime research objectives are:

- Investigate the design issues regarding wireless sensor network applications primarily for event-driven data reporting.
- Emphasize mainly on energy efficiency for extending system lifetime over multiple years.
- Jointly consider cross-layer issues in sleep scheduling, data communications, sensing coverage, and energy efficiency.

The major application-specific assumptions made throughout the proposed design approach are as follows:

- The sensor nodes are immobile and battery-operated, and they are all identical in functionality.
- The sensor nodes report to a single data sink on the sensing field.
- The transmission range of each node is determined during network planning stage and is fixed after deployment.
- There exists a low degree of spatial coverage overlapping among the sensor nodes (see Section 2.4.2 for further description).
- Event-driven data reporting is the main focus in sensing, though timer-driven and request-driven types should also be accommodated (see Section 2.4.3 for further description).
- The expected traffic load is low over time, most of which consists of periodic and coordinated status updates and network maintenance packets.

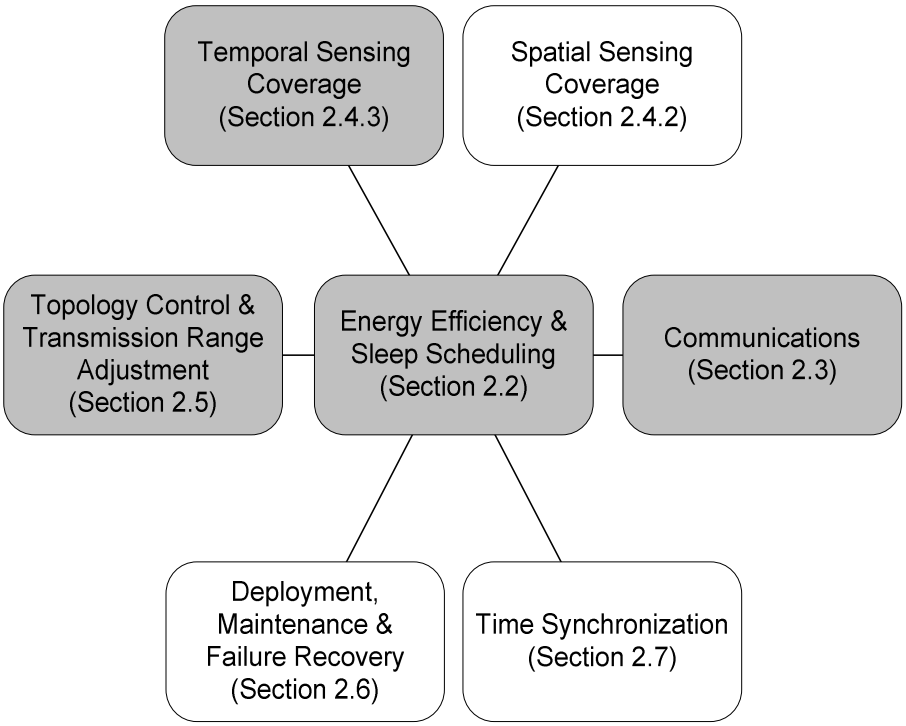

Figure 1 - Important WSN design concepts discussed in this work.

The ultimate design goal is to balance, through a cross-layer organization scheme, the sensing requirements, end-to-end data communication overhead, and network control effectiveness with energy efficiency. The main contributions of this thesis are:

- Energy consumption model with respect to sleep scheduling (see Section 2.2.2).
- Timing analysis for different data reporting types (see Section 2.4.3).
- Relationship between transmission range and expected hop count in multihop communications (see Section 2.5.2).
- The Sense-Sleep Tree (SS-Tree) concept (see Chapter 3, specifically Sections 3.1, 3.2 and 3.3).
- The implicit acknowledgement (IACK) scheme (see Section 3.4.3).
- Performance evaluations of the proposed cross-layer approach (see Chapter 4).

The rest of the thesis is organized as follows. Chapter 2 describes related concepts, illustrated in Figure 1, in achieving energy-efficient and reliable WSN design. While all of the design considerations revolve around energy efficiency and sleep scheduling, note that the greyed concepts in Figure 1 are the focus of the this thesis work. Next, Chapter 3 proposes a new organizational methodology, called Sense-Sleep Trees (SS-Trees), that aims to maximize energy efficiency in WSN design. Evaluation on the validity and effectiveness of the proposed SS-Tree computation approach is provided in Chapter 4. Chapter 5 gives some concluding remarks and future research outlook.

2 WSN DESIGN CONSIDERATIONS

The following subsections will detail the building blocks of a WSN, as well as fundamental design considerations with respect to achieving energy and cost savings.

2.1 WSN COMPONENTS AND SYSTEM HIERARCHY

The basic element of a WSN is the sensor node, where it is consisted of three main functional components that separately deliver sensory, communication and processing capabilities. Figure 2 illustrates some of the associated subcomponents in a typical sensor node [1]. As the primary purpose of a sensor node is to collect environmental data, each is fittingly embedded with a sensory component that is attuned to its specific data type, such as acoustic, chemical, biological, motion, or nuclear, all depending on the nature of the WSN. The actual design principles of the sensory components are beyond the scope of this work, but it is assumed that each sensor node is capable of gathering environmental data with minimal energy usage. Also in collecting such environmental data, the sensor nodes often risk sustaining damage from heat, water, dirt and other external factors. Therefore, the sensing components and circuits should be encased in some protective packaging to fend off any foreign substance penetration and disturbance. Because of the difficulty and cost of sensor node replacement in face of battery drainage or system failure, hardware reliability is also another major concern in WSN design.

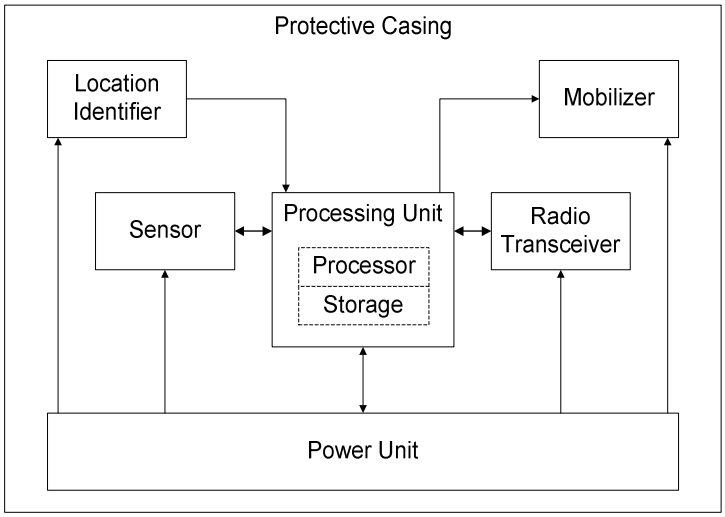

Figure 2 - Sensor node components.

The processing unit functions as the heart of the sensor node, where it processes incoming data and manages the operation of peripheral components. Because of energy and cost constraints, the capabilities of the onboard processor and storage memory would be severely restricted as the suitable processing components for WSN applications typically reside on the lower end of the performance scale. In addition, energy-saving measures such as dynamic voltage scaling and processor sleeping could be followed, thereby further limiting the overall available processing power. The challenge lies on the design of an operating system (OS) software that is able to accomplish energy efficiency and processing competency in face of such frugal circumstances. Further discussion on this issue and information on the development of TinyOS, an OS specifically designed for WSN applications, can be found in [82].

During WSN operation, the collected environmental data will undergo some degree of processing and may be forwarded to other neighbours via the onboard low-power wireless transceiver. Technology has advanced to the point that the amount of power consumed per bit for a single transmission is equivalent to processing the same bit thousands of times [6]. Likewise, the power consumed in signal reception is often comparable to the amount of energy used in transmission, where it is lost through signal amplification, noise reduction, filtering, and other signal processing tasks. Because of the high energy costs of wireless communication via the radio transceiver compared with executing instructions by the processing unit in the sensor node, the general tenet in coordinating sensor nodes is to reduce the amount of transmitted data as much as possible. Rather, most of the information should be processed locally instead of relaying back to the central processing centre. If inter-node communication is to be performed, techniques such as channel coding and transmission diversity are to be considered in order to negotiate the unique characteristics of the wireless transmission medium.

Other auxiliary functional components include location identifier, which is accomplished possibly through GPS or radio triangulation methods, and mobilizer, which allows the sensor unit to roam around the sensing field. The inclusion of these subunits depends on the WSN application and economic constraints. Installing a GPS transceiver for every sensor node would provide flexibility and accuracy in determining geographical coordinates. However, for immobile sensor nodes, the GPS transceiver could be seldom used throughout the nodal lifetime.

This fact makes the inclusion of GPS transceivers in such sensing applications unwise as it increases production cost, energy consumption and circuit complexity. On the other hand, location estimation through radio wave ranging and referencing is a promising technology [24] [26], but again it consumes additional energy and increases costs. Therefore, hard-coding the geographical coordinates upon deployment could be more energy and cost efficient than the other methods in location estimation, which requires the network topology to be well planned beforehand.

Since the sensor nodes often operate in remote areas with little possibility of battery replacement and few self-contained power generation options, achieving power supply longevity becomes the top priority in WSN design. Without the blessings of replenishable power sources such as solar cells and piezoelectric generators, equipping the sensor node with a large initial reservoir of battery power would seem to be a simple solution. Unfortunately, progress in battery research falls far short of the torrid pace experienced elsewhere in the high-tech realm. Instead of following the famous Moore's Law of doubling processing power per unit area every 18 months, emerging technologies only double battery power per unit volume every 5 to 20 years [37]. Also, some of the high-capacity battery types are made of toxic materials that are harmful to the environment, which render them unsuitable for outdoor WSN applications. With limited options in boosting initial battery power, system lifetime can only be extended through aggressive power management schemes in reducing energy expenditure [4].

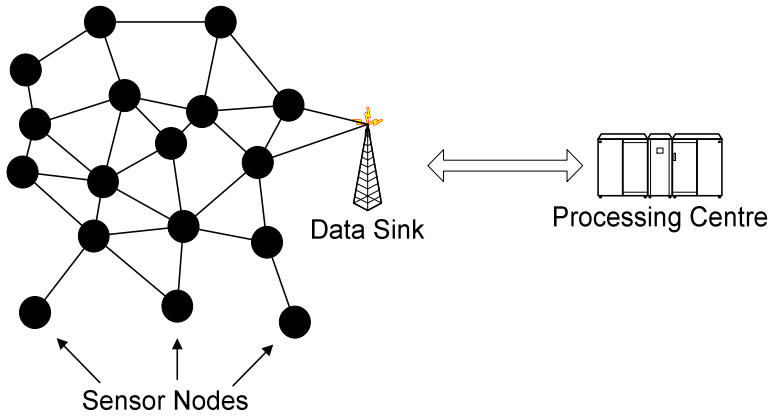

Figure 3 - Basic WSN architecture.

Above the node hardware level, a typical WSN is consisted of three hierarchical entities: sensor nodes, data sinks and the processing centre, as illustrated in Figure 3. Large numbers of sensor nodes are scattered over the target field to collect environmental data and forward it to the processing centre for further computations and interfacing with human operators. Due to the sheer magnitude of nodal deployment in WSNs, maintaining a flat hierarchy to channel data exchanges is not viable since the hop count between sensor nodes and the processing centre would be too large. Therefore, data sinks act as an intermediary between sensor nodes and the processing centre, much like the relationship in cellular systems between mobile devices, base stations and switching centre. They bridge the link between sensor nodes and the processing centre by providing higher communications capabilities and larger power supply. The role of the data sink can be performed by a satellite, fixed base station or a common sensor node elected as the cluster head. In smaller WSN applications such as perimeter surveillance and smart homes, data sink and processing centre can be integrated into a single physical entity.

In the uplink direction, the data sink gathers all of the data forwarded by the sensor nodes and may perform some preliminary processing to remove data redundancy and increase transmission efficiency. On the downlink, the data sink simply forwards whatever commands and messages issued by the processing centre onto the sensor nodes. Since the data sinks should handle more responsibility in data transmission and manipulation, they should be connected directly to the power grid, equipped with more battery power, or driven by replenishable sources such as solar panels and hydroelectric generators. For outdoor WSNs, data sinks should be constructed on vantage points such as hilltops, ridges and tall trees to maximize radio link coverage. For easier service maintenance, data sinks should be situated near roads or other existing infrastructure if possible.

To provide a large area with ubiquitous monitoring coverage, a sizable number of sensor nodes should be uniformly scattered across the sensing field. In collecting the generated environmental data, it is not a good idea to let all of the sensor nodes to report to only one data sink because of potentially crippling traffic volume and the undesired creation of a single point of failure. On the other hand, for a WSN with uniform probability of data generation at each sensor node, the nodes closest to the sink (i.e., nodes with fewest hop count from the data sink) will handle the

majority of the traffic forwarding. Therefore they will deplete their energy resource much sooner than the farthest nodes. Energy saving methods must be devised to prolong the lifetime of nodes in close physical proximity to the data sink. Boosting battery power on these sensor nodes before deployment can be a simple remedy, but it may increase production costs. Another possible solution is to install additional data sinks so that the overall traffic flow can be more distributed, as shown in Figure 4. Multiple data sinks can be interconnected via wired or wireless point-to-point links, where in turn each is connected to the processing centre through wired, point-to-point or satellite means. In essence, two tiers of wireless networks are created, one connecting sensor nodes and the other for data sinks. A third way to extend battery power is to increase nodal density surrounding each data sink, where several co-located nodes share the same communication responsibilities that are active in separated time slots according to an alternating sleep schedule, as shown in Figure 5. Issues regarding sleep schedules for sensor nodes will be discussed later on in Section 2.2.

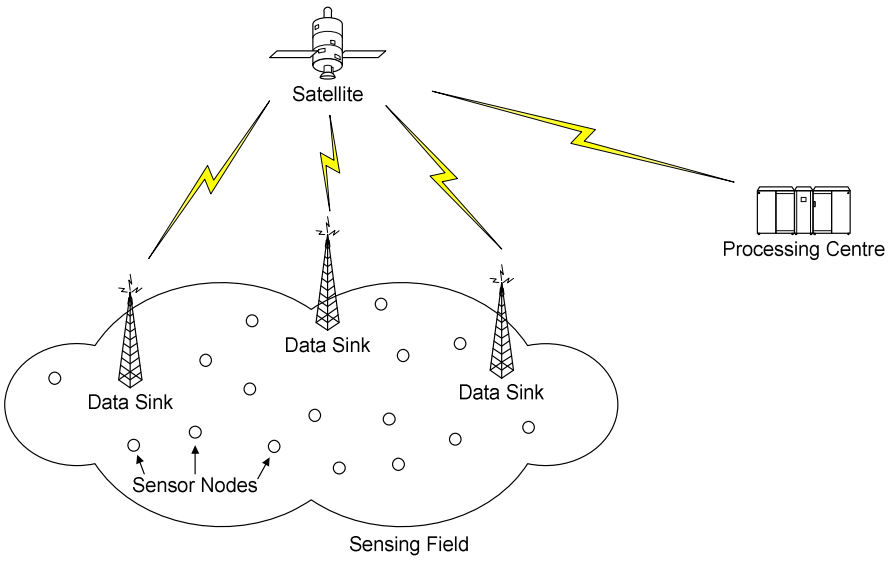

Figure 4 - WSN configuration with multiple data sinks.

The actual location of the data sink within the sensing field contributes to the overall energy consumption profile of the WSN [38]. Placing the data sink at the edge of the sensing field increases the average hop count to the sensor nodes, thus expending extra energy and increasing delay in delivering the packets. Therefore, each data sink should be placed directly in the middle

of the sensing field, if possible, in order to minimize hop counts and conserve energy. Figure 6 provides an example that illustrates the difference in average hop count in relation to the location of the data sink in a sensing field of 24 nodes arranged in a square mesh, where each sensor node has up to 8 neighbours. Figure 6(a) and (b) place the data sink, coloured in white, at the corner and in the middle of the sensing field, respectively. Each sensor node, coloured in black, is associated with a number which denotes its hop count along the shortest path to the data sink. Suppose that each sensor node generates a single message and it is to be forwarded to the data sink via neighbouring nodes on the shortest path. Note that multiple shortest paths may exist between the source node and the data sink with the same hop count. The average number of messages forwarded per node in the first case is 2.91, which is significantly higher than the 1.67 figure in the second case.

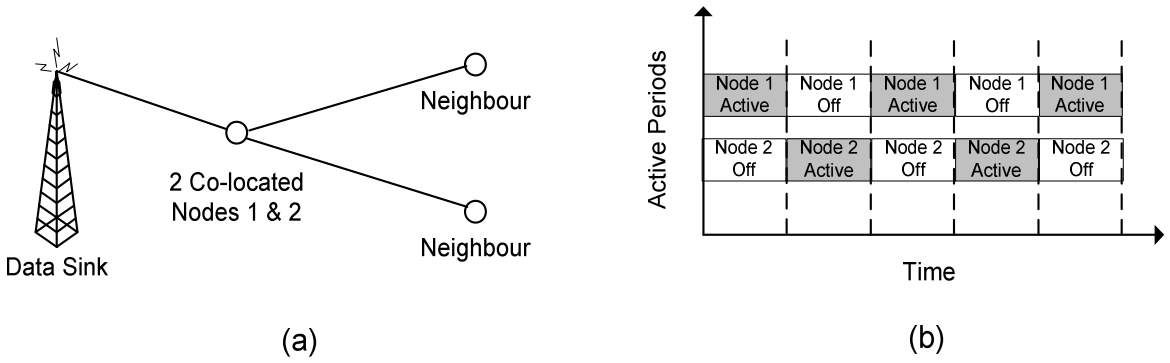

Figure 5 - Co-located nodal coordination.
(a) Diagram. (b) Sleep schedule.

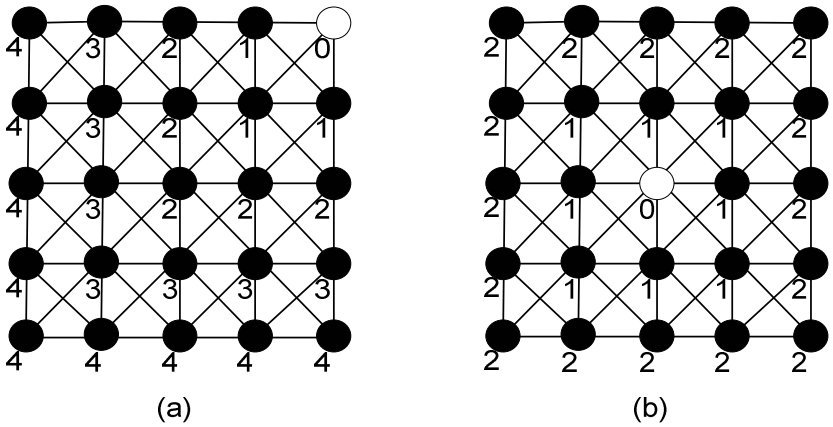

Figure 6 - Data sink placement in sensing field.
(a) At corner. (b) In the middle.

2.2 *Energy Efficiency and Sleep Scheduling*

2.2.1 Sleep Scheduling Considerations

The notion of sleep scheduling has been briefly discussed in Section 2.1 as a way to reduce nodal energy consumption by turning off any redundant sensor node that is not participating in active packet delivery sessions. A practical sleep schedule would selectively activate or deactivate certain components depending on the sensing task at hand, and Table 1 shows the useful assignment of sleep states [32]. In typical WSN implementations, the radio transceiver is often the most energy consuming hardware component, which suggests wireless communication should be conducted as sparingly as possible (i.e., minimize time spent in sleep states S_4, S_3 and S_2). However, the loss of communication capabilities not only affects network connectivity in the multihop WSN environment but also reduces the overall sensing coverage, which will be discussed in Section 2.4. For event-driven and timer-driven data reporting, collected sensor data cannot be timely forwarded to the data sink whenever nodes along the routing path have their transceivers turned off. Likewise, specific data requests issued by the data sink may not be delivered to the targeted nodes promptly due to prolonged sleep periods of intermediate nodes.

Sleep State	Clock	Processor	Memory	Sensor/A-to-D	Transceiver
S_4	active	active	active	active	Tx, Rx
S_3	active	idle	sleep	active	Rx/listen
S_2	active	sleep	sleep	active	Rx/listen
S_1	active	sleep	sleep	active	sleep
S_0	active	sleep	sleep	sleep	sleep

Table 1 - Useful sleep state assignment.

On the other hand, in single-hop or clustered WSN environment where network connectivity is usually dense, prolonged sleep periods of individual nodes have little effect on the packet forwarding capability assuming the data sink and clusterheads have extended power supply readily available. Therefore the nodes can operate in either sleep state S_0 or S_1 for most of the time to minimize energy loss through transceiver activity. Nevertheless, realizing dense network connectivity requires the support of longer transmission range relative to the physical distribution

of the nodes, which entails greater energy consumption due to higher transmission power as well as increased likelihood of packet collision and overhearing in single-channel transmission.

Besides sleep scheduling, other approaches to minimize energy consumption include topology control and power-aware routing [33]. The former deals with finding the optimal transmission range of individual nodes, often dynamically after deployment, such that the resultant network topology is the most energy-efficient, while the latter is associated with computing optimal packet delivery routes across the network based on communication costs and residual energy information such that overall system lifetime can be maximized. Even though both approaches offer some energy savings, energy loss due to transceiver idle listening still dominates in typical WSN applications that are characterized by prolonged monitoring and low data traffic if sleep scheduling is not implemented. Furthermore, such energy savings are achieved through extra hardware features such as transceivers with dynamically adjustable transmission power and fine-grained battery level monitors, implying additional design trade-offs in cost-conscious WSN applications.

Because ultra-low duty cycle sleep scheduling casts a far-reaching impact over different aspects of WSN design, a cross-layer methodology that jointly considers MAC design, routing strategies and application requirements should be advocated. While the main implication of sleep scheduling in the MAC layer is the shortening of the time the radio transceiver is engaged in idle listening, incidences of overhearing can also greatly be reduced as sensor nodes in sleep mode are no longer eavesdropping on the wireless medium. However, the effect of sleep scheduling on packet collision is uncertain because although sleeping nodes obviously do not cause packet collisions, the same volume of data traffic is to be rerouted on the remaining active sensor nodes, thus increasing the likelihood of packet collisions. For routing algorithms, link table entries will expire prematurely if an intermediate node sleeps and shuts off all links to its neighbours without prior notification, thereby forcing frequent recomputation of routing paths. Network maintenance functions such as neighbourhood discovery and time synchronization must also operate within the short time frame that the transceiver is active, thereby competing bandwidth and processing power with other data reporting functions. In the application layer, real-time data reporting functions are subject to constant and debilitating routing path breakages due to sleeping

nodes. Because of these far-reaching effects, a cross-layer perspective should be taken in devising sleep schedules such that the various inter-layer issues can be tackled collectively. Relevant design considerations regarding WSN communications will be further discussed in Section 2.3.

When computing sleep schedules, the length of each sleep or active period is determined by sensing requirements, energy consumption, and network connectivity. A very short sleep time is not practical because of two reasons. First of all, as mentioned earlier in Section 2.1, rapidly switching on-off modes of sensor nodes may actually incur more energy wastage because of the transient characteristics of hardware circuits. Secondly, executing a tightly bounded sleep schedule requires stringent timing requirements, which entails tough demands on precision clocking hardware and frequent packet exchanges for mutual synchronization. At the other end of the spectrum, the nodal lifetime can be extended considerably if the transceiver duty cycle is kept very low, but this assertion depends on the timing requirements of the sensing functions. If the sensor application involves constant environmental monitoring that generates a continuous flow of data packets, then the radio transceiver will have little idle time for sleeping. Therefore, the use of a low operational duty cycle is better suited for sensor applications where data reporting can tolerate some delay.

Since sleeping nodes create missing links in critical routing paths, another major consideration in devising sleep schedules is to ensure sensor nodes receive ample amount of sleep while maintaining sufficient network-wide connectivity at any moment. Normally, there exist multiple paths between any two sensor nodes in a multihop mesh network. Suppose each sensor node follows its own sleep schedule, then the probability that all of the paths fail due to some intermediate sleeping node is inversely proportional to the average communication duty cycle. Without knowledge of the global sleeping pattern, perhaps the only way to ensure a packet's successful delivery is to send it over multiple paths simultaneously or flood it over the entire network and hope that at least one copy will arrive at the destination. However as discussed earlier, flooding or routing via multiple paths will incur substantial transmission and network maintenance costs. As a result, the energy saved from randomly shutting down the transceiver unit could be offset by the increased overhead in flooding or multipath routing. Furthermore, the

ensuing increase in transmission latency and difficulty in implementing efficient routing makes random sleep scheduling unwelcome by data reporting and network maintenance functions.

Another way to overcome the uncertainty surrounding the validity of routing paths is to let all sensor nodes share the same sleep schedule, which implies the following design considerations. First of all, timer-driven data reporting and other network maintenance activities must be confined within specific time slots where all of the sensor nodes are active together to form connected routes, while the degree of support for event-driven and request driven data reporting depends on the length and frequency of active periods. Secondly, peer-to-peer overhearing cannot be eliminated if all of the neighbours wake up at the same time, which also entails no reduction in complexity in determining routing paths. Thirdly, some set of criteria is needed to select a sensor node to compute and revise the global sleep schedule for the entire WSN. Other than to flood the entire network during every sleep schedule update, an efficient way of exchanging sleep schedules with the rest of the sensor node population is also preferred.

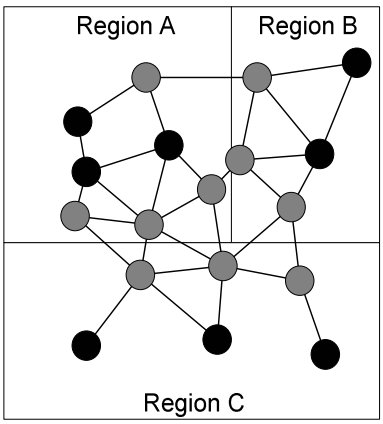

Figure 7 - Sleep regions in WSN.

In cases where the WSN application requires the sensing field to be subdivided into sleep regions that follow separate sleep schedules, sensor nodes located along the sleep region borders, coloured as grey in Figure 7, should adhere to multiple sleep schedules so that inter-regional data exchanges will not be disrupted. In this example, packets from Region A to Region B would first be cached at the corresponding border nodes during the active period of Region A and then proceed to their destination right after the active period of Region B has begun. Compared to the

use of a global sleep schedule, maintaining separate sleep schedules in different sectors obviously involves much more organizational overhead. Also, longer latency will be experienced in inter-regional packet delivery. In addition, border nodes will consume more energy than interior nodes because they are more frequently operating in active mode.

Sleep	Hibernation
• Only shuts off transceiver	• Shuts off all circuit components except for a small, low-power wakeup timer
• Can still engage in environmental sensing	• Cannot perform sensing activities
• Remain sleeping for minutes or hours	• Remain in hibernation for several weeks or months

Table 2 - Sleep and hibernation differences.

Sometimes it is adequate to turn off just the radio transceiver unit while let the rest of the circuit in active mode, while other instances require the complete shutdown of all of the sensor node components except for a small wakeup timer. These two cases are henceforth differentiated by two terms, sleep and hibernation, respectively. In the case of WSN for event detection, sensor nodes would sleep when they are not involved with packet forwarding, but they may still need to engage in active environmental sensing and data processing so that the sensor node can instantly activate the transceiver and report any detected abnormal condition. Although some energy is conserved by shutting off the radio transceiver intermittently, the sensing unit and the processing unit are still continuously drawing battery power during sleep mode. On the other hand, at times when event detection services would not be needed such as during the winter months or monsoon season, letting the sensor node enter hibernation mode by turning off all the major hardware components would save the maximum amount of energy, and Table 2 summarizes the differences of sleep and hibernation states. Because of the variations in sleep lengths in response to changing environmental conditions, the onus of determining the optimal sleep schedule should rest on some central authority where better scheduling decisions can be made with the help of global knowledge.

2.2.2 Effects on Energy Efficiency with Sleep Scheduling

Figure 8 shows an example of typical activities undertaken by the transceiver over a sleep cycle which includes transmitting data (Tx), receiving data (Rx), idle listening and sleeping. The amount of time spent in data transmission and reception is directly related to packet sizes, bit rate, protocol signalling, coding effectiveness, and general wireless channel conditions. Let the duty cycle for each sleep period, ρ, be defined as:

$$\rho = \frac{T_{active}}{T_{active} + T_{sleep}}, \tag{1}$$

where T_{active} and T_{sleep} denote the time allocated for active and sleep periods, respectively.

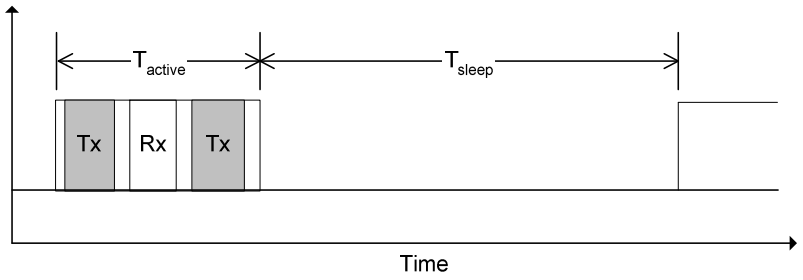

Figure 8 - Example of transceiver activity in sleep scheduling.

Summarizing the different sources of energy expenditure during a sleep cycle, the average power consumption in communications for a node (i.e., total energy consumption over a sleep cycle), P_{com}, can then be represented by:

$$P_{com} = \frac{\rho}{T_{active}}\left[\frac{D_{Tx}}{B_{Tx}}P_{Tx} + \frac{D_{Rx}}{B_{Rx}}P_{Rx} + \left(T_{active} - \frac{D_{Tx}}{B_{Tx}} - \frac{D_{Rx}}{B_{Rx}}\right)P_{com\_idle} + \frac{(1-\rho)}{\rho}T_{active}P_{com\_sleep}\right], \tag{2}$$

where D_{Tx} and D_{Rx} refer to the expected amount of bits sent and received during an active period, respectively; B_{Tx} and B_{Rx} represent the average data rates at which data is sent and received during an active period, respectively; and P_{Tx}, P_{Rx}, P_{com_idle} and P_{com_sleep} stand for the amount of

power consumed for sending data, receiving data, idle listening and sleep, respectively. Note that D_{Tx} and D_{Rx} account for both fresh and retransmitted data, as well as all necessary signalling packets and other control packets. Also, T_{active} is assumed to be set higher than the sum of the expected times needed for packet transmission and reception.

Suppose the node is supplied with an initial battery supply of $E_{battery}$, then the expected lifetime of the node (i.e., time to drain the node's battery), $\overline{T_{lifetime}}$, can be approximated by:

$$\overline{T_{lifetime}} = \frac{E_{battery}}{P_{com} + P_{proc} + P_{sense} + P_{mem}}, \tag{3}$$

where P_{proc}, P_{sense} and P_{mem} represent power consumption for the processor, sensing unit and memory, respectively. Again, it is assumed that even with sleep scheduling the transceiver will still reign as the principal energy consuming unit in each node. For typical WSN applications, it is safe to treat P_{sense} and P_{mem} as negligible values because of their brevity and infrequency in operation. On the other hand, assuming the processor follows the same sleep schedule as the transceiver and remains active throughout each active period, we have:

$$P_{proc} = \frac{\rho}{T_{active}}\left[T_{active}P_{proc\_active} + \frac{(1-\rho)}{\rho}T_{active}P_{proc\_sleep}\right], \tag{4}$$

where P_{proc_active} and P_{proc_sleep} denote power consumed by the processor during active and sleep states, respectively.

As the transceiver often is the chief power consumer among the sensor node components, it would be helpful to investigate its circuit design characteristics with respect to energy efficiency. Figure 9 shows a simple radio frequency (RF) transceiver model that is connected to the sensing and processing unit of the sensor node. Data to be transmitted will first pass through the digital-to-analog converter (DAC) and low-pass filter to prepare for up-conversion at the mixer with carrier signal generated by the frequency synthesizer. The subsequent modulated signal will be transmitted by the power amplifier (PA) over the wireless channel. On the receive path,

incoming signals will first be amplified by the low noise amplifier (LNA), and then demodulated and processed through a series of intermediate frequency (IF) and baseband filters and amplifiers (not explicitly shown in Figure 9). In the end, digital data are recovered by the analog-to-digital converter (ADC) and then forwarded to the processing unit for further decoding.

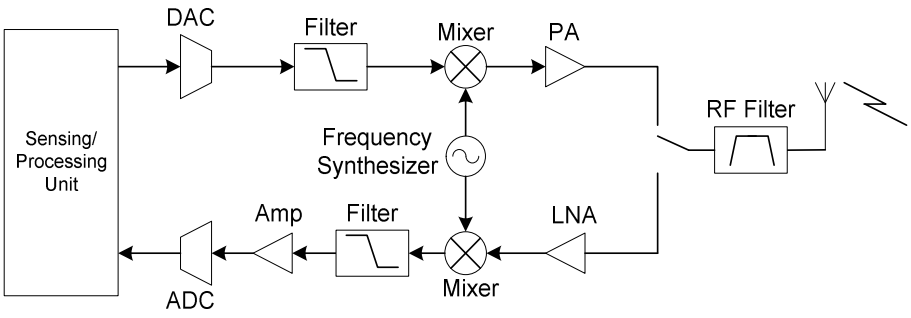

Figure 9 - A simple RF transceiver model.

The following derivations for transceiver power consumption are modeled after those given in [74] and [75], and will be used later on in Section 4.4.1 for numerical analysis. On the transmission path, the total power consumption, P_{Tx}, can be written as:

$$P_{Tx} = P_{PA} + P_E , \tag{5}$$

where P_{PA} is the amount of power consumed by the PA alone, and P_E is the amount of power collectively consumed by the other electronic components such as the mixer, frequency synthesizer, DAC, and various filters. Determining the exact values for both P_{PA} and P_E would depend on RF component design and device technology, which is beyond the scope of this research, though a simple approximation would suffice in the current work. While P_E is generally treated as a constant under various operating conditions, P_{PA} can be further broken down into the following terms:

$$P_{PA} = \frac{P_{RxSi}\left(\frac{1}{L_o}\right)\left(\frac{R_{com}}{d_o}\right)^{\alpha}}{G_{Tx}G_{Rx}\eta}, \tag{6}$$

where P_{RxSi} denotes the receiver sensitivity in Watts, L_o is the path loss attenuation at d_o metres, R_{com} refers to the distance between the transmitter and the receiver in metres, α is the path loss exponent, G_{Tx} and G_{Rx} represent transmit and receive antenna gains, respectively, and η stands for PA efficiency. In turn, the receiver sensitivity P_{RxSi} can be rewritten as:

$$P_{RxSi} = \left(\frac{S}{N}\right)_{Rx} NF_{Rx} \cdot N_o \cdot BW\,, \tag{7}$$

where $\left(\frac{S}{N}\right)_{Rx}$ is the minimum signal-to-noise ratio that provides an acceptable $\left(\frac{E_b}{N_o}\right)$ level at the receiver, NF_{Rx} is the noise factor at the receiver, N_o is the thermal noise floor in a 1 Hz bandwidth (in W/Hz or J), and BW is the channel noise bandwidth (in Hz).

Let:

$$\frac{1}{L_o} = \left(\frac{4\pi}{w}\right)^{\alpha}, \quad d_o = 0.1\,, \tag{8}$$

where w is the wavelength of the carrier frequency in metres.

Substituting Equations (7) and (8) into (6), P_{PA} becomes:

$$P_{PA} = \frac{\left(\frac{S}{N}\right)_{Rx} NF_{Rx} \cdot N_o \cdot BW \left(\frac{4\pi}{w}\right)^{\alpha} 10^{\alpha} d^{\alpha}}{G_{Tx} G_{Rx} \eta}\,. \tag{9}$$

In terms of $\left(\frac{E_b}{N_o}\right)$, P_{PA} can be represented as:

$$P_{PA} = \frac{\left(\frac{E_b}{N_o}\right) \cdot R_{Tx} \cdot NF_{Rx} \cdot N_o \cdot BW \cdot \left(\frac{4\pi}{w}\right)^{\alpha} 10^{\alpha} d^{\alpha}}{G_{Tx} G_{Rx} \eta}\,, \tag{10}$$

On the reception path, the total power consumption, P_{Rx}, depends on the power consumption of the LNA, mixer, frequency synthesizer, IF amplifiers, filters, and ADC. For the power consumption during idle listening, P_{com_idle}, it is generally assumed that this value is approximately equal to P_{Rx} as most of the transceiver components remain operational in this state. Typical power consumption during sleep mode, P_{com_sleep}, lies in the realm of μA as most of the transceiver components are switched off. Again, the exact values for P_{Rx}, P_{com_idle} and P_{com_sleep} would depend on RF component design and device technology, and estimated figures would be used in the subsequent numerical analysis. A more comprehensive study into power consumption models of the various transceiver blocks can be found in [73].

2.3 COMMUNICATIONS

Since WSNs use the wireless media for inter-node communication on any given network topology, they will encounter the same problems that are prevalent in other wireless applications. On the outset, the radio channels can deteriorate due to multipath propagation effects from dense foliage, terrain blockage, severe weather, etc., and exponential free space degradation. Besides injecting random bit errors in the data stream, these channel effects can cause any link to fluctuate between bidirectional, unidirectional or even blocked states. Various physical layer and link layer techniques such as channel coding, smart antennas, automatic repeat request (ARQ) and forward error correction (FEC) can be implemented to prevent wireless channel degradation. While the specific details of these remedies will not be elaborated further, it is assumed that the corrective methods selected for use in WSNs will not burden the sensor nodes with excessive processing load or high energy costs. The following subsections will further discuss design considerations in the upper communication layers.

2.3.1 Medium Access Control

From a network architecture point of view, since the communication range of every sensor node often does not encompass the entire sensing field in most WSN applications, inter-nodal communication will likely be relied upon multihop paths. Combining this fact with the characteristics of data transmission over the wireless medium, unique challenges will arise in the design of upper communication entities such as medium access control (MAC), network routing algorithms and transport protocols. In particular, the emphasis resides upon balancing energy

efficiency, which is the primary concern in designing WSNs, with other system attributes such as routing latency, delivery guarantees and other quality of service (QoS) requirements. Many research proposals have been published in the past few years that claim to achieve various degrees of energy efficiency, often only targeting a specific communication layer [1]. Ideally, the different communication layers should be highly modular in operation, whereby providing certain kinds of external services to upper and lower layers while hiding the implementation details. Since energy savings from one communication layer is sometimes achieved at the expense of increased energy consumption at other layers, one should deviate from traditional approach and take a macroscopic perspective across all layers when designing an energy-efficient WSN.

Before a sensor node initiates data transmission, it needs to solicit the use of a reliable and effective MAC mechanism to minimize packet collisions with other neighbours over the wireless channel. In general, the nature of MAC in WSN design is based on either contention or contentionless approach, where the former allocates collision-free channels to each sensor node while the latter lets neighbouring nodes to contend for the radio resource. For the contentionless approach, commonly available options are time division multiple access (TDMA), frequency division multiple access (FDMA), code division multiple access (CDMA), or a hybrid protocol that mixes features of all three methods. On the other hand, contention methods for WSNs include ALOHA, inhibit sense multiple access (ISMA), and carrier sense multiple access/collision avoidance (CSMA/CA).

Although contentionless approaches effectively eliminate packet collisions, it is achieved at the expense of increased complexity in hardware and protocol coordination. In any case, a central authority is often required to resolve coordination issues such as time synchronization, channel assignment and frequency of channel reuse, all of which rely on global network or adjacent neighbourhood connectivity information. Since the number of communication channels is fixed for FDMA and TDMA, flexibility and scalability in WSN management are deeply hampered. While CDMA is able to set a soft upper bound on the number of concurrent transmitters over the same radio band, some central authority is still needed to manage spreading code assignment and determine the frequency of code reuse, which is difficult given the sheer number of sensor nodes

in a WSN. In addition, while precise time synchronization is necessary for correct signal reception in TDMA and frequency hopping spread spectrum CDMA (FHSS-CDMA), it is very difficult to realize in WSNs because of the large and highly distributed nodal population, and the prevalent use of inexpensive oscillators in sensor node circuits that cause considerable clock drifts. Given that packet collisions are less likely to occur in WSN applications since sensor nodes often do not engage in frequent data transmissions, a contention-based approach should be pursued in order to avoid the many coordination problems associated with contentionless MAC protocols.

As a side note, power control problems will occur if direct sequence CDMA (DS-CDMA) methods are used in the MAC layer over a dense network topology [15]. In a regular CDMA cellular network, the power control problem was not as debilitating because the base station and mobile receivers form a star topology, which means that power control problems manifest more severely on the uplink where multiple data streams of different power levels are converged at a single base station, otherwise known as the near-far effect. On the other hand, each mobile receiver will only have to contend with weaker signal interference from neighbouring base stations, which may lead to handoff issues along cell boundaries. The common way to combat the near-far effect is to regulate the transmission power at each mobile receiver according to traffic conditions and channel fluctuations such that all reverse link signals are received at the same power level by the base station. However, such a centralized power control mechanism will encounter considerable implementation difficulty in a WSN with a random topology because of the additional overhead and complexity in exchanging power level notifications among the many neighbouring sensor nodes.

Unless the sensor node is engaged in constant environmental data gathering and packet transmission, it would normally remain in idle listening mode to monitor the radio channel for incoming packets from its neighbours for either contention or contentionless MACs. Although no data is sent or received while in idle sensing mode, energy is still being consumed by the sensor node transceiver to maintain the oscillators, amplifiers, filters and other circuit components in running state. In fact, idle listening is often the biggest energy consumer in sensor node operations [12]. In addition, the power consumed while in idle sensing mode is

comparable to that in transmit and receive modes, and previous measurements in [39] have shown that for the IEEE 802.11 wireless protocol the idle:receive:send ratio is determined to be 1:1.05:1.4. Therefore, it may be a good idea for the sensor node to shut down the transceiver and engage in sleep mode to conserve energy during light traffic periods. Note that even though the node has engaged in sleep mode, its sensing capability may remain functional if only the transceiver is shut off. However, as radio startup typically consumes a finite amount of energy, it is not wise to frequently start up and shut off the radio transceiver for the sake of energy conservation [32] [37]. On the other hand, battery lifetime can be maximized if power is consumed in spurts rather than continuous drainage, which implies that frequent sleep periods will be beneficial to extending battery operations [40]. While sleeping is an effective way to conserve energy, it increases end-to-end propagation latency and intermittently blocks communication links with neighbouring nodes. Therefore the optimal sleeping pattern can only be determined through detailed experimentation in accordance to application requirements.

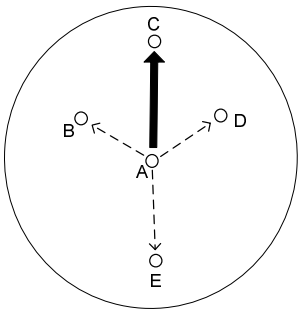

Figure 10 - Overhearing in WSN.

Another major factor that contributes to energy wastage in the MAC layer is overhearing, as illustrated in Figure 10. Suppose that Node A would like to transmit a packet to Node C, as indicated by the solid arrow. Because of the omnidirectional accessibility of the wireless medium, the radio waves will reach more than one neighbouring node (i.e., Nodes B, D and E), as shown by the dashed arrows. Even though the sender specifies the receiver address in the packet header to avoid any destination confusion, the neighbouring nodes would still inadvertently be able to capture the packet, albeit discarding it soon after reception. Unless flooding is used in packet routing, these overhearing instances simply shorten the battery life of the sensor nodes since data reception also consume a significant amount of energy as noted

earlier. For high density WSNs, the likelihood of overhearing increases as each node is surrounded by more neighbours in the vicinity. One way to minimize overhearing is to let the sensor nodes sleep right after it receives a transmission preamble that indicates it is not the intended recipient of the incoming message. Yet as mentioned above, frequent toggling of power supply modes will not necessarily achieve significant energy savings.

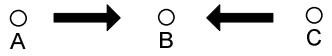

Figure 11 - Packet collision and the hidden node problem.

The third type of MAC energy wastage is called packet collisions, which are more common in wireless networks because of the existence of the hidden node problem, as illustrated in Figure 11. Here, Nodes A and C would like to send data to Node B over the open wireless medium, but they cannot detect each other's transmission activities because of the exponential loss in radio propagation. If both senders transmit their data simultaneously, then the resultant mutual signal interference at the receiver will create packet collisions. While this problem can be easily avoided with contentionless methods, the solution often brings forth cost, bandwidth utilization and scalability issues. For contention approaches, collisions can be avoided through either by an extra paging channel as in the ISMA approach, or handshaking as in the IEEE 802.11 MAC. The first solution only works if the paging channel has enough range to encompass all of the potential senders, which comes with increased cost in extra hardware and battery power, and the second option is much more adept for use in wireless networks because of its simplicity.

For example, the IEEE 802.11 MAC overcomes the hidden node problem in contention mode through the request-to-send (RTS) and clear-to-send (CTS) mechanism [7], shown in Figure 12, which works by requiring the sender to first transmit a RTS packet to the intended receiver. If the receiver is not preoccupied with another connection, it will reply by sending a CTS packet after waiting for a short time interval called Short Interframe Space (SIFS), thereby consenting incoming traffic from the sender. A duration field in the RTS and CTS packet headers specifies the time interval necessary to completely transmit the data and the subsequent acknowledgement (ACK) packet. This information is used by the neighbouring nodes to update their Net

Allocation Vector (NAV), a timer that is continuously decremented regardless of the status of the wireless channel. Neighbouring nodes would then refrain from sending their own packets until their NAV expires to reduce the probability of collisions. The receiver would reply an ACK packet to the sender upon the successful reception of the data packet to complete the transmission, and the absence of an ACK indicates the need for retransmission. The major drawback with the RTS/CTS mechanism is that since each data transmission has to be preceded by the RTS/CTS exchange, this handshaking mechanism is inefficient for sending short packets as control overhead outweighs message data. Therefore if the intended data packet is smaller than the RTS packet, then a straight data/ACK exchange would be preferable as suggested by the IEEE 802.11 specifications.

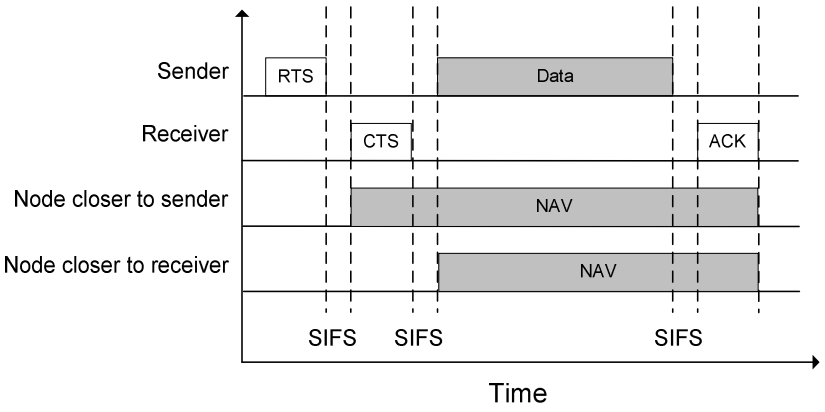

Figure 12 - RTS/CTS/ACK exchange in CSMA/CA.

The last type of energy wastage in the MAC layer is control overhead, which refers to the amount of control bits attached to each data packet's header to indicate source and destination addresses, status flags, FEC, and other control information, as well as any extra control messages in realizing each data transfer. Obviously, the number of packet header bits should be kept to a minimum without sacrificing any of the MAC layer functionalities in WSN design. For example, MAC addresses for Ethernet are represented in 48 bits and are often hard-coded to network interface equipment by manufacturers to ensure mutual addressing uniqueness within any given local area network. Although the network density of a typical WSN is usually large, the total population seldom reaches anywhere near 2^{48} or 2×10^{14} nodes. Therefore, adopting

such long MAC addressing scheme in WSN design would seem to be excessive and expensive for sensor node differentiation [41]. On the other hand, adopting the full IP addressing scheme, which uses 32 bits (IPv4) or 128 bits (IPv6) to represent a single address, is also not necessary for typical WSN applications because sensor nodes function strictly within WSN boundaries and rarely interact with external IP-based networks. To reduce the number of bits, identification markers can be assigned dynamically with the lowest possible bit length, but this requires the knowledge of the global sensor node population size beforehand.

2.3.2 Routing and Network Topology

While the MAC layer coordinates communication channel use amongst neighbouring sensor nodes that are one hop away from each other, packet delivery across the multihop WSN is dependent upon network-wide routing algorithms, in which data packets from the sender will traverse a string of intermediate sensor nodes that constitute the routing path to the data sink or other sensor nodes. Determining an optimal *a priori* routing path becomes a sizable challenge since the network topology constantly changes due to sleeping nodes, component failure and possibly sensor node mobility. As a result, maintaining an up-to-date routing table at each sensor node would require frequent control message exchanges, thereby consuming vast amounts of energy. Since there potentially exist multiple paths between a node-sink pair in a mesh network, sending packets down multiple paths (i.e., multipath routing) would increase the reliability of packet delivery against nodal failures, but only at considerable transmission and computation costs for high hop count paths [42].

One way to eliminate the need to refresh the routing table is to use network-wide flooding, which implies forwarding the data packet across the entire network nondescriptly until it arrives at the intended receiver. While flooding provides better adaptability to sensor node mobility, it achieves very poor energy efficiency because of potential inundation of the same data packets all over the network, which may result in network implosion. Therefore, classic flooding is not practical in most applications due to high communication costs except for a few scenarios such as heavy nodal mobility or control message exchanges during initial network setup stages. One way to avoid network implosion is to identify and remove the redundant packets by adding packet header fields such as incremental hop counters, sequence numbers and timestamps.

Selective flooding or gossiping is a variant of flooding that also prevents network implosion by limiting the number of outgoing neighbours in packet forwarding, but the data packet will generally take a long time to be delivered. A partial enhancement to flooding for reducing energy consumption on the downstream is via minimum cost forwarding [25], which exploits the inherent asymmetric traffic pattern in WSNs. Since sensor nodes only need to maintain end-to-end unicast routing paths with the data sink, they can forward their packets to the neighbour with the minimum cost, where the cost is computed in terms of hop count, residual nodal energy, link quality, buffer size, etc. The complexity of the cost function should be directly proportional to the network size because frequent updates of resource information would incur substantial transmission overhead for a large and dense WSN.

A priori routing algorithms constantly update the link state information regardless of data traffic, while flooding is initiated whenever data is available for transmission. On-demand routing protocols, which are originally devised for use in MANETs, lie somewhere in between these two approaches. The basic idea is to generate the routing path only when data is ready to be sent, and a couple of well-known examples are Ad Hoc On-Demand Distance Vector protocol (AODV) and Dynamic Source Routing protocol (DSR) [8]. Packet delivery is generally relied upon cached routes that were determined from previous data exchanges. Whenever a route becomes broken or stale and no backup path is available, the sender will disseminate route discovery packets across the network to search for a new path. Because of their reactive nature in routing path generation, on-demand routing protocols reduce communication and computation overhead by avoiding frequent routing table updates. At the same time, they provide robust adaptation to moderate nodal mobility while minimizing the undesired effects of flooding. Yet for WSN applications, link outages are mostly caused by sleeping nodes rather than mobility, which would lead to a very short lifetime for all of the candidate routes given a low duty cycle and no coordinated sleep scheduling. Therefore on-demand routing protocols will be forced to evoke the route discovery process practically every time data is to be sent, thereby incurring excessive transmission and processing overhead.

The routing algorithms discussed so far did not make any specific assumptions regarding the network topology. Restructuring the network hierarchy into clusters could offer better routing

performance because of its simpler networking structure [11] [21], and its effectiveness depends heavily on nodal density, transmission range, and fairness in cluster head selection. Since clustering requires high network connectivity, it is not suitable for WSN applications where the communication range of sensor nodes is very small compared to nodal density. High network connectivity is also associated with higher probability of overhearing and packet collisions if a contention-based MAC scheme is used. In addition, for a WSN that spans a large area, only the clusters that are closest to the data sink can reach it in one hop, whereas others would need to rely on inter-cluster links. Since sensor nodes within the same cluster periodically rotate to assume the cluster head role, maintaining proper inter-cluster routing path becomes an issue, and all cluster heads would need to be informed of any cluster head rotations in neighbouring clusters. Such cluster management issues will be further complicated by the need to schedule individual sleep times for sensor nodes. Issues regarding WSN clustering will be discussed further in Section 2.5.

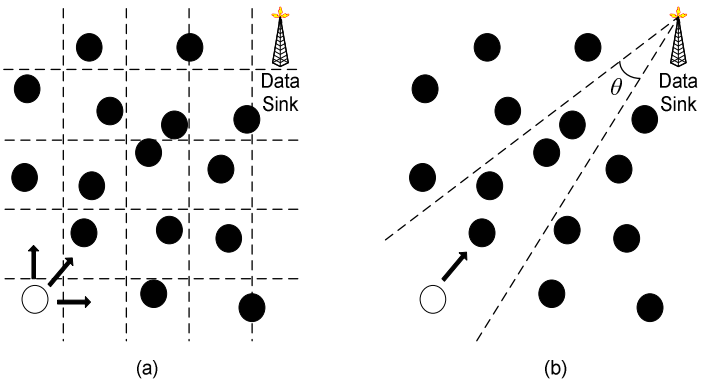

Figure 13 - Geographical routing examples.
(a) Coordinates-based. (b) Angular-based.

Since WSN applications often rely on geographical coordinates for data collection and sensor node identification, it is intuitive to use the same geographical information for routing purposes [43]. Two examples of geographical routing are shown in Figure 13, both of which assume each sensor node is aware of its geographical coordinates and that of the data sink. First, coordinates-based geographical routing demarcates the entire sensing field into a square grid, where each square may contain none, one or multiple sensor nodes. With the aid of geographical information, routing data across the grid is straightforward, and the ability to bypass natural

obstacles such as lakes, rivers or a large patch of malfunctioned sensor nodes is also greatly enhanced. On the other hand, angular-based geographical routing dictates each sensor node to forward data packets in the general direction towards the data sink that is within a certain angle, which in turn avoids the processing overhead in assigning sensor nodes to virtual grids. Still, the fundamental issue with geographical routing is that the determination of geographical coordinates is either from GPS chips or other radio ranging techniques, both of which add substantial cost, system complexity and power consumption at each sensor node.

Since in most cases a WSN is consisted of identical sensor nodes that collectively function together for data gathering applications rather than a mere aggregation of heterogeneous terminals, the emphasis is on delivering sensor data to the data sink instead of peer-to-peer inter-node communication. Based on this characteristic, chain-based routing protocols, seen in Figure 14, simplify the WSN routing structure by linking all of the sensor nodes in a single or multiple transmission chains, where data packets propagate upstream from one node onto the other until they reach the data sink [5] [9]. Similarly, control messages from the data sink are easily disseminated to all of the sensor nodes along the chain. Despite the energy savings in the reduction of routing overhead and the ease in performing in-network processing, the main challenge with the chain-based approach is to balance the transmission and processing load evenly along the chain, minimize end-to-end propagation delay, and overcome link breakages in an efficient and timely manner.

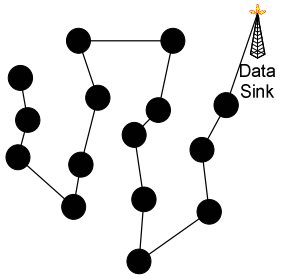

Figure 14 - Chain-based routing example.

Regardless of the amount of data traffic encountered, control messages will still be regularly exchanged across the WSN for network management tasks such as neighbourhood discovery,

routing table updates, failure detection and fault recovery. The extent of energy consumption in such control and coordination activities depends on the complexity of the network topology and the nature of the routing algorithm. Some routing approaches do require a higher vigilance towards non-responding nodes in order for the protocol to function properly, which translates into increased energy consumption and management complexity. Table 3 presents a summary of the main WSN routing approaches discussed so far and listing their advantages and disadvantages.

Routing Approach	**Advantages**	**Disadvantages**
A priori	• Mature subject in Internet research	• Poor performance in face of frequent topology changes due to sleeping nodes, mobility and node failure
Flooding	• Better adaptability to frequent topology changes • No need for routing table updates	• High energy cost • Implosion problems
On-demand	• Balances resource use and robustness against topology changes	• Routes frequently expire due to sleeping nodes
Clustering	• Straightforward routing procedure • Easy to perform in-network processing	• Inter-cluster communication issues • Only suitable for small and dense WSNs
Geographical	• Straightforward routing procedure • Better ability in bypassing topological voids	• Needs to coordinate multiple nodes within the same square area • Requires GPS chips or radio ranging techniques for location identification
Chain-based	• Eliminates link table updates • Simple routing procedure • Easy to perform in-network processing	• Increases overall end-to-end propagation latency • Weak protection against failed nodes • Needs to balance transmission and processing load along the chain

Table 3 - Summary of WSN routing approaches.

2.3.3 End-to-end transport

In the transport layer, many of the elements that are integral in regulating Internet traffic, such as flow control, congestion control, retransmission mechanism, and packet sequencing, are found to

be of little use in WSN applications. For instance, since the data packets generated by sensor nodes are most likely consisted of very short messages that do not require segmentation, the merit of having full transport layer functionality for end-to-end delivery guarantees is questionable as the chances of having out-of-order delivery are slim. Also in most WSN applications, the overall traffic profile is very simple as packets only flow from sensor nodes to the data sink and vice versa, with very few node-to-node end-to-end exchanges. Unless the sensor nodes are relaying high-volume data streams such as surveillance videos, implementing flow control and congestion control over a WSN becomes unnecessary since the overall traffic profile is known in advance and the total traffic volume is likely to be much lighter than typical Internet traffic. Moreover, packet loss occurs in the Internet when large streams of traffic converge on a network component, causing buffer overflow and packet drops. Since the traffic volume in WSNs is typically low, the probability of packet loss due to buffer overflow is virtually non-existent. Therefore, the need for end-to-end delivery guarantee mechanisms such as TCP is drastically reduced. Instead, WSNs can rely on hop-by-hop guarantees at the MAC layer to reduce the amount of upper-layer acknowledgement exchanges [27].

2.4 Sensing Coverage and Sensing Data Types

In determining the optimal monitoring coverage of a WSN, both physical distribution and communication capabilities of sensor nodes should be taken into account since they affect how events can be detected spatially and temporally. Since sensor nodes are often deployed in abundance across the sensing field, efficient coverage algorithms or heuristics should calculate the optimal network density by using the minimum number of sensor nodes, hence reducing nodal redundancy, to provide sufficient sensing coverage and communication capabilities [44]-[47]. If interested events are concentrated within a few hotspots rather than uniformly distributed across the sensing field, sensor nodes can then assume different roles where some perform actual sensing and others are designated as relays for forwarding data packets to the data sink [48]. If there exists uneven coverage across the sensing field, additional sensor nodes should be strategically placed in the monitoring service voids as identified by the coverage algorithms.

In interconnecting the sensor nodes to form a functional WSN, too much radio interference will be generated if each node's communication range is much larger than the inter-node distances. While a large communication range may help to achieve full connectivity in a densely populated WSN, each node will exert omnidirectional radio interference on its neighbours, thereby creating excessive packet collisions and peer overhearing. These interference effects can be mitigated if the communication range is artificially reduced to suit the sensory range. On the other hand, no effective WSN can be formed if the communication range is so small such that the sensor nodes cannot establish reliable links with their neighbours or even be orphaned with no reachable neighbours in the vicinity. Other considerations such as sleep scheduling, traffic volume, protocol design and transmission rate all have an impact on how prompt events can be reported to the data sink by the nodes.

2.4.1 Sensing Data Reporting Types

The nature of WSN data generation can be classified into three types: request-driven, event-driven and timer-driven. For request-driven data reporting, a request is generated by the processing centre to query specific nodes within the WSN for interested data [49]. For example, when the sensing application would like to obtain the temperature reading of a certain area, it would first send a data request to the sensor nodes covering the corresponding coordinates. Upon receiving the request packet, the sensor nodes would reply with the necessary data. Although the exchange is straightforward, the extra overhead in executing the query can be significant in terms of energy consumption. If flooding-like routing methods are used in the WSN, then a lot of redundant data traffic will be generated on both downlink and uplink during the request-driven data query exchange. Therefore, request-driven data reporting should be incurred as infrequently as possible, or at least performed according to a more regular schedule. If environmental data is to be recorded in regular intervals, then a better approach is to let the participating sensor nodes generate uplink data reports at times indicated by a timer without waiting for any query requests from the processing centre. Timer-driven data reporting achieves limited energy savings by eliminating the need for downlink request notifications, but uncoordinated uplink message forwarding could offset any gains as periodic bursts of packets are concentrated within a short time frame, thus potentially causing excessive bit collisions and

buffer overflows. Therefore further energy savings are guaranteed only if meticulous nodal coordination in message forwarding can be established.

For some WSN applications such as perimeter surveillance and event detection, the primary objective is to trigger alarms whenever interested events are detected. Similar to timer-driven cases, event-driven data reporting does not require any request packets to initiate message forwarding. Given that very few messages would be generated by a subset of the entire sensor node population, the likelihood of energy wastage as a result of collisions and other adverse traffic effects is thus minimized. However, maintaining the real-time nature of event-driven data reporting implies that all of the sensor nodes should be constantly monitoring the wireless channel for incoming packets, which entails significant energy expenditure. To summarize, Table 4 outlines the characteristics of different approaches in data reporting in WSNs. Certain WSN applications may be required to accommodate all three types, which will increase design complexity in providing adequate data reporting guarantees while minimizing energy usage.

	Request-Driven	**Event-Driven**	**Timer-Driven**
Idle Sensing Periods	Long	Long	Short
Periodicity	Aperiodic	Aperiodic	Periodic
Energy Consumption	High	High	Low

Table 4 - Properties of WSN data types.

When sensing a particular environmental phenomenon, groups of sensor nodes may report identical sets of data or values within the same numerical range. In order to reduce the amount of data reporting traffic to the data sink, it is advisable to perform some level of in-network processing first so that groups of packets with similar or even duplicate data can be aggregated into a smaller number of packets [49]. For example, suppose 100 temperature sensors record very slight differences amongst the many individual readings. Instead of sending 100 different packets to report the data, the sensor nodes can first aggregate their readings into a single packet and then transmit it to the data sink. Although data aggregation may result in a loss of data sensitivity, it is acceptable as long as the discrepancies are deemed harmless by the WSN

application. Despite the apparent energy savings, the main challenge is to develop an efficient way to collect all the packets from the sensor nodes before performing data aggregation. As described earlier in Section 2.3.2, clustering and chain-based routing are ideal candidates for in-network processing and data aggregation, though each approach brings forth additional design considerations and trade-offs.

In most WSN applications, location information is vital as they provide geographical meaning to reported environmental data. Letting every sensor node aware of its geographical coordinates and attach that to every data packet would incur high component cost and data transmission overhead. In order to reduce the amount of bits exchanged, the WSN can use the same short unique identifier across the different layers of communications and let the data sink or some central authority reference that against a list of known geographical coordinates of all the sensor nodes. This way, the sensor nodes themselves will use a shorter identifier for normal data exchanges that incurs fewer transmission overhead costs.

2.4.2 Spatial Coverage

No matter how many nodes are to remain active at any given time in a sleep schedule, a minimum level of spatial sensing coverage should be maintained for reliable sensing according to application requirements. The problem of determining the minimum number of active sensor nodes to provide a certain degree of spatial sensing coverage alludes to the classic set cover problem and its variants [47], though network connectivity should also be considered to form a minimum connected sensor cover [23]. In any case, both sensing coverage and network connectivity should jointly be considered when formulating an optimal sleep schedule that can balance all the application requirements [29] [46]. During the WSN planning stage, engineers have to contend with a wide range of design alternatives to satisfy application requirements with respect to spatial sensing coverage. Consider the following scenario: A number of identical and immobile sensor nodes are deployed over an area L x W with the data sink located at the centre of the sensing field. The method of node distribution around the data sink is either deterministic (grid or non-regular) with N nodes or Poisson with density λ. The exact manner and scale of node distribution depend on the sensing range of each node and the desire degree of overlapped

spatial sensing coverage over the sensing field, where redundant deployment of nodes mainly serves the following three purposes:

1. Provide multiple independent observations for a given event (i.e., *k*-coverage).
2. Cover for failed nodes.
3. Cover for sleeping nodes.

Intuitively, high spatial sensing coverage redundancy provides higher reliability in event detection and more flexibility in dealing with node failures. The work in [50] and [72] further proposed to rotate the sleep schedules of overlapping nodes to extend battery lifetime while maintaining sufficient spatial sensing coverage at any given time. In other words, the entire node population is divided into disjoint sets, each provide adequate coverage while adhering to different sleep schedules.

In the system model for the SS-Tree concept, the sensing field can be *k*-covered (reference), i.e., every point of interest is covered by at least k nodes, depending on application requirements [72]. However, sleep rotation of nodes with overlapping spatial sensing coverage will not be pursued. This is because the purported energy efficiency improvement comes at the expense of increased hardware and deployment costs as the degree of coverage overlapping required is at least several times than the minimum level for this sleep scheduling approach to work properly. Also, distributed computation of the optimal spatial sensing cover after node deployment would inevitably require accurate location information, which may not be available at the local level because of high costs and operational limitations with location-aware techniques such as GPS. Given that the spatial sensing cover is to be fixed over time, the main challenge therefore is to maximize energy efficiency, guarantee adequate sensing coverage, and reduce the scale of node deployment so that large-scale WSN applications can be made more economically viable.

2.4.3 Temporal Coverage

Besides spatial sensing coverage, temporal sensing coverage which is related to the timing and promptness of data reports, is equally important in sensing applications. For event-driven WSN surveillance applications, the time it takes for the data sink to be notified of an event of interest appearing somewhere in the sensing field often has to be bounded by a threshold. Aside from

satisfying application requirements, WSN designers have to take into account the various delays introduced during sensor sampling, processing and communication in determining the optimal threshold value. While delay metrics can be improved by selecting hardware components with the best timing performance, the necessity of enacting ultra-low duty cycle sleep management will inevitably drive up data communication latency significantly.

Consider the sensing field is 1-covered (i.e., every point of interest is covered by at least 1 node). Suppose the node that detected the event follows a global regular sleep schedule that only involves sleep states S_4 (i.e., every component remains active) and S_0 (i.e., every component except the clock sleeps), where data sampling and processing time, can be completed by the sensing unit within 1 active period. If the duration of the event is longer than 1 sleep cycle, then the following is the sequence of steps that may take place in event detection and notification:

1. Node wakes up.
2. Node detects event.
3. Node generates notification packet.
4. Intermediate nodes forward the notification packet.
5. Notification packet arrives at the data sink.

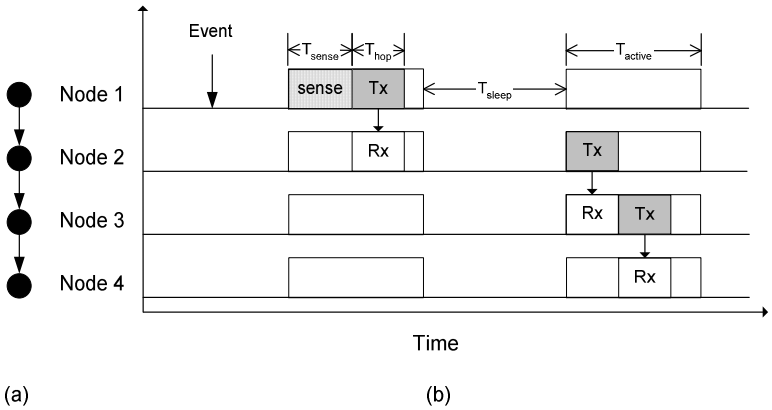

Figure 15 - Event-driven data reporting timing.
(a) Delivery route. (b) Timing diagram.

Figure 15(a) shows a linear chain of sensor nodes and Figure 15(b) illustrates the timing diagram of how an event is detected by node 1 and the corresponding notification packet is sent to node 4 through nodes 2 and 3. In the timing diagram, T_{sense} is the amount of time to perform data

sensing and processing. T_{active} and T_{sleep} denote the time allocated for active and sleep periods, respectively. It is assumed that it takes T_{hop} on average to pass the notification packet over 1 hop, whose specific value depends on packet size, processing power, transmission bandwidth, MAC signalling, modulation schemes, and general channel conditions. It is also assumed that the sensing to transmission switching times and packet processing times in intermediate nodes are negligible. Depending on the length of T_{active}, a node may be able to perform data sensing and processing, as well as forwarding the notification to the data sink over multiple hops in a single active period. For smaller T_{active} values, the end-to-end delivery process may span over several sleep cycles, thereby increasing end-to-end latency.

If the node is N_{hop} hops away from the data sink, then the expected elapsed time, $\overline{T_{event}}$, between the initial appearance of the event, assuming uniform probability of event occurrence over time, and the subsequent notification arriving at the data sink is approximately:

$$\overline{T_{event}} = \begin{cases} \dfrac{T_{active}}{2\rho} + T_{sense} + N_{hop} T_{hop} & \text{for } N_{hop} \le \alpha, \\ \dfrac{T_{active}}{2\rho} + \left\lfloor \dfrac{N_{hop} - \alpha}{\beta} \right\rfloor \dfrac{T_{active}}{\rho} - T_{sense} + \left[\left(N_{hop} - \alpha \right) \bmod \beta \right] T_{hop} & \text{otherwise,} \end{cases} \tag{11}$$

where $\alpha = \left\lfloor \dfrac{T_{active} - T_{sense}}{T_{hop}} \right\rfloor$ is the average number of hops traversable in the initial active period the event is first detected, and $\beta = \left\lfloor \dfrac{T_{active}}{T_{hop}} \right\rfloor$ is the maximum number of hops traversable in 1 active period.

For time-critical sensing applications, the various input parameters can be manipulated to minimize $\overline{T_{event}}$. However, changes in parameter value will lead to profound ramifications on other WSN design aspects such as energy efficiency and cost. For example, according to Equation (11), $\overline{T_{event}}$ can be reduced by decreasing N_{hop}, which can be done either through installing more data sinks in the sensing field to shorten mean physical distance between nodes

and their nearest data sink, or by increasing the transmission range of each node to produce denser network connectivity. The first approach would obviously push up deployment costs and network maintenance complexity, while the second approach would require higher energy consumption, potentially higher component costs, and potentially higher complexity in resolving the resultant increase in neighbourhood interference. More detailed discussions on parameter selection and trade-off will be provided later on.

Even for event-driven WSN applications, there still exists the need to provide both timer-driven and request-driven data reporting functions. Both event-driven and timer-driven types are similar as they involve data packets traveling in the upstream direction to the data sink, provided that packet acknowledgement is done hop-by-hop rather than end-to-end. However, timer-driven data reporting often needs to be coupled with data aggregation schemes for energy efficiency, so the sequence of steps that would take place in timer-driven data reporting could resemble the following list:

1. Node wakes up;
2. Internal timer triggers data reporting function;
3. Node furthest away from data sink generates initial data reporting packet from cached sensor values;
4. Intermediate nodes forward the data reporting packet, performing data aggregation along the way;
5. Data reporting packet arrives at the data sink.

If the furthest node in the data aggregation chain is N_{hop} hops away from the data sink, then the expected elapsed time, $\overline{T_{timer}}$, between the initial transmission of the data reporting packet and the aggregated data reporting packet arriving at the data sink is approximately:

$$\overline{T_{timer}} = \left\lfloor \frac{N_{hop}}{\beta} \right\rfloor \frac{T_{active}}{\rho} + \left(N_{hop} \bmod \beta\right) T_{hop} . \qquad (12)$$

Note that the processing times for data aggregation in intermediate nodes are assumed to be negligible because of the expected small data packet size and reasonably capable sensor node

microcontrollers. On the other hand, if the furthest node can only generate one timer-driven data reporting packet in each active period, then the minimum timing separation between consecutive timer-driven data reports, T_{min_timer}, is simply:

$$T_{min\_timer} = \frac{T_{active}}{\rho} . \tag{13}$$

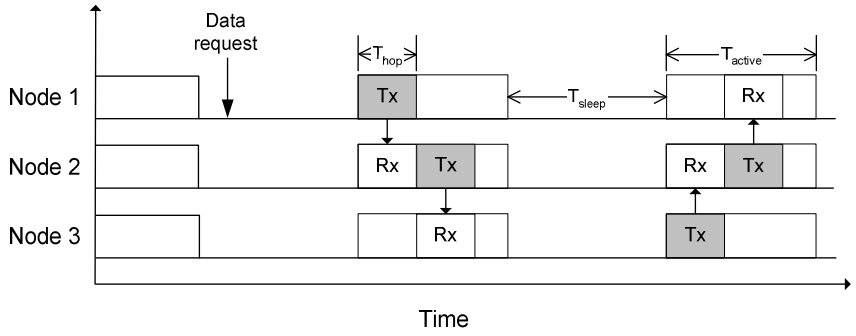

Figure 16 - Request-driven data reporting timing.

Compared with the other two data reporting types, request-driven data reporting is perhaps the most time-consuming because it usually involves packet exchanges in both downstream and upstream directions. The typical messaging sequence could be:

1. Data request arrives at data sink for querying data at a particular node;
2. Data sink forwards data request down the appropriate multihop path;
3. Targeted node receives data request;
4. Targeted node generates data reporting packet from cached sensor values;
5. Intermediate nodes forward the data reporting packet;
6. Data reporting packet arrives at the data sink.

Notice that for both timer-driven and request-driven data reporting, cached sensor values instead of instantaneous data readings are sent in order to save time. As long as the cached values are being constantly updated and the readings remain valid at least over several sleep cycles, this arrangement should be acceptable in most sensing applications. Figure 16 shows a timing example for request-driven data reporting on the linear network chain depicted in Figure 15(a).

Here, Node 1 assumes the role of the data sink and receives a data request from the WSN application that would like to query data at Node 3. In response, Node 1 sends out the data request at the next available active period towards Node 3 via Node 2. The selection of the length of T_{active} in this example allows the data request to be delivered to Node 3 within one active period. In the subsequent active period, Node 3 replies with the appropriate data report back to Node 1. Again, packet forwarding to the next hop may be relegated to the next active period if there is not enough time left in the current active period to complete one successfully. Assuming the data requests arrive randomly with uniform probability over time, then the expected elapsed time, $\overline{T_{req}}$, for the data query to be completed at the targeted node that is N_{hop} hops away from the data sink is approximately:

$$\overline{T_{req}} = \begin{cases} P_1\left[\dfrac{(1-\rho)T_{active}+\rho T_{hop}}{2\rho}+\left\lfloor\dfrac{2N_{hop}}{\beta}\right\rfloor\dfrac{T_{active}}{\rho}+(N_{hop} \bmod \beta)T_{hop}\right] & \text{for } N_{hop} \le \dfrac{\gamma}{2}, \\ \qquad +2P_2 N_{hop} T_{hop} & \\ P_1\left[\dfrac{(1-\rho)T_{active}+\rho T_{hop}}{2\rho}+\left\lfloor\dfrac{2N_{hop}}{\beta}\right\rfloor\dfrac{T_{active}}{\rho}+[(2N_{hop})\bmod\beta]T_{hop}\right] & \\ \quad +P_2\left[\left\lfloor\dfrac{2N_{hop}-\gamma}{\beta}\right\rfloor\dfrac{T_{active}}{\rho}+[(2N_{hop}-\gamma)\bmod\beta]T_{hop}\right] & \text{otherwise,} \end{cases} \quad (14)$$

where $P_1 = \dfrac{(1-\rho)T_{active}+\rho T_{hop}}{T_{active}}$ is the probability that the data request arriving at the data sink has to be processed in the next active period, $P_2 = \dfrac{\rho(T_{active}-T_{hop})}{T_{active}}$ is the probability that the data request arriving at the data sink can be processed instantaneously, and $\gamma = \left\lfloor\dfrac{T_{active}+T_{hop}}{2T_{hop}}\right\rfloor$ is the average number of hops traversable in the initial active period the data request is received.

So far, the average timing requirements for event-driven, timer-driven and request-driven data reporting types have been discussed. Since most sensing applications require all types of data reporting to be completed within a certain amount of time, the objective therefore is to minimize all of these measures through manipulating the various design parameters. Some of these parameters such as data sampling time by the sensing unit and data processing speed at the

microcontroller are hardware-specific and their selection is beyond the scope of this work. Instead, parameters related to communications and sleep scheduling such as N_{hop}, T_{hop}, T_{active} and ρ will be studied further later on in the following sections.

2.5 Topology Control and Transmission Range Adjustment

2.5.1 Types of Network Topologies

For a WSN with a random and complex interconnected topology, the individual distances between a sensor node and each of its neighbours will likely be unique. Determining the overlaying topology for routing purposes will depend on the transmission range of individual sensor nodes. If all sensor nodes are assumed to emit equivalent amounts of transmission power in all directions, then each node will require less sophisticated transceiver hardware and the resultant network topology will remain more stable. However, the WSN will have little flexibility to adapt to different nodal densities and neighbourhood failures, especially in ad hoc deployment applications. On the other hand, permitting individual sensor nodes to dynamically gauge its transmission range provides better ability for manipulating network topologies to suit various terrains and maximize transmission efficiency, though at the expense of increased cost and higher processing load [51] [52].

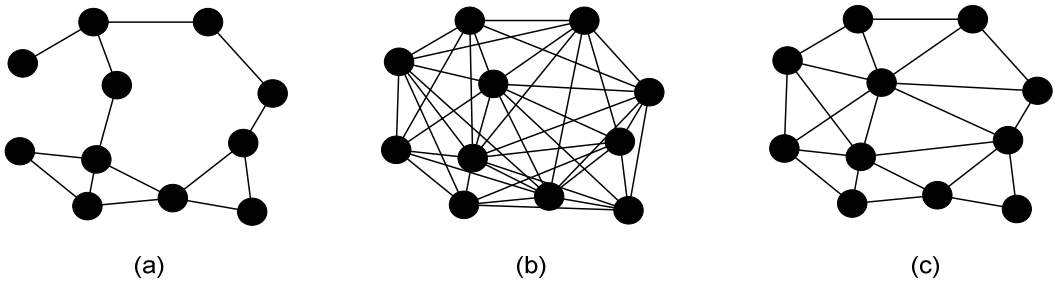

Figure 17 - Transmission range effects in WSNs.
(a) Fixed short. (b) Fixed long. (c) Dynamic.

Figure 17 illustrates the impact of different transmission ranges on a set of sensor nodes. Longer transmission distances tend to group the nodes into a fully-connected graph, thus creating more direct one-hop links that result in better connectivity [53]. Yet since the transmission power of radio waves degrade exponentially in the order of 2 to 4, precious sensor node battery power will be drained more rapidly. Also, more neighbours per sensor node imply additional complexity

and processing overhead in reducing mutual radio interference and packet delivery ambiguity. Through dynamically changing the transmission range of individual sensor nodes, the network topology can adequately balance simplicity and efficiency while maintaining adequate network connectivity [54].

If the transmission range of the data sink is extended to encompass all of the nodes within a WSN, then a unidirectional star network like the one shown in Figure 18 is formed, where packets can be delivered to their downlink destinations in just one hop. Much like ordinary cellular networks for mobile communications, star topologies adapt well to sensor node mobility since the one-hop relationship is maintained as long as the sensor node is roaming within the range of the data sink. While forming a star topology reduces hop count and processing overhead in packet forwarding, any gains in routing efficiency is offset by the high energy costs in long range wireless transmission. Also, if the sensor nodes also increase their transmission range to reciprocate the star network on the uplink, then their tiny battery power supplies can be depleted more quickly. On the other hand, many more data sinks would need to be installed across the entire sensing field in order to guarantee that all of the sensor nodes can be reached in one hop. Despite these shortcomings, organizing sensor nodes into star topologies remains the formation of choice in indoor WSN applications such as perimeter surveillance and merchandise tracking because of good adaptability to high nodal mobility and the relative ease in installing additional data sinks. However, a wide-area WSN would have no choice but to rely on hop-by-hop communication over a mesh topology, which involves much in-depth nodal coordination, because establishing and maintaining energy-self-sufficient data sinks in large numbers is a technically challenging and expensive proposition.

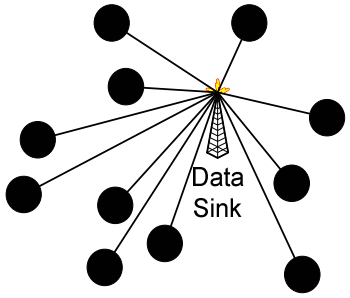

Figure 18 - Star topology.

In some application scenarios, grouping the sensor nodes in clusters offers added benefits in inter-nodal organization and packet routing over a flat networking structure [11]. The basic idea, as illustrated in Figure 19, is to divide the entire sensing field into small clusters, each revolving around a pre-determined or dynamically elected cluster head. Within the cluster, each member node is only a few hops away from the cluster head, which simplifies intra-cluster routing. On the uplink, the cluster heads can either directly reach the data sink in one hop, or via other cluster heads through inter-cluster links. The clustering hierarchical structure avoids some of the complicacies in routing packets through a multihop wireless mesh network, and cluster heads are an ideal place for performing in-network processing tasks such as data aggregation and duplicate suppression. However, since cluster heads are often equipped with the same limited power sources as other sensor nodes, they will deplete their energy supply at a much quicker pace due to their additional communication and processing burden. Therefore, each member node should shoulder an equal share of management duties by rotating the cluster head role periodically within the cluster. Also, in order for clustering to function effectively, the WSN must have high nodal density and each sensor node should be capable of long communication range to link with the data sink or neighbouring clusters. Another drawback is the increased complexity for cluster heads to simultaneously manage intra-cluster, inter-cluster, and cluster to data sink communications, as well as the periodic cluster head elections and rotations. Therefore for a sparser WSN that opts for simpler organization procedures, linking the nodes with a spanning tree-like structure would provide better routing and network management [17] [18].

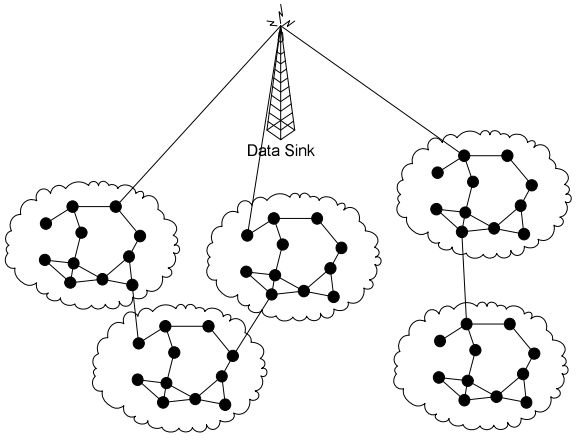

Figure 19 - Clustering in WSN.

2.5.2 Transmission Range and Hop Count

Notwithstanding the type of network connectivity model used, the measure of N_{hop} usually indicates how dense a network is, whose values are dictated by the physical location of the node in relationship to the data sink, node deployment density in the sensing field, and the level of transmission power. A dense network with low N_{hop} values can provide higher communication reliability through multipath routing over shorter hops, though it comes at the expense of increased wireless interference, route selection overhead, and power consumption during data transmission. With sleep scheduling, on the other hand, nodes on redundant routes can enter sleep states, thereby reducing idle energy usage and incidences of packet overhearing. Choosing the appropriate N_{hop} is important for balancing temporal sensing coverage requirements as well as network connectivity.

Optimizing N_{hop} after node deployment is a rather straightforward task if the nodes are enabled to dynamically adjust their transmission ranges. It would first involve finding out the overall network connectivity and power consumption profile with every node transmitting at full power, and then try to converge to the optimal N_{hop} values by selectively or collectively reducing the transmission power of each node. Despite of its simplicity, the amount of computation and messaging overhead involved can become quite considerable. Also, installing transceivers with dynamically adjustable transmission power on every node can become relatively expensive compared with the case where a simple design with a fixed transmission range is adopted. Still, because of the lack of network configuration flexibility with fixed transmission ranges, it is important to determine the optimal R_{com} value during the node pre-deployment phase such that temporal sensing coverage, network connectivity and energy efficiency can be balanced.

The following derivation is inspired by the work presented by Takagi and Kleinrock on optimal transmission ranges for packet radio terminals [55] and the geometric principles listed in [83]. Assume that for certain applications nodes are Poisson-distributed with density λ over a 2-D space of area L x W, and that each is equipped with a transceiver of fixed communication range R_{com}. Therefore, the expected number of nodes contained within a radius R is $\lambda\pi R^2$, while the probability of having i nodes distributed in an arbitrary area of size A on this sensing field is:

$$\Pr(X = i) = \frac{(\lambda A)^i}{i!} e^{-\lambda A}. \quad (15)$$

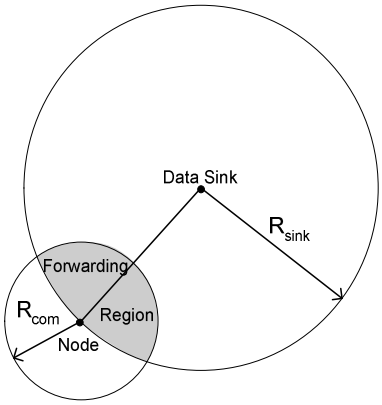

Figure 20 - Forwarding region of a node in relationship to the data sink.

If all the data packets traveling upstream from a node are always forwarded to the neighbour that is the closest to the data sink, then the forwarding region of a node is defined as the intersected area of two circles, one centred around the node with radius R_{com} and the other around the data sink with radius R_{sink}, as shown in Figure 20, where R_{sink} is defined as the distance between the node and the data sink. This greedy forwarding strategy may not always yield the best and the shortest end-to-end routing path in actual WSN implementation, but it serves as a sufficient model to analyze the relationship between R_{com} and N_{hop}, especially during the node pre-deployment phase.

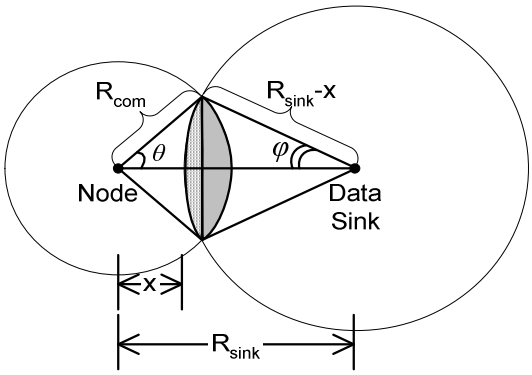

Figure 21 - Geometric relationships for calculating expected N_{hop} per node.

A neighbour located in the forwarding region of a node is said to have provided a progress of x to the node if it is $(R_{sink} - x)$ from the data sink. Then the progress of the node, z, is no greater than x if there is no node located in the lens-shaped combined striped and greyed areas in Figure 21. Note that if no neighbouring node is found in the forward direction (i.e., $x > 0$), then the least backward neighbour from the data sink will be selected. To find the size of the striped and greyed areas, we need to derive angles θ and φ from geometry principles:

$$\theta = \cos^{-1}\left[\frac{{R_{sink}}^2 + {R_{com}}^2 - (R_{sink} - x)^2}{2R_{sink}R_{com}}\right], \tag{16}$$

and

$$\varphi = \cos^{-1}\left[\frac{{R_{sink}}^2 + (R_{sink} - x)^2 - {R_{com}}^2}{2R_{sink}(R_{sink} - x)}\right]. \tag{17}$$

Then the striped and greyed areas can be respectively represented as:

$$A_{striped} = (R_{sink} - x)^2\left(\varphi - \frac{\sin 2\varphi}{2}\right), \tag{18}$$

and

$$A_{grayed} = {R_{com}}^2\left(\theta - \frac{\sin 2\theta}{2}\right). \tag{19}$$

Let:

$$A_x = A_{striped} + A_{grayed}\,. \tag{20}$$

Then the probability that the progress of a node is no greater than x becomes:

$$\Pr(z \le x) = e^{-\lambda A_x}. \tag{21}$$

It follows that the expected progress of a node, $E(z)$, is:

$$\begin{aligned} E(z) &= \int_{-R_{com}}^{R_{com}} x \Pr(x < z \le x + dx) \\ &= xe^{-\lambda A_x}\Big|_{-R_{com}}^{R_{com}} - \int_{-R_{com}}^{R_{com}} \Pr(z \le x) dx \ . \\ &= R_{com}\left(1 + e^{-\lambda \pi R_{com}^{\ 2}}\right) - \int_{-R_{com}}^{R_{com}} e^{-\lambda A_x} dx \end{aligned} \tag{22}$$

Once $E(z)$ is solved for one hop, then the expected hop count of a node R_{sink} away, $\overline{N_{hop}}$, can be computed by the algorithm EXPECTED_HOP_COUNT listed below. Note that the function *Progress*(R, R_{com}) returns the value $E(z)$ obtained by substituting the variable R in place of R_{sink} through Equations (16)-(22).

Program EXPECTED_HOP_COUNT

1 $hop\_count \leftarrow 1$

2 $R \leftarrow R_{sink}$

3 **while** ($R > R_{com}$) **do**

4 $R \leftarrow [R - Progress(R, R_{com})]$

5 $hop\_count \leftarrow hop\_count + 1$

6 **end while**

7 $\overline{N_{hop}} \leftarrow hop\_count$

From Equations (11), (12) and (14), the outcome of the hop count calculation is inherently tied to the timing performance of the three data reporting types, where a lower $\overline{N_{hop}}$ value translates into better data reporting times. Given R_{sink} and λ are fixed, such timing improvement can only be achieved by increasing R_{com}, which would lead to a list of problems associated with denser network connectivity as mentioned earlier. As a side note, throughout the above calculations it is

assumed that the nodes are Poisson-distributed with density λ. In practical implementation, the actual node distribution on the sensing field may be highly variable, and the direct forwarding path may be affected the presence of physical obstacles such as mountains and lakes. Therefore, the derivations can be modified to take into account such effects, which will be relegated as part of our future work.

2.6 Deployment Strategies, Topology Maintenance and Failure Recovery

Many of the WSN applications envision sensor nodes to be deployed in an ad hoc manner, where the sensor nodes will be dropped off aerially and manage to be uniformly distributed over the sensing field. If the sensor nodes are to be delivered by airplanes, then the cost of deployment would be proportional to the distance travelled in covering all of the target terrain. Protective measures are to be in place to ensure a safe landing for the sensor nodes and to stabilize their location foothold. Since mass deployment of sensor nodes via blind aerial means often results in uneven local nodal densities, the entire sensing field would need to be saturated with sensor nodes with longer communication range in order to guarantee the nodal density over any sector exceeds some minimum sensing coverage threshold. However, this nodal redundancy may lead to excessive overhearing in highly concentrated areas, thereby lowering energy reserves at a much quicker pace. Also, blanketing the sensing field with tiny non-biodegradable and potentially toxic sensor nodes would not be conducive to protecting the environment.

In many of the ad hoc situations where WSN applications were to be deployed in place like battlefields or disaster zones, the resultant network topology is highly variable due to the rapid placement of the sensor nodes over the entire sensing field. For infrastructural WSNs where the system lifetime is expected to last for a longer period, more stationary deployment means such as land crews or helicopters can be adopted so that a more favourable network configuration can be arranged for maximum energy conservation. Of course, such meticulous deployment approaches will increase manpower costs, but a carefully planned and laid out WSN topology will generally reduce network management overhead, provide better coverage to hotspots within the sensing field, and facilitate the inclusion of energy and cost saving design features.

Ideally, network-wide connectivity should be determined shortly after all of the sensor nodes have been deployed. Since it takes time to deploy all of the sensor nodes over a large expanse in for wide-area WSNs, the link state can either be computed dynamically as nodes are successively introduced to the network, or let every sensor node to keep an initial sleep timer that allows all of the sensor nodes to wake up at the same time some time after deployment. The first approach essentially let every node to engage in idle listening until the complete link state information has been determined, which implies a slow link state converging time for a large WSN as newer nodes are slowly appended to the core network and an unfair share of energy expenditure to those sensor nodes deployed early on. The second approach expedites the neighbourhood discovery process, assuming that all of the sensor nodes have been successfully deployed at wakeup time, but the challenge lies in deciding the optimal starting sleep time and initiating the sensor nodes with this information.

During steady state, some sensor nodes fail from component failures or depleted battery while new replacement sensors are introduced to the WSN. Network maintenance functions have to continuously monitor the network connectivity status and detect nodal failures and other abnormalities. Besides running self-checkups on hardware circuits, the sensor nodes should also report energy level information to the network control authority so that pre-emptive measures can be devised to bypass sensor nodes or regions with low energy reserves. Whenever a critical loss in sensing field coverage caused by nodal failures is predicted or detected, there should exist an efficient way to replace those malfunctioned components. However, since the deployment cost for individual sensor node replacement is likely to be expensive (e.g. helicopter ride over hundreds of kilometres), such replacement activities should be conducted only when a substantial number of failures have occurred. This implies that the WSN should be able to function effectively and provide adequate sensing capabilities in face of a considerable number of failed sensor nodes.

Figure 22 illustrates two common strategies in bypassing a failed sensor node, coloured as white, on the routing path that is drawn using solid lines. The first approach, shown as the dashed line, simply increases transmission power to skip over the failed node and re-establish the broken link. While this approach allows the route repair to be performed locally, the subsequent extension of

transmission range accelerates the energy depletion of the participating nodes as well as increases the incidences of overhearing at neighbouring nodes. The second recovery approach computes a new route around the failed node without increasing transmission power, though the availability of global link state information is required at each node for the route update, which is difficult to implement given the transmission overhead in providing periodic link state updates around the WSN.

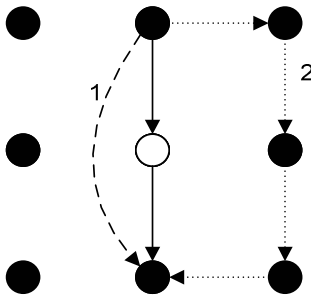

Figure 22 - Bypassing a failed sensor node.

2.7 TIME SYNCHRONIZATION

Macroscopically, the primary role of time synchronization protocols is to allow the sensor nodes to determine the correct sequence of events, where obtaining precise timing information is not important. More advanced sensor applications require participating nodes to maintain relative clocks with each other by periodically exchanging beacons messages containing local timestamps. Yet the most complex time synchronization model requires every node to reference a central clock for the absolute global timescale, which would consume more energy if very accurate timing is to be achieved. Even with the use of coarse time synchronization, potential clock drifts in the range of several milliseconds to several minutes per day may still appear. Therefore, ideal WSN applications should exercise more tolerance in their design towards such timing distortions.

Since obtaining absolute timing is important for WSN applications that deal with time-sensitive data, some central timekeeping authority should provide a benchmark global time to the entire sensor node population. Data sinks could fill this role by obtaining the precise time reading from the processing centre and then periodically disseminates that information across the WSN.

However, transmission latencies in multihop communication and clocking imperfections in inexpensive hardware oscillators make obtaining precise timing information for each sensor node a formidable challenge [10]. Many of the modulation and channel access methods in wireless communications require time synchronization precision in the microsecond range, which is difficult to achieve without incurring significant transmission overhead from frequent timing information updates. In addition, not only do more precise clocking parts cost more in the marketplace, but they also consume more energy since they typically run at a much higher frequency. Environmental factors such as temperature, vibrations and humidity further affect the accuracy of clocking functions in sensor nodes. Together, all of these adverse factors make maintaining time synchronization for WSNs no trivial task.

Issues concerning time synchronization have been intensively studied by the research community, though mostly in the realm of Internet and cellular networks. Well-developed protocols such as NTP [28] have kept the Internet reasonably in sync for years, but they are far too complex and communication-intensive to be applied to energy-constrained WSNs. On the other hand, accurate time synchronization can be easily accomplished if location awareness tools such as GPS and radio triangulation are available since these methods use time differences from several reference points to deduce location information. However because of the cost and energy concerns mentioned earlier, such methods are prohibitive for low-cost low-energy WSN applications to provide time synchronization. A number of other proposals claim to provide time synchronization services to WSNs with accuracy in the realm of microseconds [14] [22], but only for small network size with low hop counts. Because of the high transmission latency and relentless clock drifts of sensor nodes, time synchronization for larger WSNs remains the prime engineering concern. A more detailed discussion on the various aspects of time synchronization in WSN applications can be found in [81].

3 PROPOSED APPROACH AND THE SENSE-SLEEP CONCEPT

In light of the many design considerations and suggested approaches outlined in the previous chapter, the following list highlights the key concepts embedded in the current work.

- *Application-Specific* - Given the application-specific nature of WSNs, it is rather not practical to seek a one-size-fits-all solution in WSN design. While some WSN applications require *continuous monitoring* capabilities that generate a constant stream of delay-sensitive data, this thesis work focuses on *event-driven* WSNs for wide-area surveillance, which monitors infrequent but important events such as fire, intrusion, and other sudden environment abnormalities. The sensing field is assumed to be laden with a large number of identical and immobile nodes, collectively providing adequate sensing coverage with low degree of coverage redundancy. Nodes within a particular area report to a single data sink, which is assumed to possess a more stable energy supply and extra computing power. Event occurring frequency varies with time and changing environment conditions, and there potentially exist long periods of sensing inactivity and even the need for extended node hibernation. A mixture of heterogeneous control and data traffic would contend for limited bandwidth provided by the active nodes. While data reporting format is mostly event-driven, request- and timer-driven types should also be accommodated.

- *Coordinated Sleep Scheduling* - On average, nodes need to operate at an ultra-low communication duty cycle (<1%) for extending nodal lifetime to multiple years. Without sleep scheduling coordination, sensing and communication reliability cannot be guaranteed under an ultra-low duty cycle without depending on stochastic approaches that compensate the sleep randomness with the deployment of a large number of redundant nodes, which could increase hardware cost and management complexity tremendously. The length of each active period should minimize end-to-end packet propagation delay with reference to application requirements.

- *Near Connected Domatic Partition* - If a node can just shut off its power-hungry transceiver during a sleep period, then it can still monitor events through its alert sensing

unit without wasting energy via idle listening. Therefore the active virtual backbone, along with a subset of inactive nodes connected to the virtual backbone, can provide the necessary sensing coverage during a particular active period. The notion of connected domatic partition presented in Section 1.1 can be applied in this case to find the optimal number of disjoint or near-disjoint connected dominating sets such that the energy load can be spread out across the WSN while sufficient network connectivity and sensing coverage can still be maintained.

- *Spanning Tree Structure* - Since packets are assumed to flow either from sensor nodes to the data sink or vice versa with very few direct end-to-end inter-node exchanges, the virtual backbone can take the form of a spanning tree to connect all the active nodes. The main advantage of using a spanning tree is that each node does not need to maintain full link state information for the entire WSN in order to forward the packets correctly. Instead of relying on exquisite routing protocols that require frequent routing table updates, simple flooding procedures that allow the packets to flow either upstream to or downstream from the data sink can be used on the spanning tree without incurring substantial messaging costs. While a tree-like structure is susceptible to breakage caused by failed upstream nodes, communication reliability can still be maintained via other CDSs within the domatic partition provided that each node is aware of the CDS assignment of its 1-hop neighbours.

- *Centralized Approach* - Many prior works advocate the use of distributed and localized approaches to determine everything including sleep schedules, topology management, routing setup, and sensing coverage in WSN design. This way of thinking has its roots in the development of large-scale heterogeneous networks such as MANETs and the Internet, which regard each node to be unique and autonomous. For WSNs, however, all nodes need to cooperate together to perform joint monitoring tasks, and the presence of a central authority (e.g. the data sink) is vital to provide critical information such as application requirements and accurate global time. Therefore it is logical to let the data sink be actively involved in WSN management for better communication and sensing reliability. Issues of scalability and robustness, often brought up in centralized approaches, will be discussed later on.

- *Cross-Layer Design* - Finally, to overcome the main challenges in WSN implementation with respect to the design aspects of hardware, communication and processing, the general consensus is that a cross-layer optimization approach in resolving the various issues in WSN design should be advocated [13] [20] [69]. Because of the use of ultra-low duty cycle sleep scheduling, all the necessarily control and data packet exchanges, no matter how infrequent they may be, are to be conducted within the tiny fraction of the time allotted as active period. Effective management of such packet exchanges for maximum energy efficiency require cross-layer collaboration among application requirements, routing procedures and MAC design.

In addition to the key concepts listed above, the current work also makes some additional assumptions regarding hardware selection. First of all, each node is equipped with very simple transceivers that only permit a fixed transmission range. Secondly, a single-channel CSMA-based MAC is used for wireless channel access. Thirdly, no geographical location information through GPS or radio ranging techniques is readily available at sensor nodes. All three assumptions are made to reduce overall hardware costs, though the proposed scheme would still work well, maybe even better, under more favorable operating conditions with variable range transceivers, multi-channel MAC and location identification chipsets available.

3.1 BASIC CONCEPT

Figure 23(a) shows a simple WSN with a data sink and 9 nodes arranged in a square grid pattern. Suppose that a spanning tree with 3 branches is logically overlaid on top of the original topology, and all 9 nodes follow the same global sleep schedule, as shown in Figure 23(b) and (c), respectively. Then during the active period, considerable amounts of overhearing and packet collisions, represented as the dashed lines in Figure 23(b), would occur amongst neighbouring nodes. In contrast, none of the nodes will be capable of communicating during the sleep period, which renders the WSN useless if it were to detect signs of abnormality occurring during that time span. Only until the next active period appears can the node pass on any urgent notification to the data sink.

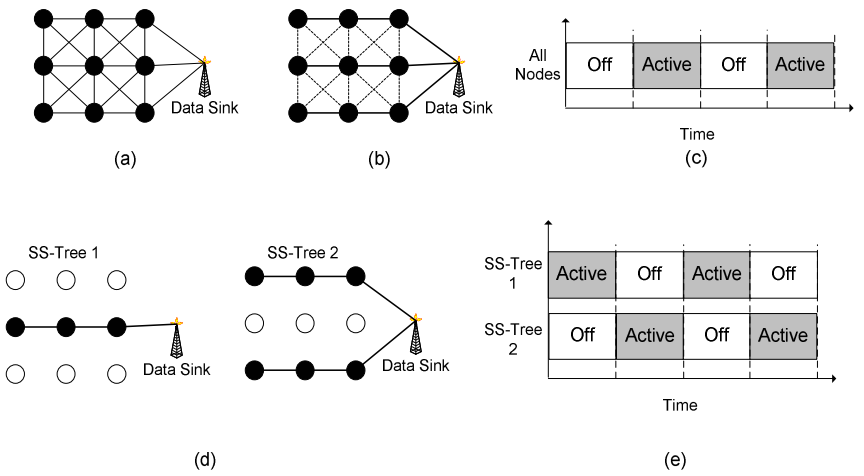

Figure 23 - SS-Tree concept for WSN topology simplification.
(a) Original WSN topology. (b) Logical spanning tree overlay. (c) Global sleep schedule.
(d) SS-Tree configuration. (e) Interleaved sleep schedules.

Suppose that the 9 nodes are divided 2 groups of 3 and 6, respectively, in the manner shown in Figure 23(d), where each group follows its own sleep schedule such that the active periods of each group alternate, as illustrated in Figure 23(e). The nodes of each group are arranged to form a tree rooted at the data sink with much sparser branches such that nodes on separate branches cannot communicate with one another. With fewer neighbours per active node, incidences of overhearing and packet collisions would be drastically reduced even with the use of only a single wireless channel, which translates into energy savings. Since the nodes on each tree share the same sensing and sleeping cycle, the tree itself is named as *Sense-Sleep Tree*, or *SS-Tree* for short.

Besides achieving energy savings from simplifying the WSN topology, notice that in Figure 23(d) the sleeping nodes, coloured as white, are strategically located beside at least 1 branch of the other SS-Tree that is effectively a connected dominating set (CDS). If each sleeping node in sleep state S_0 wakes up and enters sleep state S_1 whenever a neighbouring SS-Tree becomes active, then whenever signs of abnormality emerge, the node can instantly switch on their transceivers and forward the emergency notification to the data sink via the active SS-Tree. As different SS-Trees rotate in time to act as the virtual backbone, they avoid overburdening any set of nodes from being the sole virtual backbone. In Figure 23(e), the SS-Tree formation allows the nodes to remain on alert and capable of event reporting 100% of the time even though the

transceiver is functioning at a 50% duty cycle, thereby providing twice the level of temporal sensing coverage using about the same amount of energy without making any substantial changes to the WSN. Based on Equation (11) in Section 2.4.3, the improvement attained in $\overline{T_{event}}$ if each node is adjacent to (N_{sst}-1) SS-Trees with evenly interleaved sleep schedules is:

$$\overline{T_{event}} = \begin{cases} \dfrac{T_{active}}{2\rho N_{sst}} + T_{sense} + N_{hop} T_{hop} & \text{for } N_{hop} \le \alpha, \\ \dfrac{T_{active}}{2\rho N_{sst}} + \left\lfloor \dfrac{N_{hop} - \alpha}{\beta} \right\rfloor \dfrac{T_{active}}{\rho} - T_{sense} + \left[\left(N_{hop} - \alpha \right) \bmod \beta \right] T_{hop} & \text{otherwise.} \end{cases} \quad (23)$$

Despite this advantage in increasing sensing reliability for SS-Trees, one minor drawback is that both timer-driven and request-driven data cannot be simultaneously gathered from all SS-Trees since each follows its own sleep schedule and only one SS-Tree may be active at any given time. For event-driven surveillance applications though, the impact of having unsynchronized periodic or request-driven reports of ambient conditions and operational status of sensor nodes is far less significant than experiencing any delays in alerting the data sink of signs of abnormality and other emergency events. Therefore besides requiring the WSN application to tolerate a higher delay in timer-driven and request-driven data reporting, event-driven data is to be given a higher priority in packet delivery that allows it to be expedited to the data sink on the active SS-Tree when both types of data coincide.

Another issue with SS-Trees is their potential inability to provide full spatial and temporal coverage at any given time, due to uneven network connectivity across the WSN such that at least one SS-Tree cannot form a true CDS. Figure 24(a) shows the same basic WSN topology as that used in Figure 23, but the 9 nodes are now arranged into 3 separate SS-Trees operating under the same low communication duty cycle with the sleep schedules arranged in a staggered manner, as shown in Figure 24(b). Because of sparse network connectivity, only SS-Tree 2 can form a CDS while the other two can be seen as partial CDSs at best. Therefore there exists an uneven distributing of event reporting windows as illustrated in Figure 24(c). Allowing dynamic adjustment of transmission ranges would certainly be helpful in reducing the number of partial

CDS, though it is more important to determine how many SS-Trees can be formed for a given R_{com} in the first place.

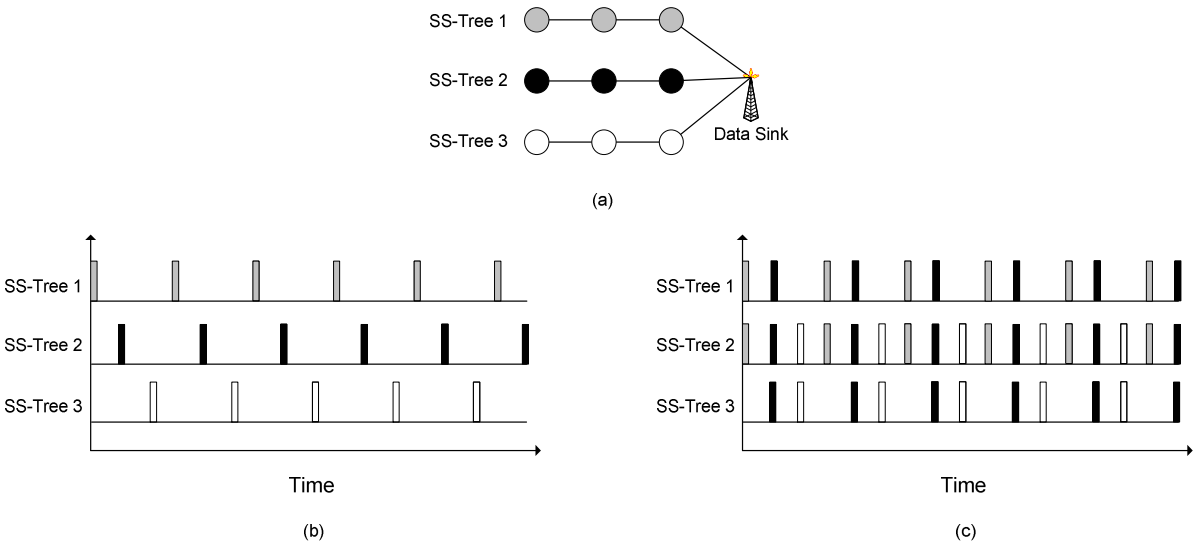

Figure 24 - Impact of SS-Trees on spatial and temporal coverage.
(a) SS-Tree assignment. (b) Staggered sleep schedules. (c) Event reporting windows.

The key issue for realizing the SS-Tree concept is the determination of how the sensor nodes can be assigned to a certain number of SS-Trees on a given WSN topology. If transceivers with fixed transmission ranges are used, then the number of SS-Trees computable, N_{sst}, has to be inferred from the parameters R_{com} and λ since the full network topology is not known during pre-deployment phase. The fact that a node is adjacent to (N_{sst}-1) SS-Trees is equivalent to saying this node has N_{sst} different paths to the data sink. Assuming all such paths are disjoint, then N_{sst} for a single node can be approximated by the number of neighbours residing in the forwarding region as illustrated in Figure 20. Nevertheless, with multiple SS-Trees coexisting in a WSN comes the possibility of tree overlapping, where a selected number of sensor nodes may have to belong to multiple SS-Trees to maintain tree connectivity. Such nodes, called *shared nodes*, need to follow multiple sleep schedules since each SS-Tree maintains its own sleep schedule. Therefore, the shared nodes constitute the weak points of the network where they would deplete their batteries much sooner than the rest of the WSN population. To minimize the likelihood of producing shared nodes, the estimate of N_{sst} for the entire WSN should be made as conservative

as possible. Therefore by setting $x = 0$ in Equations (16) to (22) and taking the minimum overall result, a conservative estimate of N_{sst} is given by:

$$N_{sst} = \min_{R_{com} < R \le \frac{\sqrt{L^2+W^2}}{2}} \left\lfloor \lambda \left[R^2 \left(\varphi' - \frac{\sin 2\varphi'}{2} \right) + {R_{com}}^2 \left(\theta' - \frac{\sin 2\theta'}{2} \right) \right] \right\rfloor, \tag{24}$$

where $\theta' = \cos^{-1}\left(\frac{R_{com}}{2R} \right)$ and $\varphi' = \cos^{-1}\left[1 - \frac{{R_{com}}^2}{2R^2} \right]$. Note that any node with $R \le R_{com}$ is within reach of the data sink in one hop, so they are excluded in the N_{sst} calculations.

3.2 SS-TREE OPERATIONAL STAGES

Figure 25 shows the complete operational stages throughout the WSN's life cycle using SS-Trees. Soon after initial nodal deployment, the WSN will enter the *Network Initialization* stage, which allows the data sink to gather network connectivity information from individual sensor nodes, compute the SS-Trees, and disseminate the sleep schedules to every sensor node. The sensor nodes will then alternate between *Active* and *Sleep* stages for the majority of their lifetime in providing constant physical monitoring and performing the necessary data reporting tasks. During prolonged periods when sensing services are not needed, the entire WSN would enter *Hibernation* mode to conserve the maximum amount of battery power. To preserve network integrity, sensor nodes need to undergo the *Neighbourhood Update* process periodically for keeping informed of any status changes of adjacent nodes in sleep schedules or hardware failure. Finally, sensor nodes and the data sink will enact the *Failure Recovery* procedures in case node failures are detected. The following paragraphs will further explain the operational dynamics in each of the stages.

To realize the benefits of SS-Trees, it is important to devise an efficient method for determining and disseminating the sleep schedule to all of the nodes during the *Network Initialization* stage. Distributed approaches for sleep schedule computation offer better scalability and robustness against single point of failure [2] [16] [71]. However because of the need to adapt to different monitoring sensitivity requirements in response to varying environmental conditions, the optimal

sleep schedules should be prepared by the data sink or the more powerful processing centre since they are most sophisticated to handle scheduling decisions in a global manner. The following list of steps concisely describes the process in determining the initial sleep schedules at network initialization:

1. Each node learns of its 1-hop neighbours
2. Each node forwards local link state information to data sink
3. Data sink computes optimal SS-Tree structures and sleep schedules with respect to the global connectivity map and application requirements
4. Data sink disseminates computed sleep schedules to every node through source routing
5. Each node exchanges sleep schedules with all 1-hop neighbours
6. Each node follows its received sleep schedule to rotate between active and sleep states

Since the data sink obtains global knowledge of network connectivity and link costs after Step 3, any rescheduling commands issued by the data sink from then on can be delivered to the nodes swiftly with direct source routing or relying on SS-Trees as efficient broadcast trees. Also, the fact that each node is made aware of its neighbours' sleep schedules after Step 5 ensures robustness against future SS-Tree breakages from upstream nodes. To ensure network integrity, the data sink periodically broadcasts a probing message down each SS-Tree to confirm the well-being of individual nodes. A missed periodic broadcast indicates a high probability of an upstream breakage on a particular SS-Tree branch, and the corresponding nodes can refer to the stored neighbourhood sleep schedules and reconnect to the data sink via the next neighbouring node that is scheduled to become active. More importantly, the data sink would assume a central role in permanently repairing SS-Trees with the help of the global connectivity and sleep scheduling information it possesses.

During the *Sleep* state, the sensor node shuts down the radio transceiver to conserve power, thereby excluding it from intra-WSN communication. However, other hardware components such as the processor and the sensing unit can remain active to allow the sensor node to monitor the surrounding area for signs of abnormalities. In case an emergency situation arises, the sensor node which sensed the abnormality will search its neighbourhood sleep scheduling list and find out which nodes are scheduled to be active next. When that active period commences, the sensor

node will turn on its transceiver and sends a notification message to the active neighbour. That neighbour will in turn forward the urgent message to the data sink through the active SS-Tree. When the WSN is expected to undergo an extended period of inactivity, the entire sensor node population should enter *Hibernation* state by shutting off all hardware components except for a tiny low-power wakeup timer. While the nodes will gain a few months of rest during hibernation, they should periodically wake up to participate in global synchronization sessions to minimize clock drift. These sessions are also ideal for notifying the nodes of any changes to the hibernation schedule, as well as to detect any changes to the network topology from nodal failure or newly added nodes.

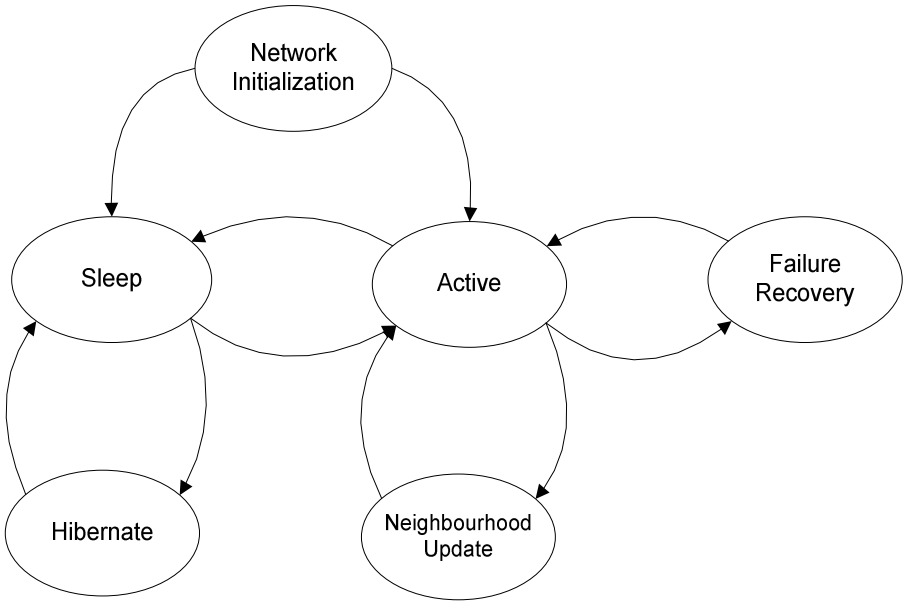

Figure 25 - WSN operational stages with SS-Trees.

Given the ultra-low communication cycle, sensor nodes will spend only a tiny fraction of their lifetime in the *Active* state. Nonetheless, it is the most important state with respect to the overall operation of the WSN where all the necessary data reporting and network maintenance tasks are performed within this short time span. In order to complete all packet transmission and forwarding activities within a single short active period, the data reporting process should also be carefully managed instead of letting the nodes transmit packets at will to further reduce management overhead. In addition, data aggregation and duplicate suppression will be enforced

so that the processing load will be distributed as evenly as possible along routing paths and across the sensing field to prevent premature battery depletion. Such issues related to the sensing application requirements will be discussed later in Section 3.4.

A potential issue with SS-Trees is their vulnerability towards nodal failures since any failed intermediate node will instantly sever the end-to-end path on the spanning tree. Since each sensor node does not keep the full network connectivity information, it is difficult to route around the failed nodes via some distributed algorithm alone. Also, the *Failure Recovery* process is further complicated by the fact that neighbouring nodes often cannot reach each other during steady state operations as they belong to separate SS-Trees with different sleep schedules. Therefore in order to recover from nodal failures without merely resorting to classic flooding, it is important to let neighbouring nodes be aware of each other's sleep schedule through exchanging local information during *Neighbourhood Update* sessions from time to time. Whenever a node senses an upstream breakage on its SS-Tree branch, it can refer to the stored neighbourhood sleep schedules and reconnect to the data sink via the next neighbouring node that is scheduled to become active. More importantly, the data sink would assume a central role in permanently repairing SS-Trees with the help of the global connectivity and sleep scheduling information it possesses. Failure recovery issues will be briefly discussed later in Section 3.4.4, though its precise recovery mechanism and neighbourhood update procedures are beyond the scope of this thesis, and they will be instead delegated as part of future research.

3.3 SS-TREE COMPUTATION METHODS

Since SS-Tree is a novel concept in WSN organization and management, issues such as sleep schedule determination, data dissemination dynamics, neighbourhood discovery process, and failure recovery procedures remain to be explored. However, the core problem for realizing the SS-Tree concept is the actual determination of how the sensor nodes can be assigned to a fixed number of SS-Trees on a given WSN topology. With multiple SS-Trees coexisting in a WSN comes the possibility of tree overlapping, where a selected number of sensor nodes may have to belong to multiple SS-Trees to maintain tree connectivity. Such nodes, called *shared nodes*, need to follow multiple sleep schedules since each SS-Tree maintains its own sleep schedule. Therefore the shared nodes constitute the weak points of the network where they will deplete

their batteries sooner than the rest of the WSN population. Since no existing tree computation algorithm or general approach has been suggested regarding SS-Trees, the following formal definition of the SS-Tree problem will address the various objectives described thus far.

Symbols - Let:

- V be the set of all sensor nodes plus the data sink
- E be the set of all bidirectional links between nodes in V
- K be the set of all SS-Trees
- s be the symbol representing the data sink in V

Problem Definition: Given an undirected connected graph $G = (V, E)$ with node s denoted as the data sink, form $|K|$ connected SS-Trees, all rooted at node s, with the following main objectives:

1. Minimize the number of shared nodes (i.e., nodes belonging to multiple SS-Trees)
2. Minimize the number of co-SS-Tree neighbours of each node
3. Minimize the cost of forwarding messages between the data sink and each node

Since the presence of shared nodes on SS-Trees has the most adverse impact on the expected lifetime of a given WSN, it becomes the top priority in the proposed computation approaches. For the second objective, each node should preferably be adjacent to the maximum number of neighbours that reside on other SS-Trees in order to take advantage of a larger number of available event reporting windows. Also, specifying a smaller number of co-SS-Tree neighbours per node would decrease the amount of overhearing interference produced. The third objective requires that all nodes on each SS-Tree should reside on the minimal cost path to the data sink, where the cost can be interpreted in terms of energy usage, transmission distance or hop count.

One approach to search for an optimal SS-Tree solution is via an integer linear programming (ILP)-based model, which formulates the SS-Tree computation criteria into a set of constraint equations with a minimization goal. However, since the more general problem of finding a maximum domatic partition is known to be NP-Complete [80], a less complex iterative algorithmic approach for searching feasible SS-Tree assignments should also be considered. The

following sections will present three different approaches in solving the SS-Tree computation problem, two using ILP techniques and one through an iterative searching algorithm.

3.3.1 Iterative Algorithmic Approach

The objective of the proposed SS-Tree computation algorithm is to offer a fast approach to compute SS-Trees while balancing the three objectives outlined in the problem definition. The algorithm follows a greedy depth-first approach that constructs SS-Trees from the bottom-up on a branch-by-branch basis. The general idea is to construct the SS-Trees based on the underlying shortest path tree rooted at the data sink as determined by Dijkstra's algorithm [30]. The SS-Tree computation algorithm proceeds in a number of *iterations*, where in each iteration an end-to-end minimum cost path is appended to one of the SS-Trees. At the start of the algorithm, all the nodes in the WSN topology are deemed *unselected*, and each path is built starting from the node with the highest path cost and uses as many unselected nodes as possible along the way. Each iteration is then divided into a number of *steps*, where in each step the path grows by one hop by adding a single unselected upstream node belonging to the next lowest hop level to the currently selected set of nodes. For picking the ideal unselected upstream node among multiple candidates, the selection criteria favour those with the fewest number of neighbours in an effort to reduce the number of shared nodes in future iterations. If no unselected upstream node is available for selection, then a node is picked from those that were already selected, which carries the risk of creating a shared node if the selected node belongs to a different SS-Tree. Path construction for a given SS-Tree in the current iteration stops when either the data sink or a selected upstream node belonging to the same SS-Tree is reached.

Across the WSN topology, two constructed paths are said to be *adjacent* if all of the vertices on one path are adjacent to at least 1 vertex on the other path, and vice versa. If each path is assigned to a different SS-Tree, then the nodes on both paths will enjoy the advantages of increased sensing duty cycle and added protection from nodal failures. However, suppose in every iteration the algorithm constructs the paths by selecting candidates from the same pool of unselected nodes, then it would be very difficult to maintain path adjacency among different SS-Trees because the constructed paths can crisscross each other in an unordered fashion. An

intuitive approach to maximize path adjacency is to construct a path from the set of nodes that are neighbours to the nodes belonging to a different SS-Tree.

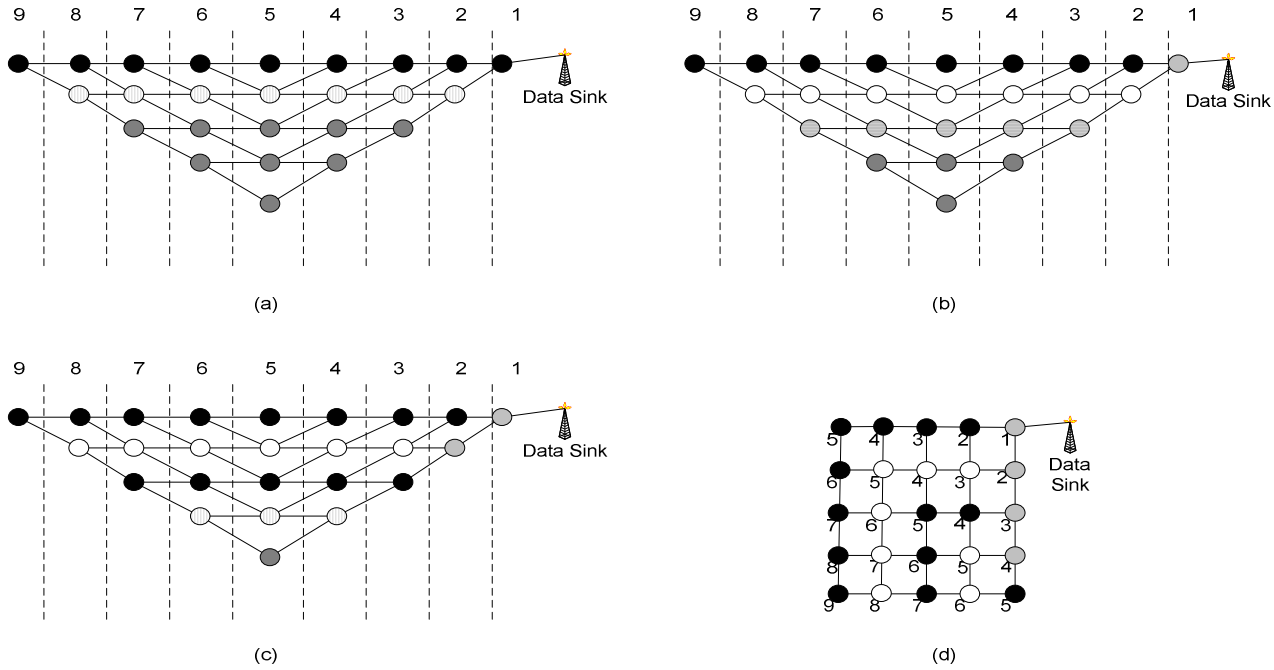

Figure 26 - Successive iterations in SS-Tree computation.
(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Final Results.

To illustrate this idea, Figure 26 shows the successive iterations in computing 2 SS-Trees for a 25-node square grid WSN, where path cost of each node is represented by its hop count to the data sink. Here, the nodes coloured in solid black and solid white represent that they are selected for SS-Tree 1 and SS-Tree 2, respectively, whereas nodes coloured in solid grey indicate they are shared nodes, which means they belong to both SS-Tree 1 and SS-Tree 2. For example, after a path is constructed for SS-Tree 1 in Figure 26(a), all of the selected nodes' neighbours, which are coloured in vertical stripes pattern, become the set of candidate nodes, or *candidate set*, from which the next path for SS-Tree 2 is based upon. Subsequently when a path is constructed for SS-Tree 2 in Figure 26(b), all of its selected nodes' neighbours will form the candidate set for SS-Tree 1, which are coloured in horizontal stripes. The path construction process continues until every node is selected, and the final SS-Tree configuration is shown in Figure 26(d).

The psuedocode for computing *k* SS-Trees on a given WSN topology is shown below, where the algorithm is divided into the main program and 2 subroutines:

Input: An adjacency list or matrix describing the complete WSN topology, where each node is aware of its hop count to the data sink.

Output: $|K|$ sets, each named S_{Sk} for $k = [1..|K|]$, where each set contains the nodes that belong to each of the SS-Trees

Variables: S_{Sk} - Selected Set for SS-Tree *k*, S_{Ck} - Candidate Set for SS-Tree *k*, S_U - Unselected Set

Initialization: $S_U \leftarrow V - s$, $S_{Sk} \leftarrow s$, $S_{Ck} \leftarrow$ NULL, for $\forall k \in K$

Program COMPUTE_SST

```
1   while ( |S_U| + Σ_{k∈K} |S_Ck| ) > 0 do
2       if Σ_{k∈K} |S_Ck| = 0
3           run MAKE_NEW_PATH
4       end if
5       for k counts from 1 to |K|
6           if (|S_Ck| > 0)
7               run MAKE_SSTk_PATH
8           end if
9       end for
10  end while
```

Subroutine MAKE_NEW_PATH

```
1   if (|S_U| = 0)
2       exit subroutine
3   end if
```

4 $i \leftarrow$ select a node in S_U with the maximum path cost
5 **while** (true) **do**
6 search in S_U for an upstream neighbour of node i
7 **if** such a node can be selected
8 $i \leftarrow$ newly selected node
9 **else**
10 **for** k counts from 1 to $|K|$
11 search in S_{Sk} for an upstream neighbour of node i
12 **if** such a node is found in S_{Sk}
13 move all previously selected nodes from S_U to S_{Sk}
14 move all unselected peer and upstream neighbours of the selected nodes from S_U to S_{Cj} for $\forall j \in K,\ j \neq k$
15 **exit subroutine**
16 **end if**
17 **end for**
18 **end if**
19 **end while**

Subroutine MAKE_SSTk_PATH

1 **if** $(|S_{Ck}| = 0)$
2 **exit subroutine**
3 **end if**
4 $i \leftarrow$ select a node in S_{Ck} with the maximum path cost
5 **while** (true) **do**
6 search for an upstream neighbour of node i
7 **if** such a node can be selected in S_{Ck}
8 $i \leftarrow$ selected node
9 remove node i from S_{Ck}
10 **else if** such a node can be selected in S_{Sk}
11 place all the previously selected nodes to $\underline{S_{Sk}}$
12 move all unselected peer and upstream neighbours of the selected nodes

from

S_U to S_{Cj} for $\forall j \in K, j \neq k$

13 **exit subroutine**

14 **else if** such a node can be selected in S_U

15 $i \leftarrow$ selected nodes

16 remove node i from S_U

17 **else if** such a node can be selected in S_{Cj} for $\forall j \in K, j \neq k$

18 $i \leftarrow$ selected node

19 remove node i from S_{Cj}

20 **else if** such a node can be selected in S_{Sj} for $\forall j \in K, j \neq k$

21 $i \leftarrow$ selected node

22 **end if**

23 **end while**

In the iterative algorithm, all of the neighbours of a given node need to be searched after it has been selected for a particular SS-Tree (e.g. Line 6 of both subroutines MAKE_NEW_PATH and MAKE_SSTk_PATH). So for a given WSN and assuming the WSN topology is implemented using adjacency lists, the worst case running time of the SS-Tree computation algorithm is in the order of $O(|V| \, |D_{\max}|)$, where $D_{\max}$ denotes the maximum degree per node in the WSN topology. Otherwise, the running time complexity could reach $O(|V|^2)$ if an adjacency matrix is used. Similarly, the memory requirement for running this algorithm is in the order of $O(|V| \, |D_{\max}|)$ when using adjacency lists, $O(|V|^2)$ for adjacency matrices.

3.3.2 ILP-Dijkstra Approach

Given a WSN topology of Figure 27(a), the main idea behind the ILP-based computation approach, named *ILP-Dijkstra*, is to first build a shortest path tree rooted at the data sink using Dijkstra's algorithm as illustrated in Figure 27(b), and then minimizes the number of shared nodes on the SS-Trees using ILP techniques, whose result is presented in Figure 27(c). In this way, the third objective of minimizing path cost is satisfied in all feasible solutions while the ILP formulation concentrates in balancing the first and second objectives of minimizing the number of shared nodes and the number of co-SS-Tree neighbours, respectively.

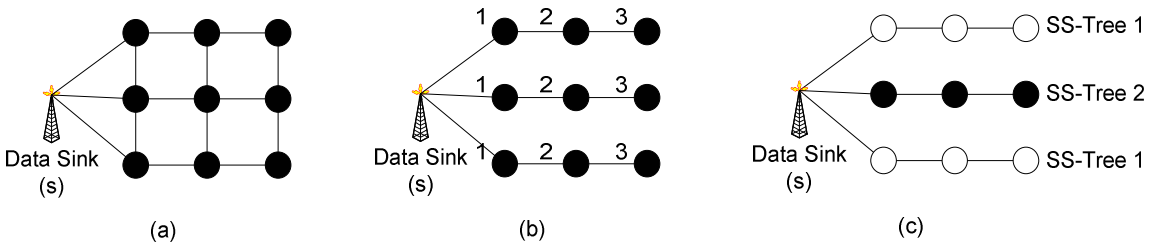

Figure 27 - ILP-Dijkstra concept.
(a) WSN topology. (b) Shortest path tree. (c) SS-Tree assignment.

Constants - The following constants with binary values are defined to describe the WSN topology, where the first constant, named, $a_{i,j}$, represents the node adjacency map of G:

$$a_{i,j} = \begin{cases} 1 & \text{if node } i \text{ is adjacent to node } j \\ 0 & \text{otherwise (including when } i = j\text{),} \end{cases}$$

where $i, j \in V$. In this model, two nodes are said to be adjacent to each other if they can establish a bidirectional radio link. However, due to the constant fluctuations in the wireless channel, it is at each node's discretion to determine the validity of a link with respect to certain predetermined signal-to-noise ratio (SNR) and bit error rate (BER) thresholds.

On the other hand, the second constant, called $u_{i,j}$, denotes the relationship between two adjacent nodes in terms of the path cost to the data sink. The purpose of this constant is to provide a means to minimize path cost as well as to guarantee network connectivity in the ILP-Dijkstra formulation. First, node j is said to be the *upstream neighbour* of node i if node j is the next upstream node on node i's shortest path to the data sink, which also implies that both nodes are also adjacent to each other. Then:

$$u_{i,j} = \begin{cases} 1 & \text{if node } j \text{ is the next upstream node on node } i\text{'s shortest path to the data sink} \\ 0 & \text{otherwise (including when } i = j\text{),} \end{cases}$$

where $i, j \in V$. The $u_{i,j}$ values are determined by referencing the computed path cost for each node after executing Dijkstra's algorithm over the WSN topology. Note that a node can have multiple upstream neighbours if it has more than one shortest path to the data sink.

Parameters - Two integer parameters are defined in the ILP-Dijkstra formulation, which are:

N_{max} - maximum number of nodes allowed in any SS-Tree
C_{max} - maximum number of co-SS-Tree neighbours allowed per node

N_{max} controls the SS-Tree size such that each SS-Tree is to be assigned with a balanced share of nodes in a uniformly distributed nodal topology. Without this constant, a feasible solution that includes SS-Trees that are over- or under-populated could be produced, which lead to unbalanced time slot allocations in the subsequent sleep scheduling process. The other parameter, C_{max}, serves the purpose of realizing the second objective of minimizing the number of co-SS-Tree neighbours per node. Obviously, a feasible solution that is generated with a C_{max} value of 2 implies that the branches of the SS-Trees are entirely comprised of linear chains. While linear chains bring forth a lot of advantages in minimizing messaging overhead, this type of SS-Tree configuration is seldom achievable in an arbitrary WSN topology without producing shared nodes. Therefore a more conservative C_{max} value of 3 or more should be applied.

Variables - The only variable used in the ILP-Dijkstra formulation defines which nodes belong to which SS-Tree:

$$sst_i^k = \begin{cases} 1 & \text{if node } i \text{ belongs to SS-Tree } k \\ 0 & \text{otherwise,} \end{cases}$$

where $i \in V$ and $k \in K$. Since for every node in the set V will have $|K|$ number of binary variables attributed to it, the total number of expected variables used in the ILP-Dijkstra formulation, N_{var}, is:

$$N_{\text{var}} = |K||V|, \tag{25}$$

Also if node *i* is to become a shared node, then the sum of all its associated sst_i^k variables for $k \in K$ would be greater than 1.

Objective Function - Minimize the number of shared nodes:

$$\min \sum_{i \in V} \sum_{k \in K} sst_i^k \ , \tag{26}$$

Constraints - Each node belongs to at least 1 SS-Tree:

$$\sum_{k \in K} sst_i^k \geq 1, \qquad \forall i \in V - \{s\} \tag{27}$$

- The number of nodes per tree is restricted:

$$\sum_{i \in V} sst_i^k \leq N_{\max}, \qquad \forall k \in K \tag{28}$$

- The data sink must belong to all SS-Trees:

$$sst_i^k = 1, \qquad \forall i = s, \forall k \in K \tag{29}$$

- The number of co-SS-Tree neighbours per node is restricted:

$$\sum_{j \in V} a_{i,j} sst_j^k \leq sst_i^k C_{\max} + (1 - sst_i^k)|V|, \qquad \forall i \in V - \{s\}, \forall k \in K \tag{30}$$

- Each node must have at least 1 upstream neighbour on the same SS-Tree:

$$\sum_{j \in V} u_{i,j} a_{i,j} sst_j^k \geq sst_i^k, \qquad \forall i \in V - \{s\}, \forall k \in K \tag{31}$$

The above constraint declarations are straightforward probably with the exception of Constraint (30), which refers to the goal of minimizing the number of co-SS-Tree neighbours per node. The left side of the constraint gives the number of neighbours of node *i* that belong to SS-Tree *k*. The right-hand portion of the constraint simply provides a condition that if node *i* also belongs to SS-

Tree k, then the summation on the left side must be less than or equal to C_{max}, which is the maximum number of co-SS-Tree neighbours allowed per node. Otherwise, that summation is less than the size of V, which essentially means that the number of neighbours of node i that belong to SS-Tree k is inconsequential for the time being. Furthermore, Constraint (31) is put together based on similar concepts to demonstrate the relationship between a node and its neighbours through a manipulation of the sst_i^k variables. In this case, the sst_i^k variable for node i cannot be greater than 1 (i.e., node i is not assigned to SS-Tree k) unless there exist at least one other upstream neighbour that belong to the same SS-Tree k as well in order to guarantee flow connectivity and minimum path cost.

3.3.3 ILP-Multicommodity Flow Approach

The second ILP-based approach, called ILP-Multicommodity Flow (ILP-MF), aims to compute the SS-Tree assignments based on the network flow theory with special emphasis on multicommodity flows. First of all, consider Figure 28, where the 9 nodes in a WSN have been assigned to 2 SS-Trees identified by their respective k values. The fact that each SS-Tree is rooted at the data sink can be viewed as having a flow of type k streaming down from the data sink to the assigned nodes, where each SS-Tree is identified by a distinct flow type as if it carries a separate commodity. Whenever a node is traversed by a flow of type k means that the node belongs to SS-Tree k, and shared nodes are created when flows of different types traverse the same node. The data sink acts as the common source for all of the multicommodity flows and it is designated by the name s in the following ILP formulation. In Figure 28(a), each node is traversed by flows of either type 1 or 2, which means that no shared nodes exist. However in Figure 28(b), the centre node is traversed by both types of flow, thereby rendering it a shared node.

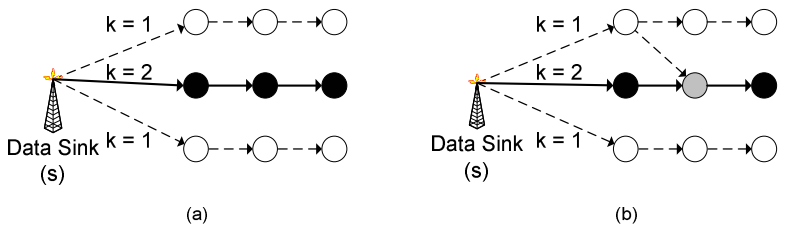

Figure 28 - ILP-Multicommodity Flow concept.
(a) No shared nodes. (b) With 1 shared node.

When working with ILP network flow models, it is important to assign one or more destination nodes to guarantee flow conservation. Simply assigning nodes at the fringes of the WSN as destination nodes is insufficient because this places undue influence on how the SS-Tree flows traverse the network, which would in turn affect the quality of the solution. On the other hand, it is difficult to account for the SS-Tree assignment of each node in the subsequent ILP computation through using a simple network flow model. To satisfy both needs, a virtual node called the *supersink*, denoted as *t*, is be added to the WSN (i.e., add node *t* to set *V*) that connects to every node in the WSN except for the data sink. Its purpose is to become the destination of all flows and its implications on variable declaration will be explained later.

The same binary constant $a_{i,j}$ and integer parameters $N_{\max}$ and $C_{\max}$ that were previously defined in Section 3.3.2 are used in the ILP-Multicommodity Flow formulation, whereas two types of flow variables are defined in this network flow model. The first type, $f^k(i,j)$, denotes the amount of flow of type k traveling from node i to node j where $i, j \in V\text{-}\{t\}$ and $k \in K$, and it takes on integer values only. For regulating the flow on the bidirectional link between 2 nodes, *skew symmetry* is defined where for an edge $(i,j) \in E$ and $k \in K$, $f^k(i,j)$ equals - $f^k(j,i)$. Here, the convention is:

$$f^k(i,j) \begin{cases} > 0 & \text{if there is a net positive flow traveling from node } i \text{ to node } j, \\ < 0 & \text{if there is a net positive flow traveling from node } j \text{ to node } i, \\ = 0 & \text{if there is no net flow traveling between nodes } i \text{ and } j, \end{cases}$$

where $i, j \in V$ and $k \in K$. Obviously, $f^k(i,j)$ is equal to 0 if nodes i and j are not adjacent in the underlying network topology, and as well it is defined that the net flow from a node to itself (i.e., $i = j$) is 0.

The second type of flow variable, $f^k(i,t)$, is restricted for representing the flows between every node $i \in V\text{-}\{s,t\}$ and the supersink t, and its purpose is to define which nodes belong to which SS-Tree:

$$f^k(i,t) = \begin{cases} 1 & \text{if a flow of 1 of type } k \text{ passes from node } i \text{ to supersink } t, \\ 0 & \text{otherwise,} \end{cases}$$

where $i \in V\text{-}\{s,t\}$ and $k \in K$. In short, the binary $f^k(i,t)$ variable is specialized for the unique characteristics in the network flow model such that it uses the binary flow concept to manage the SS-Tree assignment of each node. On the other hand, the integer $f^k(i,j)$ variables just account for all the flows of all types destined to the supersink and play a lesser role in satisfying the ultimate minimization objectives.

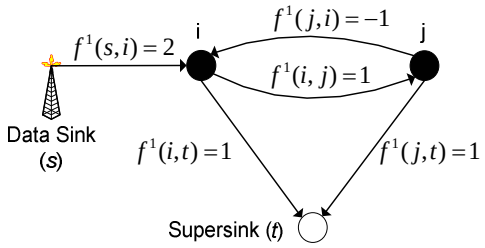

Figure 29 - Flow variable relationship in ILP-Multicommodity Flow approach.

To illustrate the relationship between the variable types $f^k(i,j)$ and $f^k(i,t)$ in the overall network model of the ILP-Multicommodity Flow approach, Figure 29 shows a simple 2-node WSN connected to the data sink *s* and the virtual supersink *t*. Suppose both nodes belong to SS-Tree 1 (i.e., $K = \{1\}$), then both $f^1(i,t)$ and $f^1(j,t)$ would be equal to 1 according to the above variable type definition. To compensate for this flow demand and maintain flow conservation, a net flow of 2 of type 1 must be supplied by the data sink to node *i* and then onward to node *j*, which means $f^1(s,i)$ equals 2 and $f^1(i,j)$ equals 1. Therefore for a given node $i \in V$ (including the data sink and the supersink), the net flow exiting the node is:

$$\sum_{j \in V} f^k(i,j) \begin{cases} > 0 & \text{for the data sink (i.e., } i = s\text{),} \\ < 0 & \text{for the supersink (i.e., } i = t\text{),} \\ = 0 & \text{for all other nodes,} \end{cases}$$

where $k \in K$. Also according to the skew symmetry property, $f^1(j,i)$ becomes -1 in Figure 29.

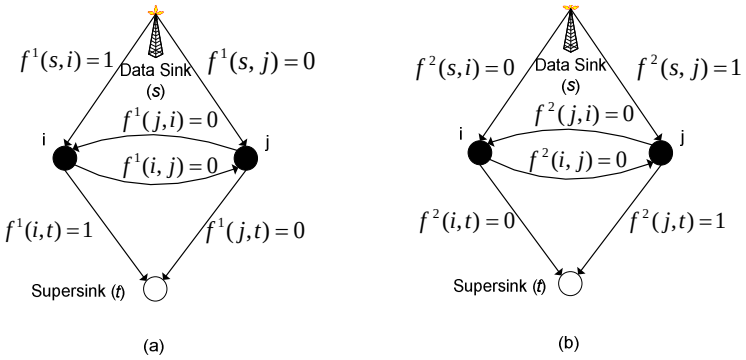

Figure 30 - ILP-Multicommodity Flow model for multiple SS-Tree assignment.
(a) Flow variables for SS-Tree 1. (b) Flow variables for SS-Tree 2.

Figure 30 further explains the use of multicommodity flow variables in determining the nodal assignments for multiple SS-Trees. Suppose that another 2-node WSN is connected to the data sink in a slightly different manner than the previous example, and this time nodes *i* and *j* are assigned to SS-Trees 1 and 2, respectively (i.e., $K = \{1, 2\}$). Flow variables of type 1 are collectively shown in Figure 30(a) to show the association of each node to SS-Tree 1. Likewise, Figure 30(b) displays the flow variables of type 2 to demonstrate each node's relationship to SS-Tree 2. If node *i* is to be assigned to SS-Tree 2 in addition to SS-Tree 1 as well, then both variables $f^1(i,t)$ and $f^2(i,t)$ would become 1. Therefore, a shared node *i* is identified by the fact that the sum of all its associated $f^k(i,t)$ variables for $k \in K$ is greater than 1.

In terms of the problem size, the number of variables, N_{var}, involved in the ILP-MF approach , is

$$N_{\text{var}} = |K||V-\{s,t\}| + 2|E||K|, \tag{32}$$

where the first ($|K||V-\{s,t\}|$) term accounts for the type of binary variables $f^k(i,t)$ and the latter ($2|E||K|$) term refers to the integer $f^k(i,j)$ variable type.

Objective Function - Minimize the number of shared nodes:

$$\min \sum_{k \in K} \sum_{i \in V - \{s,t\}} f^k(i,t), \tag{33}$$

Constraints - Each node belongs to at least 1 SS-Tree:

$$\sum_{k \in K} f^k(i,t) \geq 1, \qquad \forall i \in V - \{s,t\} \tag{34}$$

- The number of nodes per tree is restricted:

$$\sum_{i \in V - \{s,t\}} f^k(i,t) \leq N_{\max}, \qquad \forall k \in K \tag{35}$$

- The data sink is the source of all flows:

$$\sum_{k \in K} \sum_{j \in V - \{s,t\}} f^k(s,j) \geq |V - \{s,t\}| \tag{36}$$

- Flow conservation:

$$\sum_{j \in V} f^k(i,j) = 0, \qquad \forall i \in V - \{s,t\}, \ \forall k \in K \tag{37}$$

- Skew symmetry:

$$f^k(i,j) = -f^k(j,i), \qquad \forall i,j \in V, \ \forall k \in K \tag{38}$$

- A node belongs to SS-Tree *k* if it is traversed by a flow of type *k*:

$$f^k(i,j) \leq f^k(i,t)|V|, \qquad \forall i \in V - \{s,t\}, \ \forall j \in V - \{t\}, \ \forall k \in K \tag{39}$$

- The number of co-SS-Tree neighbours per node is restricted:

$$\sum_{j \in V - \{s,t\}} a_{i,j} f^k(j,t) \leq f^k(i,t) C_{\max} + \left(1 - f_i^k(i,t)\right)|V|, \qquad \forall i \in V - \{s,t\}, \ \forall k \in K \tag{40}$$

Constraint declarations (34) to (36) are straightforward, and flow conservation and skew symmetry properties of the network flow model are maintained by Constraints (37) and (38), respectively. However, special consideration is to be given to Constraint (39), which requires that a node must belong to SS-Tree *k* if a flow of type *k* traverses the node. The reasoning behind this constraint is given in Figure 31, where a 3-node chain WSN is connected to the data sink as well as the virtual supersink. If nodes *h* and *j* are assigned to SS-Tree 1, then without the restrictions placed by Constraint (39), the flow variables could be allotted values in the manner shown in Figure 31(a), where $f^1(i,t)$ is equal to 0 with a flow of type 1 passing through node *i*. This flow allocation may be perfectly valid in other types of network flow systems and none of the other ILP constraints have been violated. As illustrated in Figure 31(b), however, the physical meaning of this SS-Tree assignment is that node *i* would not be sharing the same sleep schedule with nodes *h* and *j*, thereby severing the multihop link during the latter nodes' respective active periods. To rectify this assignment error, Constraint (39) limits all net inflow into a node to less than 0 unless $f^2(i,t)$ is 1 for all SS-Trees and for node *i* not being the data sink or the supersink. Therefore, the combination of flow conservation and this constraint would guarantee the correct flow assignment, as demonstrated in Figure 31(c) and (d).

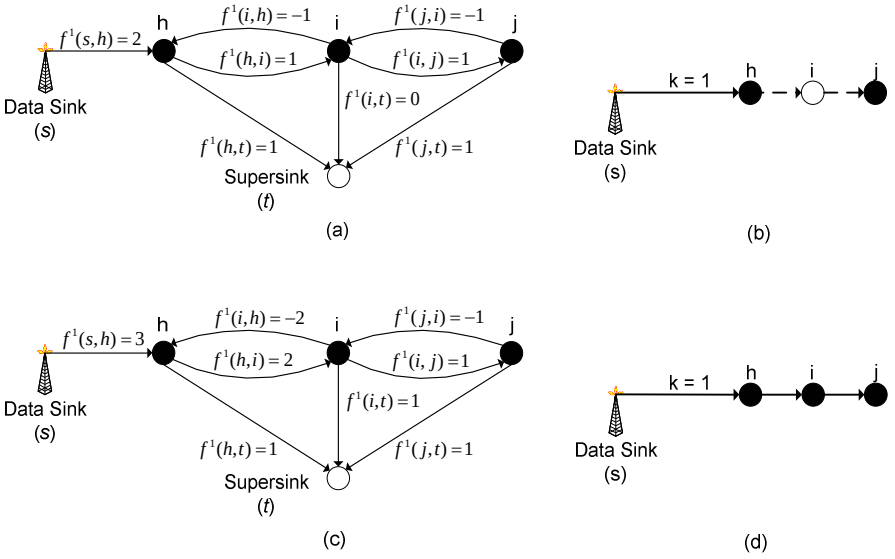

Figure 31 - Explanation on the SS-Tree flow traversal constraint.
(a) Incorrect flow assignment. (b) Physical meaning of incorrect flow assignment.
(c) Correct flow assignment. (d) Physical meaning of correct flow assignment.

Furthermore, Constraint (40) refers to the goal of minimizing the number of co-SS-Tree neighbours per node. The left side of the constraint gives the number of neighbours of node *i* that belong to SS-Tree *k*. The right-hand portion of the constraint simply provides a condition that if node *i* also belongs to SS-Tree *k*, then the summation on the left side must be less than or equal to C_{max}, which is the maximum number of co-SS-Tree neighbours allowed per node. Otherwise, that summation is less than $|V|$, which essentially means that the number of neighbours of node *i* that belong to SS-Tree *k* is inconsequential for the time being.

3.4 SS-TREE OPERATIONAL SPECIFICS AND SLEEP SCHEDULING

After SS-Trees are computed, the next major task is to determine an optimal sleep schedule that maximizes energy efficiency. As will be shown later in Section 4.1, using sleep schedules with short active period duration that only permits data to travel over one or a few hops per active period will incur phenomenally high transmission latency. Therefore the use of longer active periods is preferred so that all of the packet exchanges can be completed within fewer cycles for a given duty cycle. Furthermore, if all of the end-to-end data exchanges between the data sink and the nodes can be completed within a single active period, then network control functions such as time synchronization and sleep schedule updates can be implemented with less difficulty. While longer active period lengths have the additional advantage of requiring less stringent time synchronization requirements, they will increase the amount of sleep time between two consecutive active periods, which in turn affects how often sensing applications can generate their data. Therefore, it is imperative to determine an upper bound of active period duration in order to balance the low communication duty cycle, monitoring sensitivity, and end-to-end packet transmissions.

3.4.1 Network Routing

Since periodic link state updates for all of the sensor nodes are very expensive in terms of energy usage for large WSNs, a more energy-efficient packet delivery solution is preferred where different routing strategies may be employed to exploit the asymmetric upstream and downstream WSN traffic patterns. The proposed SS-Tree design streamlines the routing procedures by restricting individual sensor nodes to only maintain local connectivity information of its immediate 1-hop neighbours, whereas the data sink is given the sole right to compute the

global network connectivity map from the link state information gathered from the sensor nodes.

Without the availability of global link state information, sensor nodes will rely on minimum cost forwarding [25] for sending packets to the data sink. Through locally exchanged connectivity information, each sensor node becomes aware of the cost of forwarding data packets to the data sink via each of its neighbours. On the other hand, the data sink can use source routing [8] for all types of downstream traffic, namely unicast, multicast and broadcast, where the routing path is to be explicitly listed in the packet header. However because of the potentially high hop counts in end-to-end paths, the packet header will become enormously large compared to the actual data sent, thereby inflicting substantial transmission costs. Therefore source routing should be refrained from use in normal sensing operation and reserved only for special occasions such as network initialization and failure recovery. For more energy-efficient downstream dissemination, SS-Trees, which are essentially spanning tree structures, can be adapted for multicast and broadcast communications.

Since accurate time synchronization cannot be guaranteed in large WSNs, guard bands should buffer each active period to compensate for potential clock drifts along the entire path, as shown in Figure 32. The first guard band, T_{G1}, compensates for common time synchronization errors amongst sensor nodes, whose upper bound depends on the type of synchronization method, and clock drifts occurred during the preceding sleep period, which typically reside in the neighbourhood of milliseconds for sleep periods lasting several minutes or more. After T_{G1}, the necessary packet forwarding activities would commence and should be completed within a length of T_{PP}, where the exact packet exchange sequencing will be discussed in Section 3.4.2. The second guard band, T_{G2}, accounts for timing overshoots of the packet forwarding period due to packet collisions and other unexpected events that cause end-to-end transmissions fail to complete within T_{PP}. Since T_{G1} is mainly deterministic in nature, it would very much be predictable and likely to be the shortest in duration out of the three active period partitions. Both T_{PP} and T_{G2} depend on monitoring requirements, traffic patterns, processing overhead and network topology, which makes calculating the perfect timing allocation for active periods difficult because of the high variability of the different factors. Therefore, the active period should be given a more liberal share of time in the initial sleep schedule and its length is to be

adjusted dynamically in response to ongoing network performance measurements such as roundtrip time and packet collision rate.

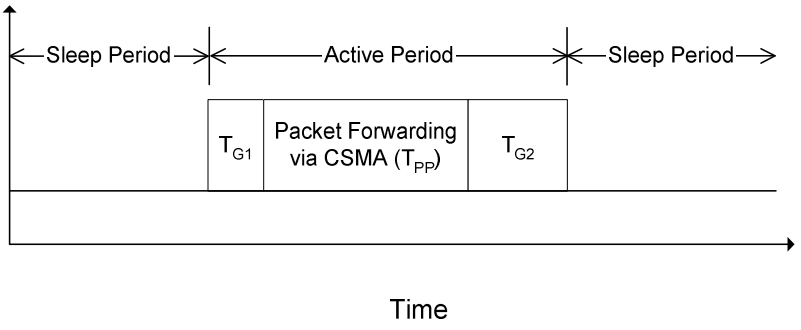

Figure 32 - Organization of active periods.

Figure 33(b) illustrates an example for coordinating sleep schedules for a 3-hop routing path shown in Figure 33(a), where the arrows at the bottom indicate potential transmission times at Node 3 for packets destined to Node 1 via Node 2. Arrows A and D are definitely dreadful timing choices for packet transmission because they reside in times when all of the nodes are asleep. Then around the start of a particular active period, all 3 nodes will wake up at roughly, but not exactly, the same time because of imperfections in time synchronization and clock drifts happened in the previous sleep period. The optimal transmission window occurs around Arrow B, where all the nodes have entered the active period and the packet should have enough time to traverse the 2 hops within T_{PP} as shown by the slanted dashed lines, assuming the assorted transmission delays are much shorter than the active period duration. Since Arrow C begins transmission at the latter part of the active period, its delivery cannot be guaranteed within a single active period even with the extra buffering of T_{G2}. Therefore the packet may have to be intermediately cached at Node 2 and forwarded to Node 1 at the next active period, which may be many minutes away. On the other hand, if sporadic packet losses can be tolerated, Node 2 can discard the packet at the end of the active period if it cannot be delivered to Node 1 in time, thereby cutting down energy usage. To prevent packet loss of this type, an accurate assessment of T_{PP} and T_{G2} as well as careful data traffic coordination in response to sleep schedules should be implemented.

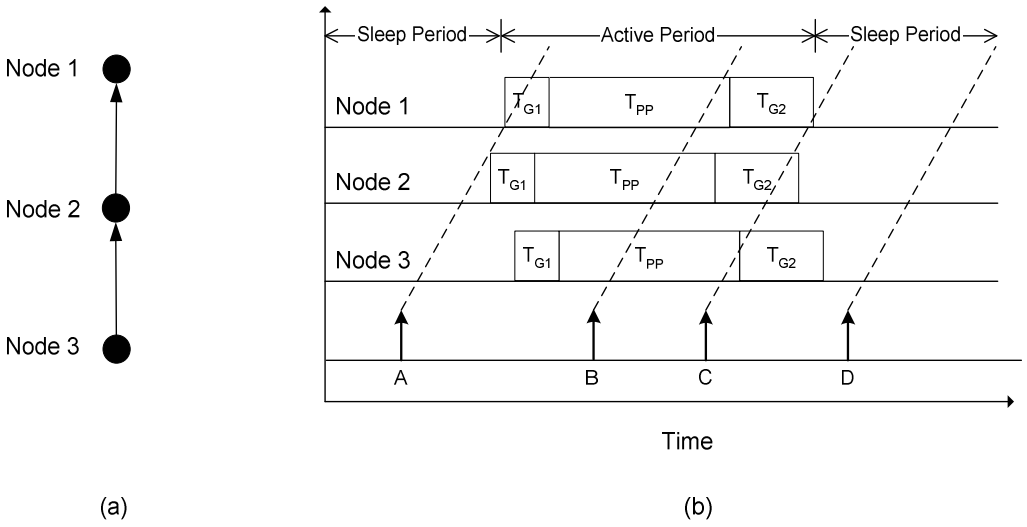

Figure 33 - Coordinated sleep scheduling for multihop routing paths.
(a) Routing path configuration. (b) Sleep schedules and packet transmission timing.

3.4.2 Sensing Requirements and Traffic Engineering

In order to complete all packet transmission and forwarding activities within a single and short active period, the data reporting process should also be carefully choreographed to further reduce management overhead instead of letting the sensor nodes transmit packets at will. Since the objective of WSN applications is to oversee all sensor nodes cooperating together to execute common data reporting tasks instead of cater to thousands of terminals with individual QoS guarantees, more flexibility and control exist in manipulating application requirements and data flow patterns to suit dynamic operational situations. For example, in order to reduce hop-by-hop transmission time, each data reporting packet can shrink in size by formatting data queries to solicit Boolean answers (e.g. Is ambient temperature above 30°C? → yes/no) rather than absolute values (e.g. What is the ambient temperature? → 25.75°C). Besides speeding up data transmission, a smaller packet size would not require segmentation and reassembly services in lower layers, as well as decreases the instances of buffer overflow at intermediate nodes and reduces packet loss as a result.

In addition to compact query formats, aggressive data aggregation and duplicate suppression will be enforced so that the processing load will be distributed as evenly as possible along routing paths and across the sensing field to prevent premature battery depletion, especially for nodes

located closer to the data sink. For example, suppose that each sensor node is required to report their circuitry well-being to the data sink each hour, where a positive response would indicate the node is alive and operational. Instead of generating separate packets, nodes on the same routing path can collectively rely on a single *seed reply* packet and transmit the same copy upstream, starting from the node furthest away from the data sink. If the node is experiencing no operational problems, then it will simply forward the incoming seed reply packet to its immediate upstream neighbour on the routing path without altering the packet's contents. Since the data sink possesses the global connectivity knowledge, receiving this single positive response packet would show that all the nodes along that path are faring well. Again, a spanning tree structure would be ideal for performing in-network processing, and some timing coordination in data generation such as timer-driven data reporting would be very helpful in achieving efficient data aggregation and duplicate suppression.

While timer-driven data reporting is favoured because of its simplicity and periodicity, event-driven and request-driven types should also be accommodated within the traffic engineering framework. Excluding false alarms, event-driven data reports triggered by abnormal events are to be given a high priority such that their deliveries can be expedited to the data sink with high fidelity at minimal packet loss. In contrast, occasional losses of environmental readings and hardware status reports, though also undesired, will not seriously compromise event detection capabilities. Still, such packet losses, regardless of request or timer-driven nature, do signify possible nodal failures or traffic load imbalance that require the immediate attention of the data sink for network modifications or repairs, and the failure recovery mechanism will be discussed in Section 3.4.4.

In face of aggressive data aggregation and duplicate suppression in the proposed WSN design, providing end-to-end ACKs by the data sink for regular data reporting packets is rather infeasible, especially under an ultra-low communication duty cycle. Also, buffer overflow, the common culprit in causing packet loss in the Internet, is deemed virtually non-existent in the envisioned WSNs because of small packet size and low traffic volume, thereby drastically diminishes the role of end-to-end ACKs. Therefore, energy expenditure can be further reduced by advocating the use of hop-by-hop ACKs in the MAC layer instead of end-to-end ACKs in the

transport layer. However, hop-by-hop ACKs cannot always guarantee packet delivery success because a packet will get stranded on an intermediate node due to the premature expiration of the active period or a path blockage from nodal failures upstream. To limit energy use and to simplify the recovery process, packets with low priority such as timer-driven environment data readings and status reports will be discarded, while high priority packets such as event alert messages will be cached and forwarded to the data sink at the next available active period. If the processing centre indeed wishes to obtain the data contained in the discarded packets, it can always issue new data requests to the corresponding sensor nodes. Given the assertion of meticulous sleep schedule planning and the low probability of nodal failures occurred during the short active period, such blatant packet discards will likely be uncommon.

Unlike event-driven data reporting, the reporting frequency of timer-driven types can be preset so that it coincides with each active period, depending on application requirements. After every node along a routing path wakes up at the start of each active period, intuitively the node with the largest hop count (i.e., at the end of the path) would automatically send a seed reply packet to initiate the data aggregation and duplicate suppression processing along the path. However, since the WSN may encounter unexpected nodal failures, experience considerable clock drifts, or detect new sensor node additions during the long sleep periods, the first task the sensor nodes should perform at the start of the active period is to assess any changes in the neighbourhood topology and to keep each other's onboard clock in sync. Still, even the simple act of exchanging *Hello* and *Sync* messages with neighbours will occupy a sizable portion of the active period, not to mention the possibility of further delays due to packet collisions with legitimate data packets and other control traffic.

To better streamline the active period initialization process, the data sink should assume a much more involved role in coordinating topology maintenance and time synchronization functions, and the following example illustrates the proposed approach in doing so. After the start of the active period, the data sink should send a network probing packet called a token down every routing path to detect any link breakages, as shown in Figure 34(a). At the tree junction points, the token will be broadcast to all downstream nodes, effectively splitting the single token into multiple copies to be pushed down each branch. When a node cannot reach its immediate

downstream neighbour to forward the token, this would indicate a nodal failure has occurred due to depleted battery, hardware malfunction, off-sync sleep scheduling, or worsened radio conditions. The node should then instantly report back the failure discovery to the data sink, where the appropriate recovery procedures, which will be presented later in Section 3.4.4, will be undertaken. Otherwise if every node is functioning properly, then the nodes at the fringes of the network will transmit back seed reply packets right after they receive the tokens, as in Figure 34(b), where upstream and junction nodes will perform traffic merging and in-network processing on the seed reply packets. Similarly, since time synchronization packets have to be periodically distributed throughout the WSN to offset clock drifts, they can be combined with the tokens to minimize transmission overhead, and the resultant packet size will be very short for accelerated forwarding and processing as they only contain a small packet header and a global timestamp. The act of sending the tokens downstream and the subsequent response of the seed reply packets will be referred to as the *push-pull traffic sequencing* henceforth.

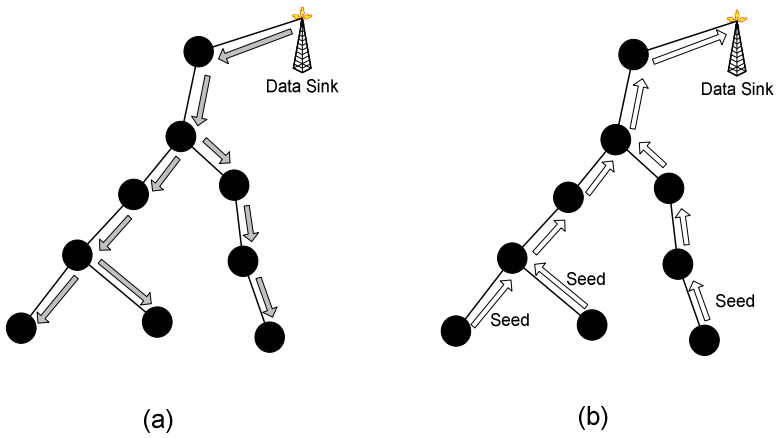

Figure 34 - Push-pull traffic sequencing for control and data packets in active period.
(a) Downstream token. (b) Upstream seed reply.

In addition to accommodating control packets for failure detection and time synchronization, push-pull traffic sequencing can certainly incorporate other control message types such as data requests, link state updates, low battery notification, and sleep schedule updates, with slight procedural changes. Normally during steady state operations, these control messages are to be mixed in with upstream data traffic to contend for limited bandwidth during the short active periods. Since these control packets are vital in maintaining monitoring capabilities and network

connectivity, any transmission delay or packet loss should be minimized or avoided if possible. For example, a sleep schedule update packet would become stale and useless if its delivery is bogged down by packet collisions and channel access delays. To minimize delays, data requests and sleep schedule updates can be piggy-backed or even incorporated into the token packets as well at the start of the active period. Although this may increase transmission and processing overhead if these additional control messages are not intended for a broadcast or multicast audience, their infrequency in generation deftly offsets the negative effects. Similarly, infrequent uplink control messages such as link state updates and low battery notification as well as event-driven data packets can also latch onto the seed reply packet for maximum communication efficiency.

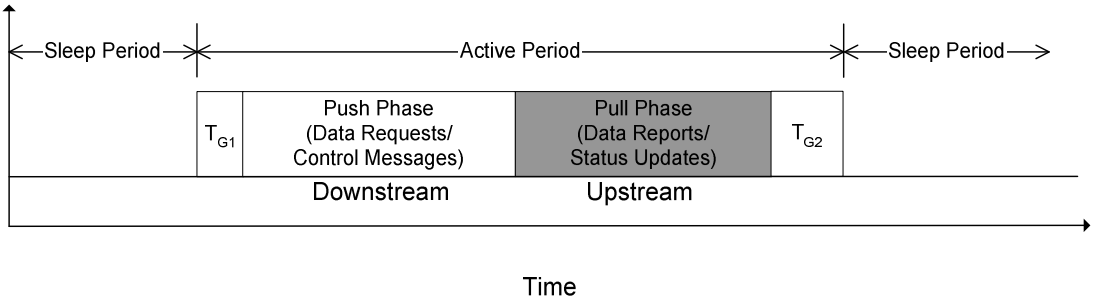

Figure 35 - Active period time slot partitioning for push-pull traffic sequencing.

Figure 35 shows the partitioning of the T_{PP} portion of the active period into two phases, where each part has enough time allocated for the push-pull traffic sequencing to traverse end-to-end on the longest routing path in either downstream or upstream direction, with the exact timing demarcation of the two halves not explicitly defined. The push-pull traffic sequencing begins when the data sink sends the token downstream T_{G1} seconds after the start of the active period. The direction of traffic flow on the spanning tree reverses sometime in between when the fringe nodes receive the tokens and respond with the seed reply packets. Hopefully all of the data reporting activities can be completed before the active period encroaches into the T_{G2} buffering area. The advantage of having downstream control and data request traffic preceding upstream data reporting and status update traffic is that the former can always verify entire path is free of node failures via forwarding tokens. Also since the simultaneous convergence of unorganized

downstream and upstream traffic over the WSN will inevitably raise the likelihood of packet collisions, allowing the packet traffic to only flow in one direction during a particular time slot would greatly alleviate the packet collision problem, though the improvement may come at the expense of potential channel under-utilization, which is of lesser priority in WSN design.

On the other hand, the use of push-pull traffic sequencing makes timer-driven data reporting behaving much like request-driven data reporting running on a regular schedule. Instead of letting sensor nodes spontaneously transmit data packets according to a predetermined timer, the tokens transmitted by the data sink in the push phase act as data requests to solicit the data reporting packets to be sent upstream in the pull phase, which happens periodically according to the sleep schedule. While timer-driven data reporting incurs fewer messaging overhead because of its periodicity and the absence of downlink control messages, keep in mind that other network control aspects such as routing path integrity, time synchronization, and sleep scheduling have to be accounted for in the overall design. As a result, a compromise is reached to incorporate data reporting tasks and network maintenance functions into the push-pull traffic sequencing model, where both will adhere to a periodic sleep schedule but with the timer-driven data generation spontaneity notion removed.

With respect to end-to-end delivery reliability in face of premature ending of the active period, lost downstream control packets can always be resent by the data sink if no seed reply has been received. However for sleep schedule updates, since some packet loss would cause nodes on the same routing path to lose partial connectivity, nodes involved in updating sleep schedules will not enact upon them until the nodes themselves receive the seed reply from downstream to confirm the receipt of updated sleep schedules of downstream peers. On the other hand, upstream control messages will need to be cached in intermediate nodes if they cannot be delivered within a single active period since no upstream end-to-end transport mechanism is to be implemented. Nevertheless, to prevent packet loss from happening at all, the active period should be allocated with plenty of time to let every packet to be delivered within a single active period.

3.4.3 Medium Access Control and Sleep Scheduling

Due to the extra messaging overhead in maintaining accurate time synchronization and managing channel assignment in face of the low communication duty cycle and hardware cost restrictions, single-channel unslotted CSMA is preferred over contentionless methods such as FDMA, TDMA, and CDMA for channel access during active periods of the sleep schedule because of its simplicity, greater scalability, and looser time synchronization requirements. While the RTS/CTS mechanism in CSMA/CA is effective in preventing the hidden node problem during channel contention, it increases the overall end-to-end propagation delay which in turn affects the monitoring sensitivity, especially when the length of data reporting packets is less than RTS packets [7]. Because of the long end-to-end propagation delay and the low volume of traffic in the envisioned WSN for wide-area surveillance, steady state data exchanges can bypass the RTS/CTS handshake as traffic management techniques discussed in Section 3.4.2 effectively reduces packet collisions through streamlining overall data traffic flows.

Putting all of the cross-layer considerations described in Sections 3.4.1 and 3.4.2 together, the timing components that constitute a single active period shown in Figure 32 can be determined. Assuming the duration of the active period, T_{active}, is the same for each SS-Tree, then T_{active} can be represented by:

$$T_{active} = T_{G1} + T_{PP} + T_{G2}, \tag{41}$$

where T_{G1} accounts for clock drifts, time synchronization errors and hardware switching times, T_{PP} deals with time expended during push-pull traffic sequencing, and T_{G2} buffers the any timing overshoots. For standard crystal oscillators with well-known time synchronization methods, T_{G1} is largely deterministic. On the other hand, T_{PP} can be approximated according to round-trip time calculations at the data sink during the *Network Initialization* phase such that:

$$T_{PP} \cong \max_{i \in V} \left(RTT^{i} \right), \tag{42}$$

where RTT^i is the round-trip time recorded for node i on its respective SS-Tree. Due to the use of downstream flooding and upstream minimum cost forwarding over a large and dense WSN,

the initially collected *RTT* values may not reflect a concise round-trip time measurement since it includes delays caused by random back-off timers and packet collisions. This timing inaccuracy will affect how T_{G2} is determined because the purpose of T_{G2} is to compensate for all the abnormalities during push-pull traffic sequencing such that packet loss due to premature completion of the active period can be minimized. While a longer T_{G2} will certainly diminish the chances of encountering timing overshoots, it will reduce monitoring sensitivity such that the event reporting windows will appear less frequently for adjacent SS-Trees. On the other hand, constant fine-tuning of the sleep schedule through issuing sleep schedule update packets downstream would produce high messaging overhead, thereby consuming consider-able amounts of energy. Therefore, empirical *LRTT* measurement data should be complemented with some mathematical guidelines for calibrating the timing allocation of T_{PP} and T_{G2}. The ultimate goal is to minimize T_{AP} in order to increase the monitoring sensitivity while ensuring push-pull traffic sequencing and event reporting can be accomplished within a single active period.

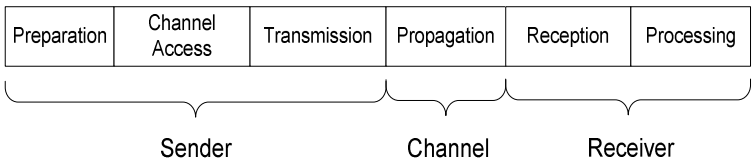

Figure 36 - Sources of delay in packet delivery over wireless link.

To produce a rough mathematical estimate of T_{PP}, a routing path of N_{hop} hops is first subdivided into single hops, where the sources of delay in packet delivery over each hop are further decomposed into individual components, as shown in Figure 36 [22]. In a 1-hop transmission, the delay components are:

- *Preparation*: Before actually sending the packet, some time is spent by the sender in handling software commands and setting hardware interrupts for data preparation. Its nature is highly variable as it depends on the type of Operating System (OS) software, packet type, and packet size.
- *Channel Access* - Since no RTS/CTS scheme is to be used because of its high messaging overhead and drastic increase in end-to-end delays, this component can be much reduced as the node instantly gains access to the wireless channel, assuming no ensuing packet collisions.

On the other hand, channel contention introduces some variability in the time used, which increases along with the number of immediate neighbours.

- *Transmission* - This largely deterministic component concerns the time needed to transmit every bit of the packet through the sender's radio transceiver, which can be estimated using radio speed and packet size.
- *Propagation* - This deals with the minute amount of time needed for each bit to traverse the wireless link from sender to receiver, which is negligible in comparison to other delay components.
- *Reception* - This refers to the time spent in receiving every bit from the wireless channel and reconstructing the packet for further processing, which is also mainly deterministic as it depends on radio speed and packet size.
- *Processing* - After the validity of the received packet has been verified, the processor will decode the packet information and decide on appropriate actions. This last timing attribute is highly variable as it depends on the packet type and the software commands executed during processing.

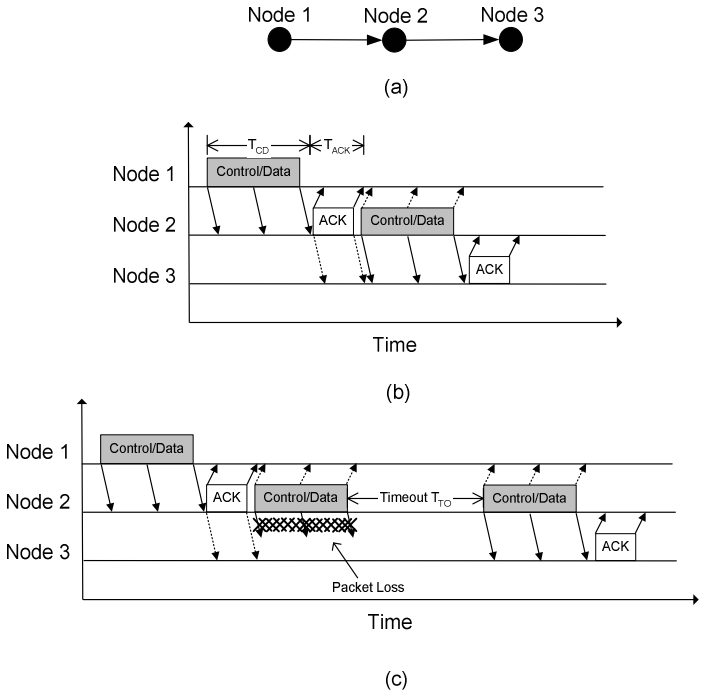

Figure 37 - C/D packet delivery delay analysis over multihop links.
(a) Network configuration. (b) Timing diagram of a successful delivery.
(c) Timing diagram of a successful delivery with 1 retransmission.

Even though uncertainties exist in the choice of processor, OS software and radio components to be implemented on the sensor nodes, the control or data packets (*C/D packets*) traversing the WSN are likely to be extremely short (< 10 bytes) in order to minimize preparation, transmission, reception, and processing delays. The short C/D packet size, along with topology simplification through SS-Trees and aggressive traffic engineering, also helps to drastically reduce the likelihood of packet collisions. On the other hand, for an SS-Tree whose branches are shaped in the form of linear chains, the cause of packet loss is mostly due to channel errors during the push-pull traffic sequencing.

Figure 37(b) demonstrates the timing sequence of a simple Control/Data (C/D) packet exchange along a multihop 3-node chain shown in Figure 37(a), where a data-acknowledgement-timeout mechanism ensures safe packet delivery in face of poor radio channel conditions. Due to the open wireless medium, the dotted arrows indicate overhearing by adjacent nodes of packets not intended for them. When a corrupted C/D packet is received by Node 3, as shown in Figure 37(c), no ACK will be sent back to Node 2. As a result, another copy of the C/D packet is transmitted by Node 2 after a timeout period. Summarizing all of the associated delays shown in Figure 36 for analyzing timing delays in Figure 37, let the total time for processing and delivering each C/D packet be T_{CD}, the total time for processing and delivering a single ACK be T_{ACK}, and the duration of each timeout period be T_{TO}. If there is no packet loss present, then the time to complete the exchange over 1 hop on a linear chain, T_{hop}, can be simply referred to as:

$$T_{hop} = T_{CD} + T_{ACK} . \tag{43}$$

When packet corruption occurs, the receiver will not generate an ACK and the sender would automatically retransmit the same C/D packet after a random timeout with mean value of T_{TO}. To analyze the impact of packet corruption on end-to-end propagation delay, let the generalized packet delivery success rate is p. Then the probability that the C/D packet transmission will be successful on the j^{th} try can be represented as a simple geometric distribution. The expected number of tries before a successful delivery, E, can therefore be given by:

$$E = \sum_{j=1}^{\infty} j(1-p)^{j-1} p = \frac{1}{p}. \tag{44}$$

Factoring in parameters T_{CD}, T_{ACK}, and T_{TO} and assuming zero correlation of packet delivery success rate for consecutive hops, the expected duration of each C/D packet delivery is:

$$T_{hop} = \frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD} + T_{ACK}. \tag{45}$$

For a regular push-pull traffic sequencing exchange with a downstream token and an upstream seed reply, suppose both types of packets are equally sized and require the similar preparation and processing time. Then the estimated *RTT* on a path of N_{hop} hops is:

$$RTT = 2N_{hop}\left(\frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD} + T_{ACK}\right). \tag{46}$$

While any lost C/D packets can be easily recovered after a timeout, the retransmission mechanism will not be effective in an open multihop environment if each timeout period is too short to combat ACK packet losses. Figure 38 shows how a single ACK packet loss at point A can trigger future packet collisions with the absence of the RTS/CTS mechanism. After Node 1 fails to receive an ACK from Node 2 for the correctly received C/D packet, it will retransmit another copy of the C/D packet after a short timeout at point B in which it also has to wait for a clear channel through carrier sensing. On the other hand, Node 2 has already forwarded the C/D packet to Node 3, where the corresponding ACK is replied at point C. Without using the inefficient RTS/CTS mechanism to prevent the notorious hidden node problem, the retransmitted copy of the C/D packet will collide with the incoming ACK reply from Node 3, thus wasting the entire packet exchange sequence. Therefore, in order to prevent such circumstances from happening on a linear chain, the timeout period T_{TO} has to be at least:

$$T_{TO} \geq T_{CD} + 2T_{ACK}. \tag{47}$$

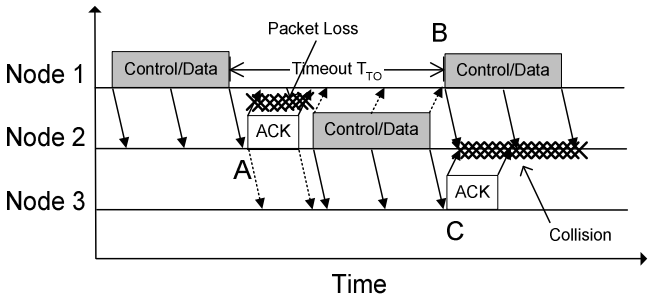

Figure 38 - Impact of ACK losses in face of short timeout periods over multihop links.

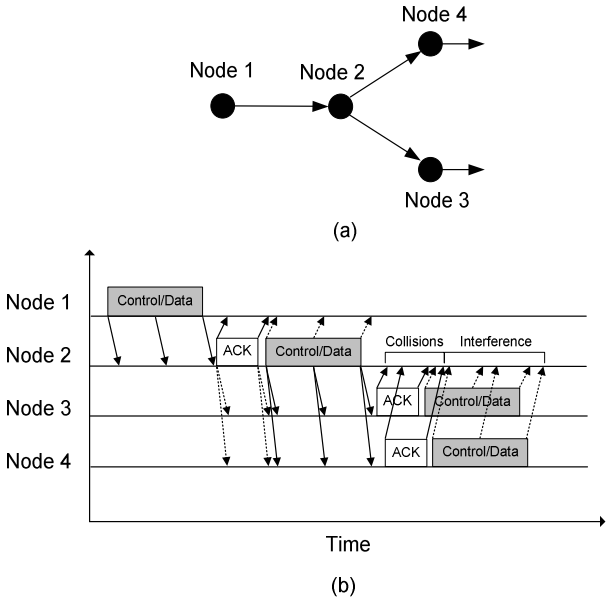

Figure 39 - Effects of packet collision at SS-Tree junction point.
(a) Network configuration. (b) Timing diagram.

Another concern with the open wireless medium is that additional care has to be taken when passing tokens and other packets downstream across SS-Tree junction points to bypass the hidden node problem. Figure 39(a) and (b) respectively show a simple SS-Tree junction configuration and its corresponding timing diagram where a C/D packet originated at Node 1 will be broadcast at junction Node 2 to its downstream descend-ants Node 3 and Node 4. Afterwards, Node 3 and Node 4 will forward the C/D packet to their respective downstream descendents. Without any coordination in the acknowledgement sequence and assuming both Node 3 and Node 4 cannot detect each other through carrier sensing, the ACK packets from

Node 3 and Node 4 would collide at Node 2. Since Node 2 subsequently cannot confirm if the C/D packet deliveries were successful or not, it has to rebroadcast the same packet after a timeout, thereby consuming more energy and creating even more potential collisions. Other than to minimize the number of junction points on the SS-Trees or resort to expensive RTS/CTS mechanism, solutions such as separating the C/D packet broadcasts into unicast links and introducing random timers for ACK replies are worth exploring further, though they may also increase propagation delay and control overhead without completely eliminating the hidden node problem.

Since the combination of wireless medium openness and multihop communications introduce the unpleasant effect of hidden terminal problem that will eventually lead to a decrease in monitoring sensitivity through the lengthening of active periods, it would be conducive to explore other approaches to expedite the push-pull traffic sequencing mechanism while maintaining hop-by-hop packet acknowledgements. One way is to implement *implicit acknowledgements* (IACKs), where an over-heard data packet also acts as an acknowledgement when it is being forward along a path. Figure 40 shows an example of mixing IACKs and explicit acknowledgements (EACKs) when passing a single C/D packet along the same short chain as in Figure 37(a). After Node 1 passes the C/D packet to Node 2, Node 2 in turn will directly forward it to Node 3 without sending an EACK. Because of the open wireless medium, Node 1 will overhear the C/D packet sent to Node 3, thereby indirectly acknowledging its safe arrival earlier at Node 2. On the other hand, Node 3 will need to send an EACK back to Node 2 because it is at the end of the routing path and has no other neighbour to send the C/D packet to.

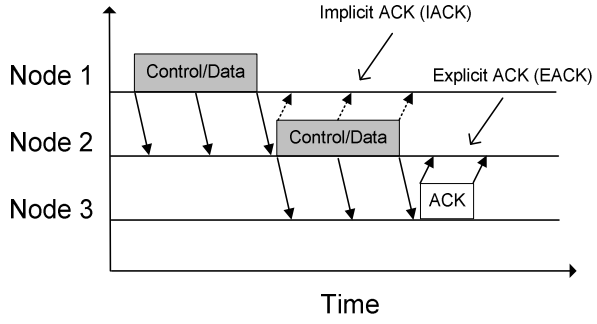

Figure 40 - Use of implicit ACK (IACK) and explicit ACK (EACK).

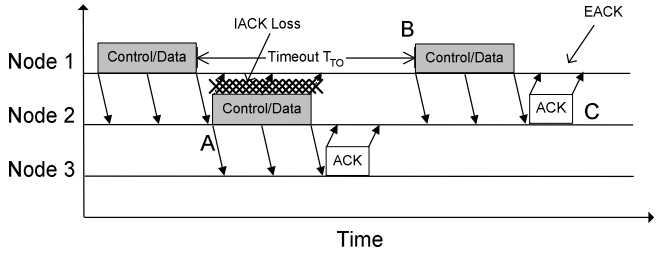

Figure 41 - Use Interchange of IACK and EACK in face of packet loss.

Both IACK/EACK modes can be explicitly set by toggling a particular control bit in the C/D packet header by the sender, and the receiver of the packet will respond accordingly. However, the decision to use either ACK mode also depends on routing path topology and application requirements, hence involving yet another kind of cross-layer considerations. In fact, instances where EACK must be used is when the C/D packet reaches the end of a routing path like the previous example, when the C/D packet has already been forwarded to the next hop, and when the packet is traveling through a junction point on the SS-Tree in the upstream direction where the parent needs to wait and perform data aggregation on incoming packets collected from different branches. Figure 41 describes the second case on the same topology as in Figure 37(a). Initially, Node 1 did not receive the initial IACK and retransmits after a timeout. Since the C/D packet has already been successfully passed from Node 2 to Node 3, Node 2 responds to the retransmitted C/D packet by an EACK back to Node 1.

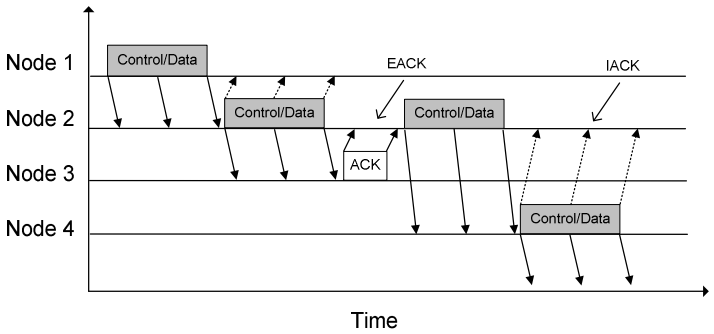

Figure 42 - Timing diagram of mixed IACK and EACK use at SS-Tree junction.

To simplify the process of message passing over SS-Tree junction points, the upstream sender will treat the wireless multicast situation as multiple unicast links. Figure 42 shows its operation on the same topology as in Figure 39(a), but only this time Node 3 replies Node 2 with an EACK because it is made to have no downstream descendents to forward to. On the other hand, Node 4 acknowledges the C/D packet with an IACK back to Node 2.

With mixed use of IACKs and EACKs, the estimated *RTT* on a linear path of N_{hop} hops with IACKs, RTT', becomes:

$$RTT' = 2\left(N_{hop} - 1\right)\left(\frac{2T_{CD} + T_{TO}}{p} - T_{TO} - T_{CD}\right) + 2\left(T_{CD} + T_{ACK}\right). \quad (48)$$

To prevent the disruptive effects a corrupted IACK can cause in multihop communications, a new timeout value, T_{TO}', is defined to be:

$$T_{TO}' \geq 2T_{CD} \ . \quad (49)$$

Comparing Equations (47) and (49) shows the IACK mechanism would work better in reducing the time dwelled in push-pull traffic sequencing when the size of C/D packets is comparable to that of EACKs, which can be achieved through data reduction and data aggregation schemes mentioned earlier.

Since the traffic flow is light and coordinated during push-pull traffic sequencing, the likelihood of collision between IACKs and other packets is low. Still, with the introduction of spontaneous event-driven data to the push-pull traffic sequencing stream come increased chances of packet collisions and timing overshoots that exceed the T_{PP} period. To reduce the effects of event-driven data and random channel errors that can invalid-ate the IACKs, the original sender needs to only decode the first few bytes or just the header portion of the IACK and accept as legitimate packet acknowledgement when C/D packets are long. Nevertheless, both factors will affect the determination of the packet delivery success rate *p*. The exact derivation of *p* will also depend

on the underlying wireless channel model assumptions, and this will be delegated as part of future work.

3.4.4 Failure Recovery

Nodal failures can be a result of drained battery, failed transceiver, adverse radio propagation conditions, or a node that failed to wake up according to its sleep schedule. For example in Figure 43(a), when the node coloured in white fails, a stub network consisting all of the downstream nodes forms which is disconnected from the data sink. During the active period, the nodal failure can be detected if the immediate upstream node (i.e., Node A) cannot pass on the push phase token to the white node. As a result, Node A will instantly send a *Node Unreachable* urgent notification to the data sink indicating the location of the failure. On the other hand, the nodes on the stub network are still functioning properly and are expecting the arrival of the downstream token. If the nodes do not receive the token by the end of the active period, they will know that some upstream blockage have prevented the toke from passing through. To reconnect to the data sink, they need to transmit an *I'm Alive* urgent notification to the data sink in the next available active period on the adjacent SS-Tree to signal their well-being. After assessing the scope of the nodal failure, the data sink will try to restructure the nodes in the vicinity of the failed node to build a new route around the failure. In Figure 43(b), Node B is adjacent to the current SS-Tree but originally belongs to another SS-Tree with a different sleep schedule. Then it receives a sleep scheduling update message from the data sink which orders it to follow an addition set of sleep schedule so that it can connect to Node A and the stub network in the same active period. Obviously this temporary failure recovery solution will cause Node B's energy reserves deplete much quicker since it becomes a shared node between 2 SS-Trees, so other more permanent approaches such as deploying replacement nodes should be considered.

For some reason a node may become unsynchronized with the main sleep schedule and lose touch with the data sink, as indicated by a consecutive number of unaddressed *I'm Alive* urgent notifications. To reattach itself back onto the WSN, the node should reset to become a new node and begin transmitting *Hello* messages in short intervals for the neighbouring nodes to capture during their active periods. In the worst case, all the neighbours of a node have failed, thus creating an island shown in Figure 44 that the node cannot breach the isolated confines. On the

other hand, for the nodes surrounding the island that are still connected to the data sink, the current messaging scheme may not be sufficient in recovering from such large types of nodal failures. Other large-scale failure recovery approaches such as those described in [34] and [35] should be pursued instead.

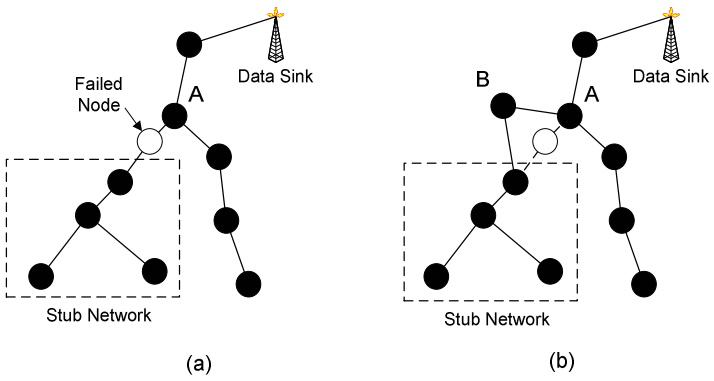

Figure 43 - Failure recovery example.
(a) Failure detection. (b) Routing around failed node.

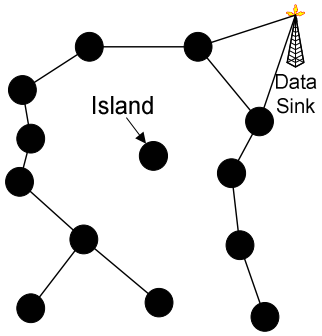

Figure 44 - Island creation.

Since unexpected nodal failures disrupt data reporting traffic and cause packet loss, pre-emptive measures should be enacted to prevent these circumstances. For example, while energy-aware network management may delay the onset of energy depletion of sensor nodes, it cannot extend system lifetime forever. Therefore, when the complete drainage of battery power is imminent, the sensor node should send a *Low Energy* urgent notification to the data sink at the next available active period so that appropriate SS-Tree reconfigurations can be performed. On the other hand, the sensor nodes would periodically conduct self-diagnostic tests to examine the

well-being of their peripheral components. Having both smoke detector and thermometer installed on each sensor node offers added assurance that event monitoring can still continue in case one of the sensory units has failed. Nevertheless, whenever a critical component failure is detected, a *Hardware Failure* urgent notification is to be sent to the data sink for traffic route reconfiguration around the useless node.

Data sink plays a paramount role in connecting the sensing field with the processing centre. Since all of the communication coordination activities are handled by the data sink, it is important to ensure its robustness and survivability since its failure would mean possible prolonged disruption in WSN operations. Therefore, its operating status is constantly under scrutiny of the processing centre to detect any signs of abnormality. If multiple data sinks reside on a sensing field, then they should periodically exchange sensor node link state and sleep scheduling information via satellite or point-to-point link so that in the event of data sink failure, adjacent data sinks can immediately assume responsibility of sensing field coordination without delay. It may also be useful for individual nodes to know the presence of other nearby data sinks for replacing the failed data sink. Since predefining backup recovery paths could be costly, any failure recovery procedures should be computed at real-time. Because of the complexity involved, the concept of data sink failure recovery will not be investigated further in the current design approach.

4 PERFORMANCE EVALUATIONS

The performance evaluations are divided into 5 sections. Section 4.1 will demonstrate graphically how sleep scheduling affects temporal sensing coverage, whose relationship is first described in Section 2.4.3. As the length of the active period of each sleep cycle is associated with how quickly event-driven data reporting can be achieved, Section 4.2 will look into how the proposed IACK scheme in Section 3.4.3 fares compared to traditional EACK scheme in reducing active period length subject to various MAC-level effects. Based on the results obtained in Section 4.1, Section 4.3 will incorporate the notions of transmission range adjustment (see Section 2.5.2) and SS-Trees (see Section 3.1) to evaluate the improvement possible in temporal sensing coverage via the proposed SS-Tree scheme. Section 4.4 further presents a general example and a case study for evaluating energy efficiency with the use of SS-Trees in face of temporal sensing coverage considerations. Finally, Section 4.5 compares the effectiveness of the three SS-Tree computation methods presented in Sections 3.3 in providing adequate sensing coverage and energy efficiency.

4.1 SLEEP SCHEDULING AND TEMPORAL SENSING COVERAGE

This section investigates the relationship between sleep scheduling and temporal sensing coverage, specifically surrounding Equations (11) to (14) in Section 2.4.3. Table 5 summarizes the list of system parameters involved. The first three parameters, namely $\overline{T_{event}}$, $\overline{T_{timer}}$ and $\overline{T_{req}}$, respectively serve as performance benchmarks against parameters T_{active}, ρ and N_{hop}. All timing information will be treated as relative to T_{hop} and T_{sense}, and both of them will assume the value of 1 in the test cases of this section. MATLAB is used to solve the various equations to obtain the results in this section. Figure 45 and Figure 46 plot $\overline{T_{event}}$ vs. T_{active} against different values of ρ and N_{hop}, respectively, whose relationship is defined by Equation (11) earlier in Section 2.4.3. As shown in Figure 45, event-driven data reporting performs poorly with respect to timing for low values of ρ as expected, where in general $\overline{T_{event}}$ is inversely proportional to ρ. Notice that for all any value of ρ, a low T_{active} to T_{hop} ratio (i.e., a packet can only traverse a few hops per active period) produces a much longer $\overline{T_{event}}$ than a ratio that permits more hop traversals within T_{active}. It implies that the popular MAC strategy of shutting off the transceiver right after transmitting 1

packet may not fare well with ultra-low duty cycle sleep scheduling in this regard. However, as this ratio increases further, the improvement in $\overline{T_{event}}$ eventually reaches its maximum, which is approximately at $T_{active} = \frac{T_{hop} N_{hop}}{2}$ as indicated in Figure 46, before increasing linearly in proportion to T_{active}.

Symbol	Description
$\overline{T_{event}}$	Expected event-driven data reporting delay
$\overline{T_{timer}}$	Expected timer-driven data reporting delay
$\overline{T_{req}}$	Expected request-driven data reporting delay
T_{hop}	Per hop delivery time
T_{semse}	Data sensing and processing time
T_{active}	Length of active period in each sleep cycle
ρ	Duty cycle
N_{hop}	Hop count
N_{sst}	Number of SS-Trees

Table 5 - Summary of system parameters involved in analyzing the relationship of sleep scheduling and temporal sensing coverage with SS-Trees.

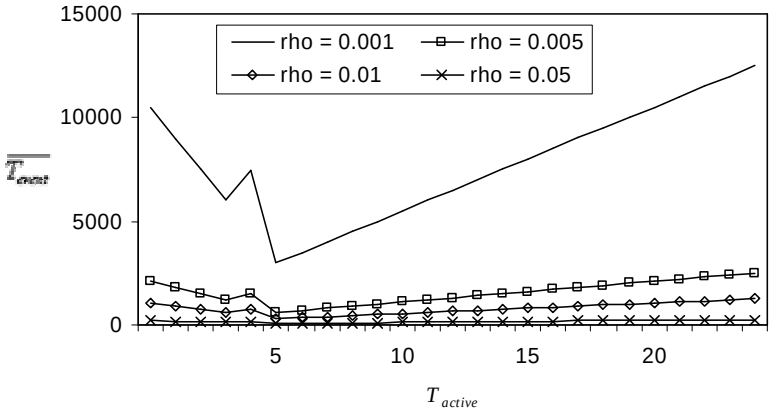

Figure 45 - $\overline{T_{event}}$ vs. T_{active} for various values of ρ(rho) at N_{hop} = 10, T_{hop} = 1 and T_{sense} = 1.

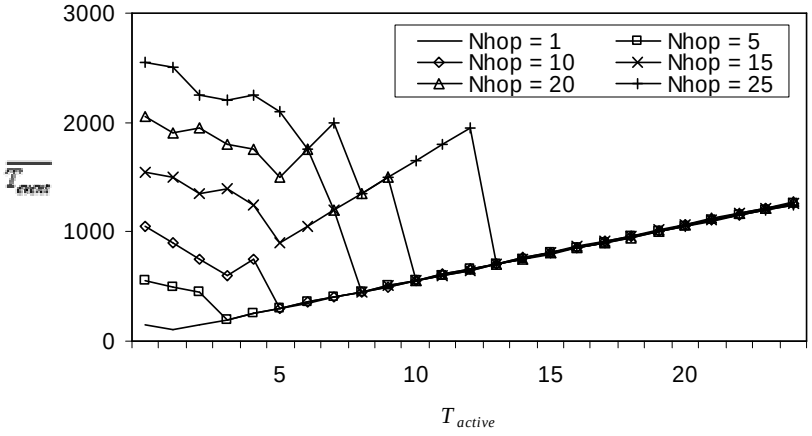

Figure 46 - $\overline{T_{event}}$ vs. T_{active} for various values of N_{hop}(Nhop) at ρ = 0.01, T_{hop} = 1 and T_{sense} = 1.

The overall saw-tooth profile produced by smaller values of T_{active} in Figure 46 is due to the combined effects of data reporting packet being able to complete more hops in a single active period and increase in duration of each sleep cycle as T_{active} gets higher. Interestingly, although lowering N_{hop} generally leads to a proportional decrease in $\overline{T_{event}}$, the absolute timing improvement of shorter hop counts is significantly reduced when T_{active} is set such that the packet to be delivered end-to-end in one active period, which leaves the duration of the sleep action as the sole major factor contributing to event-driven data reporting latency.

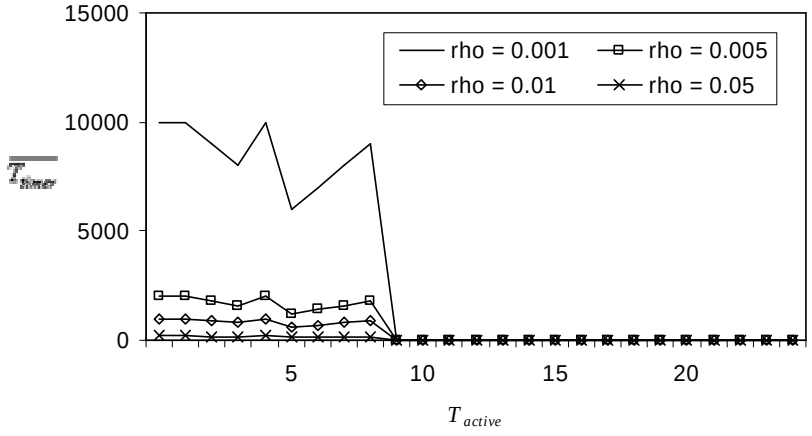

Figure 47 - $\overline{T_{timer}}$ vs. T_{active} for various values of ρ(rho) at N_{hop} = 10, T_{hop} = 1 and T_{sense} = 1.

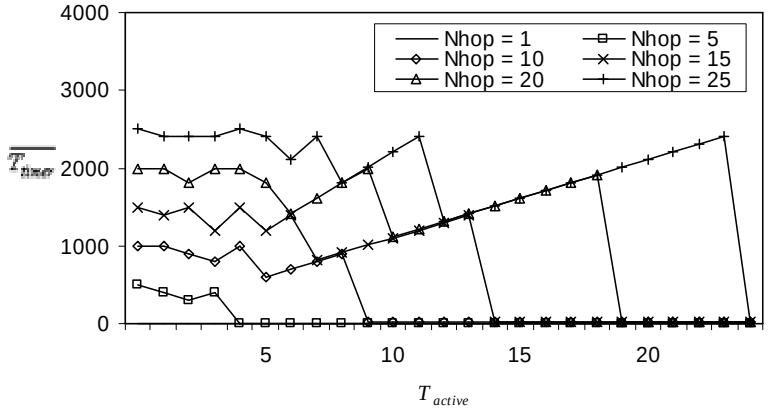

Figure 48 - $\overline{T_{timer}}$ vs. T_{active} for various values of N_{hop}(Nhop) at ρ = 0.01, T_{hop} = 1 and T_{sense} = 1.

Figure 47 and Figure 48 graphically illustrate Equation (12), which relates $\overline{T_{timer}}$ and T_{active} for different values of ρ and N_{hop}, respectively. Because of the similarities of the event-driven and timer-driven models, their results very much resemble each other, though at high T_{active} values the effects of sleep cycle duration do not play any role in determining $\overline{T_{timer}}$. From both figures, it is obvious that the end-to-end completion time for timer-driven data reporting is very small if a high T_{active} allows the packets to travel end-to-end in a single active period. However, high T_{active} values would increase the minimum timing separation between consecutive timer-driven data reports, especially for low ρ values as shown in Figure 49 (see Equation (13)).

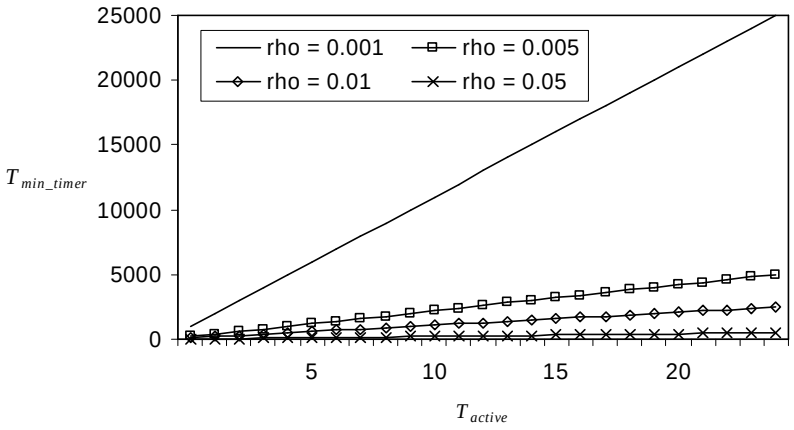

Figure 49 - T_{min_timer} vs. T_{active} for various values of ρ(rho).

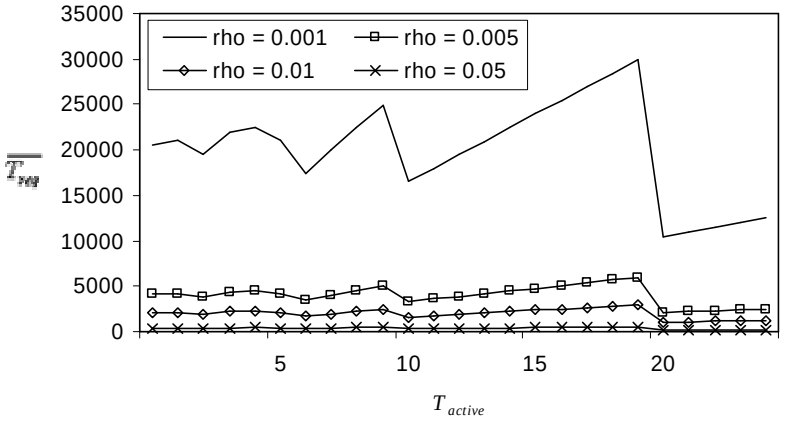

Figure 50 - $\overline{T_{req}}$ vs. T_{active} for various values of ρ(rho) at N_{hop} = 10, T_{hop} = 1 and T_{sense} = 1.

The results for $\overline{T_{req}}$ vs. T_{active} against different values of ρ and N_{hop} are shown in Figure 50 and Figure 51, respectively, which in turn are given by Equation (14). Compared with the other two types of data reporting, request-driven data reporting requires at least twice the amount of time to complete, which can be exacerbated with a lower duty cycle and longer end-to-end hop count. As with the other two types, a higher T_{active} value that allows end-to-end data requests to be completed in a single active period produces a smaller $\overline{T_{req}}$. However, as T_{active} increases lower N_{hop} values retain their timing advantage of $\overline{T_{req}}$ better in request-driven data reporting than the other types.

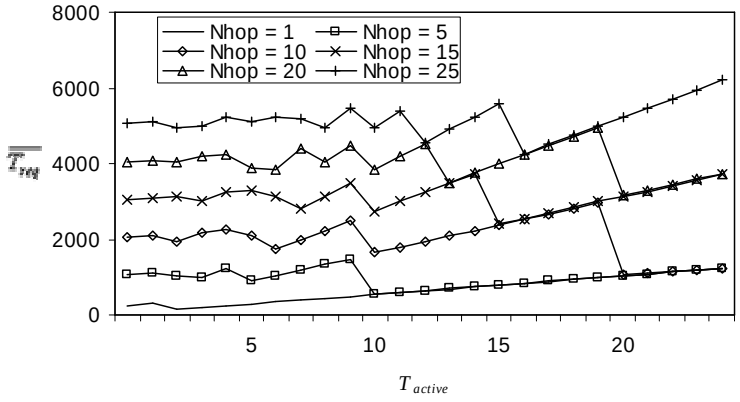

Figure 51 - $\overline{T_{req}}$ vs. T_{active} for various values of N_{hop}(Nhop) at ρ = 0.01, T_{hop} = 1 and T_{sense} = 1.

4.2 MAC-LEVEL EFFECTS

The performance evaluation for sleep scheduling with respect to MAC dynamics is conducted using a custom-built discrete event simulator written in ANSI C, whose implementation details are presented later in Appendix C. By forgoing existing wireless network simulator choices (e.g. ns2, Glomosim, Omnet++, Qualnet, etc.), more flexibility can be achieved in manipulating traffic generation and MAC-layer signalling to better portray the cross-layer SS-Tree approach. Comparison is to be made between the normal MAC acknowledgement procedures without RTS/CTS and the proposed combined IACK/EACK approach described in Section 3.4.3 for minimizing the time used in end-to-end push-pull traffic sequencing under different WSN operating conditions. Table 6 shows the parameters to be used in the performance evaluation, whose definitions were given in Section 3.4.3.

Parameter	Range
N_{hop}	2 to 10
T_{CD}	1, 2 and 5
T_{ACK}	1
T_{TO}/T_{TO}'	See Eqations. (47) and (49)
p	0.9, 0.95 and 0.99

Table 6 - Parameters for sleep scheduling performance evaluation.

For simplification, the WSN topology used for performance evaluation is assumed to be a linear path with no junction points, where the hop count is determined by parameter N_{hop} in each test case. RTT measurements are to be taken at the data sink, which is at one end of the linear path, for the amount of time to execute pull-pull traffic sequencing from end to end. Each test case is run 10 times in Monte Carlo style and final results are taken as the average of all recorded data. Simulation data are in turn compared with analytical results obtained from computing Equations (46) and (48). Given the various transceiver choices and operational requirements in WSN design, generalized time units are used in time measurements for better independence from actual data rate and packet size. For example, time to send a C/D packet, T_{CD}, is given as 1, 2, or 5 time units, which can be easily converted to metric units for a given modulation scheme and

packet length. Timeout values for EACK and IACK schemes, T_{TO} and T_{TO}', are equal to the values given in Equations (47)and (49), respectively.

4.2.1 Packet Loss Effects

The following test cases demonstrate the effects in varying p while T_{CD} is fixed at 2. The selected values of p of 0.99, 0.95 and 0.9 are adequate for simulating wireless channel conditions with bit error rates from 10^{-3} to 10^{-6} given that very short packets are passed within the WSN. Figure 52 and Figure 53 show how different p values affect the eventual RTT measurements in EACK only and IACK/EACK schemes, respectively, where the dotted and solid lines denote analytical and simulation results, respectively. Simulation data figures adhere well to the linear profiles of the analytical results in all test cases with less than 3% difference except for the EACK only scheme at p = 0.9, where the deviation is over 10%. A closer inspection on the actual packet exchange log revealed that more frequent instances of corrupt ACK packet at p = 0.9 created havoc in the MAC signalling as noted in Figure 38, thus leading to timing prolongation and uncertainties in the eventual RTT measurements.

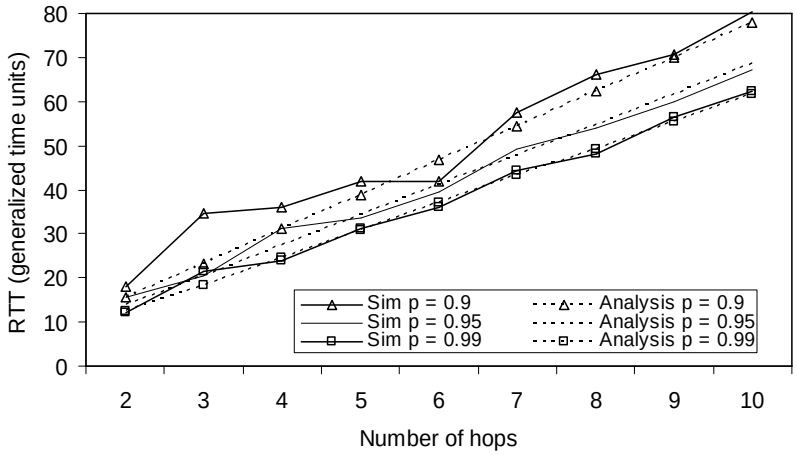

Figure 52 - Packet loss effects in EACK only scheme.

In both schemes, RTT measurements increase along with packet losses because of the extra time spent recovering from packet losses. However when compared to the EACK only scheme, the IACK/EACK scheme achieved an over 25% reduction in the time required for running push-pull traffic sequencing for all p values, thereby increasing monitoring sensitivity through scheduling

shorter and more frequent active periods for each SS-Tree. Conversely, the amount of time used in the active period for the EACK scheme can remain the same, but the extra 25% of time can be designated as the T_{G2} portion of the active period as depicted in Figure 35 and Equation (41) for better protection against all the abnormalities and timing overshoots during push-pull traffic sequencing, as well as increasing the available time to safely deliver event-driven data to the data sink.

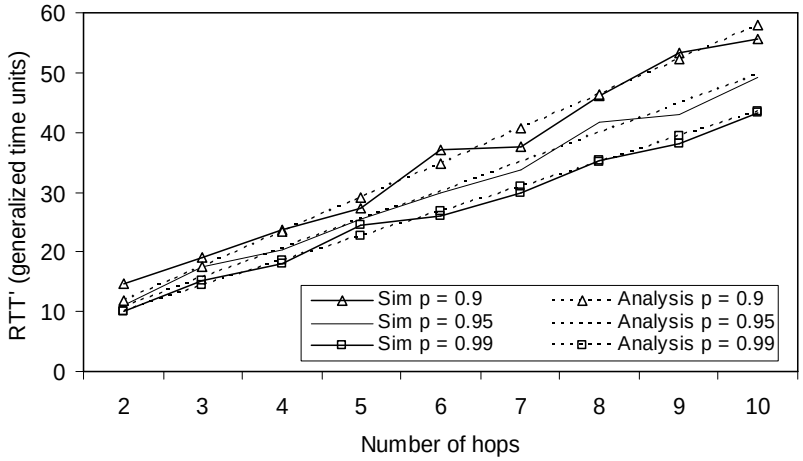

Figure 53 - Packet loss effects in IACK/EACK scheme.

4.2.2 Packet Length Variations

Previously, a 2:1 relationship is assumed in the time used for processing and delivering ACK and C/D packets. Therefore it would be helpful to investigate the effects on RTT measurements with respect to changes to this ratio. The following test cases study the effects in changing T_{CD} (i.e., the length of C/D packets) with p fixed at 0.95 and T_{ACK} set at 1, and the results are shown in Figure 54 and Figure 55 for both acknowledgement schemes, respectively. T_{CD} takes on values of 1, 2 and 5, and its relationship to T_{ACK} will determine how much control information and data can be incorporated into C/D packets without affecting active period scheduling and temporal sensing coverage.

From both Figure 54 and Figure 55, it is apparent that RTT measurements increase at a faster rate as T_{CD} becomes greater. The reasoning is that since the timeout value T_{TO} increases along with T_{CD}, any increase of T_{CD} will put double pressure to increase the RTT values. On the other hand, increasing T_{CD} will diminish the performance advantage achieved by the IACK/EACK

scheme as its main feature of reduction in explicit ACK use become a lesser factor in influencing RTT values when C/D packets get longer. Specifically, the timing reduction of using the IACK/EACK scheme is nearly 40% over the EACK only approach when T_{CD} = 1, whereas is the same performance metric drops to about 25% for T_{CD} = 2 and then further to only about 12% for T_{CD} = 5. Therefore WSN designers using SS-Trees need to keep in mind of the ramifications of trading off C/D packet size for sleep scheduling and monitoring sensitivity.

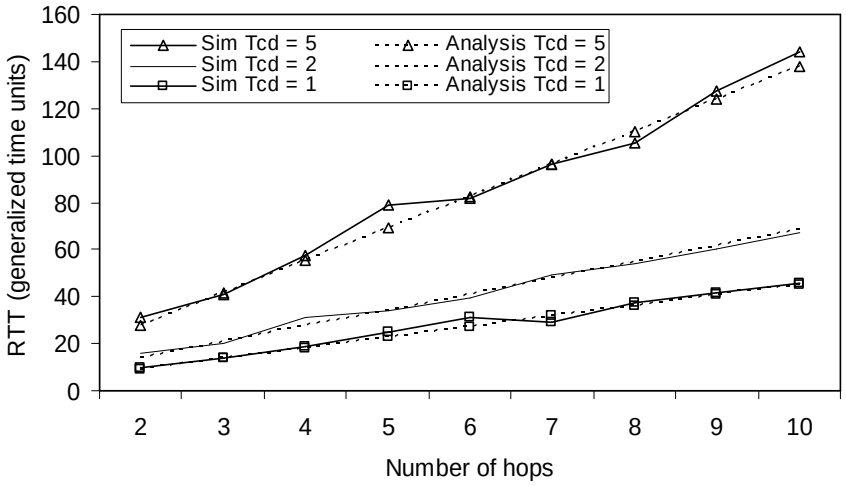

Figure 54 - Effects of packet length variations in EACK only scheme.

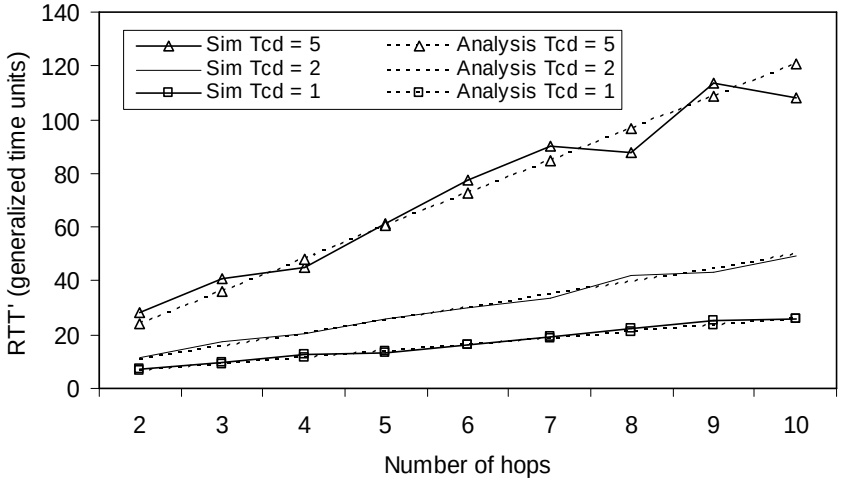

Figure 55 - Effects of packet length variations in IACK/EACK scheme.

4.3 TRANSMISSION RANGE, TEMPORAL SENSING COVERAGE AND SS-TREES

In all three data reporting types, a smaller hop count generally helps in reducing end-to-end delay, albeit to various degrees when coupled with the effects of ρ and T_{active}. As hop count is related to transmission range assignment and node deployment density, the interaction of these three aspects remains to be investigated. Also with the introduction of the SS-Tree concept in Chapter 3, it would be interesting to examine the timing performance of event-driven data reporting under different network configuration settings. Table 7 lists the main parameters involved in this analysis, where the values for λ, R_{sink} and R_{com} will all be taken as relative. Note that the values for R_{sink} are restricted by the dimensions of the sensing field (i.e., L x W), though the exact area attributes will not be specified in the test cases. The various formulations for evaluating $\overline{N_{hop}}$ and $\overline{T_{event}}$ with respect to SS-Trees are solved using MATLAB, where the former results are computed with the help of numerical integration techniques.

Symbol	Description
N_{sst}	Number of SS-Trees
$\overline{T_{event}}$	Expected event-driven data reporting delay
$\overline{N_{hop}}$	Expected hop count
N_{hop}	Hop count
λ	Sensor node deployment density
L x W	Sensing field dimensions
R_{sink}	Physical distance between node and data sink
R_{com}	Communication range

Table 7 - Summary of system parameters involved in analyzing the relationship between event-driven data reporting, transmission range assignment and SS-Trees.

First, the relationship between node deployment density and expected hop count with respect to the sink-to-node physical distance and communication range is examined. Results shown by Figure 56 suggest that in order to reduce $\overline{N_{hop}}$ by half, which in turn can lead to a potential 50% timing improvement for all types of data reporting, node deployment density has to be increased by at least 3 times if R_{com} is to remain the same. On the other hand, achieving a similar decrease

in $\overline{N_{hop}}$ only requires increasing the transmission range by less than twice the original value as shown in Figure 57. So this demonstrates that improving temporal sensing coverage involves trading off deployment cost or per node energy use, and the ultimate choice depends on the impact of both options in the overall WSN design. Further discussions on energy efficiency with respect to transmission range will be provided in the next section.

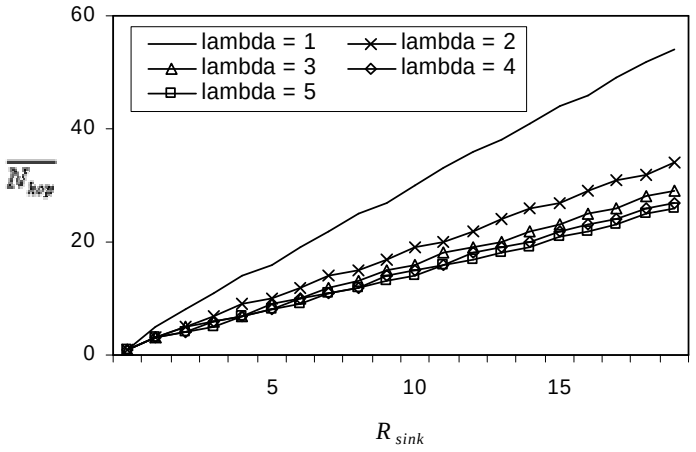

Figure 56 - $\overline{N_{hop}}$ vs. R_{sink} for various values of λ(lambda) at R_{com} = 1.

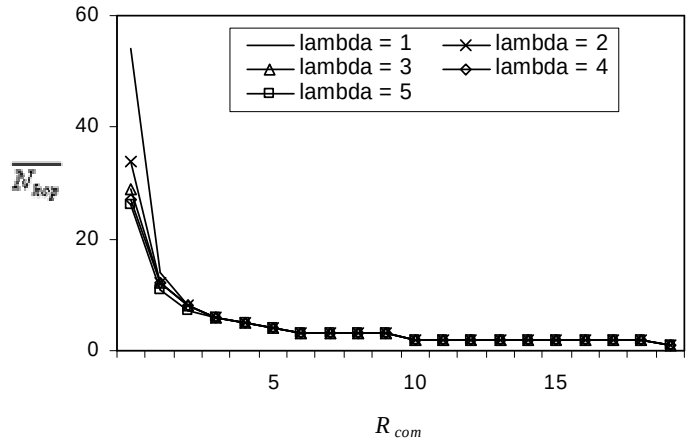

Figure 57 - $\overline{N_{hop}}$ vs. R_{com} for various values of λ(lambda) at R_{sink} = 20.

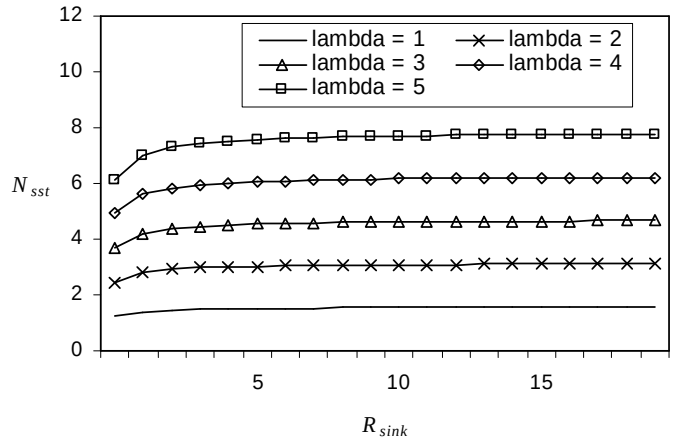

Figure 58 - N_{sst} vs. R_{sink} for various values of λ(lambda) at R_{com} = 1.

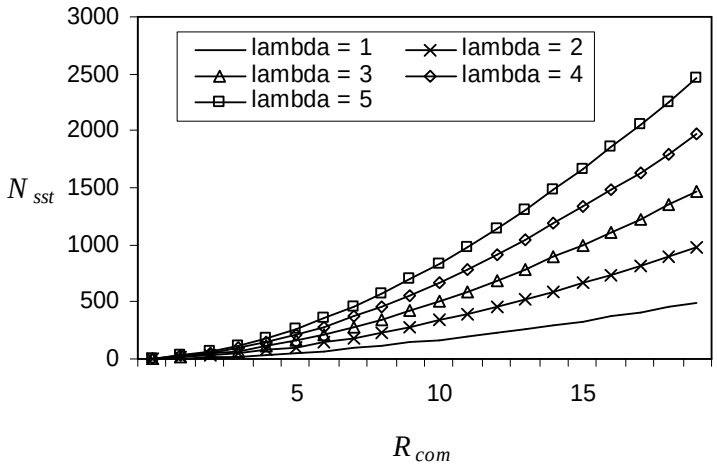

Figure 59 - N_{sst} vs. R_{com} for various values of λ(lambda) at R_{sink} = 20.

If SS-Trees are to be implemented, then increasing node deployment density is projected to allow more SS-Trees to be computed, as shown in Figure 58. However, such increase in N_{sst} pales in comparison with those attained through increasing transmission range, as shown in Figure 59. While extending R_{com} to attain an N_{sst} estimate that reaches several hundred or even several thousand is a bit extreme, numerical results suggest that a slight increase in R_{com} would be enough to produce tens of SS-Trees. Note that as the ratio of R_{sink} to R_{com} gets lower (i.e., R_{sink} becomes smaller in Figure 58), the expected number of achievable SS-Trees decreases as well

since the forwarding region shrinks as the node is located closer to the data sink (see Figure 20), which translates into a smaller of number of upstream neighbours per node. Therefore, nodes can be deployed at a slightly higher density in areas around the data sink in order to offset this effect when implementing SS-Trees.

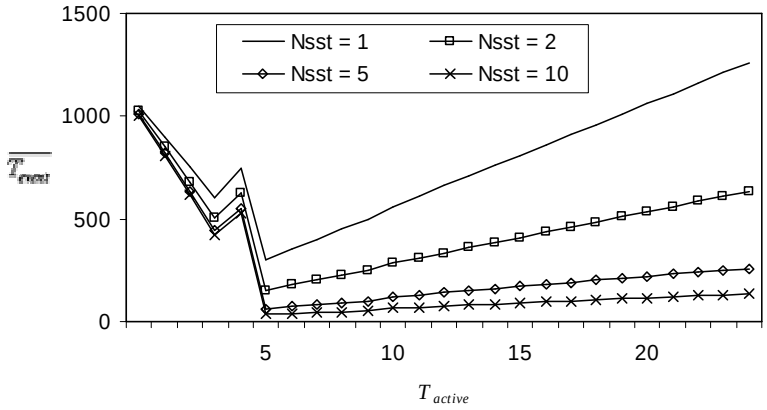

Figure 60 - $\overline{T_{event}}$ vs. T_{active} for various values of N_{sst} at ρ = 0.01, N_{hop} = 10, T_{hop} = 1 and T_{sense} =1.

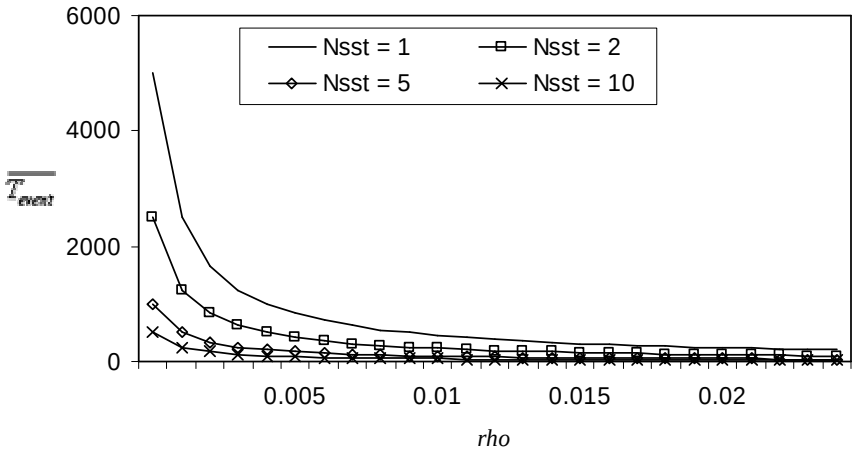

Figure 61 - $\overline{T_{event}}$ vs. ρ for various values of N_{sst} at T_{active} = 10, N_{hop} = 10, T_{hop} = 1 and T_{sense} = 1.

Increasing the number of computable SS-Trees will in turn improve the timing performance of event-driven data reporting, which is essential in the majority of WSN applications. Figure 60

examines the effects of implementing more SS-Trees on $\overline{T_{event}}$ with respect to T_{active}. It is shown that the timing improvement in $\overline{T_{event}}$ as offered by having more SS-Trees becomes more pronounced when T_{active} is high. Also, the linear increasing profile of $\overline{T_{event}}$ after it reaches its minimum becomes more and more subtle as N_{sst} increases. Figure 61 looks at how $\overline{T_{event}}$ is affected by both changes in N_{sst} and ρ and discovers that for a given ρ, a high N_{sst} count dramatically decreases $\overline{T_{event}}$. In other words, by having more SS-Trees, the operational duty cycle required for guaranteeing a certain $\overline{T_{event}}$ threshold as required by the sensing application is effectively reduced, thereby significantly improving energy efficiency.

4.4 Energy Efficiency and Temporal Sensing Coverage

Sections 4.1 and 4.3 have discussed in detail how temporal sensing coverage is affected by sleep scheduling, transmission range assignment, and the SS-Tree concept. This section will tie the other major consideration in WSN design, which is energy efficiency, into the overall discussion. Table 8 contains the many parameters that will figure into the calculations. Again, the numerical results for this section are computed using MATLAB. Note that node lifetime (i.e., $\overline{T_{lifetime}}$) in this section is defined to be the time it takes to exhaust the battery supply of each node with no consideration on network connectivity or incidences of shared nodes in SS-Trees.

Symbol	Description
$\overline{T_{lifetime}}$	Expected node lifetime
$\overline{T_{event}}$	Expected event-driven data reporting delay
$\overline{T_{timer}}$	Expected timer-driven data reporting delay
$\overline{T_{req}}$	Expected request-driven data reporting delay
L x W	Sensing field dimensions
$E_{battery}$	Initial battery supply
R_{com}	Communication range
T_{active}	Length of active period in each sleep cycle

ρ	Duty cycle
D_{Tx}	Expected amount of data transmitted during an active period
D_{Rx}	Expected amount of data received during an active period
B_{Tx}	Average data rate for data transmission
B_{Rx}	Average data rate for data reception
P_{Tx}	Power consumed by transceiver during data transmission
P_{Rx}	Power consumed by transceiver during data reception
P_{com_idle}	Power consumed by transceiver during idle listening
P_{com_sleep}	Power consumed by transceiver during sleep state
P_{proc_active}	Power consumed by processor during active state
P_{proc_sleep}	Power consumed by processor during sleep state

Table 8 - Summary of system parameters involved in analyzing energy efficiency with temporal sensing coverage, sleep scheduling and transmission range assignment.

4.4.1 A General Example

This section presents a general example to demonstrate the combined effects of sleep scheduling, transmission range assignment and SS-Trees on energy efficiency and temporal sensing coverage using the simple RF transceiver model presented in Section 2.2.2. A list of realistic parameters used in the evaluation is shown in Table 9, in which values pertinent to the calculation of P_{Tx} are drawn from [74], and those for processor operation are taken from Table 11 in the next section.

Parameter	**Values**
L x W	500 m x 500 m
λ	1 per 100 m^2
$E_{battery}$	2000 mAh x 3.3 V
D_{Tx} , D_{Rx}	10 bytes
T_{sense}	100 µs
T_{hop}	5 bytes/B_{tx}
NF_{Rx}	11 dB (12.589)
$(S/N)_{Rx}$	10 dB (10)
N_o	-173.8 dBm/Hz (4.17 x 10^{-21} J)

w	0.328 m (for 915 MHz), 0.125 m (for 2.4 GHz)
$G_{Tx}G_{Rx}$	-20 dBi (0.01)
η	0.2
α	2
BW	1 bit/Hz x B_{Tx}
P_E	3.63 mW
$P_{Rx,}P_{com\_idle}$	11.13 mW
P_{sleep}	1 µA
P_{proc_active}	8 mA
P_{proc_sleep}	15 µA

Table 9 - Additional parameters for numerical analysis in the general example.

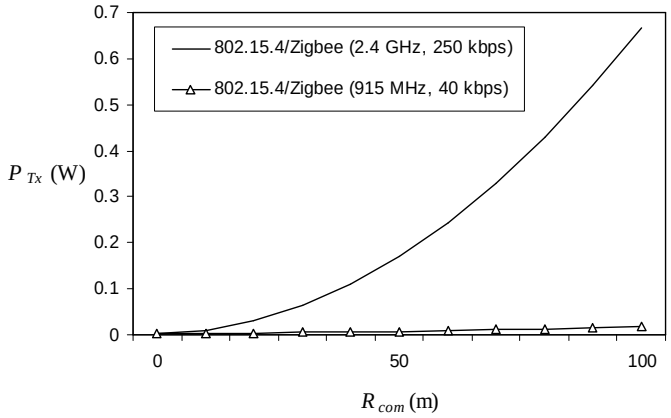

Figure 62 - P_{Tx} vs. R_{com} for different transceiver operation modes.

Initially, each node is supplied with 2 AA batteries which contain about 2000 mAh x 3.3 V of energy, and energy used by the other components besides the processor and transceiver is ignored. Delivering each data reporting packet over 1 hop is assumed to involve the exchange of 5 bytes in total, including messaging overhead and retransmissions. On the other hand, for each node it is assumed about 10 bytes of data are to be transmitted and received, respectively, during each active period. The RF transceiver model used in this general example is based on the IEEE

802.15.4/Zigbee standard, which allows 2 modes of operation where one transmits data at 40 kbps over the 915 MHz band and the other with data rate of 250 kbps over the 2.4 GHz ISM band. Figure 62 projects the changes in power consumption of the transceiver in transmit mode, P_{Tx}, with respect to transmission distance, R_{com}, for both types of operating mode, and it clearly shows while there exists an exponential relationship between P_{Tx}, and R_{com}, the power output of the power amplifier, P_{PA}, overshadows P_E, which is the amount of power collectively consumed by the other electronic components, when both data rate and carrier frequency are high. This leads to the dramatic power consumption profile of the 2.4 GHz transceiver in this figure.

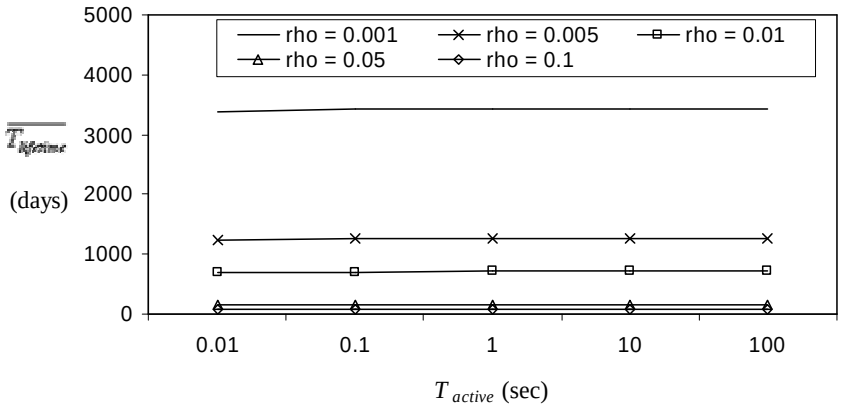

Figure 63 - $\overline{T_{lifetime}}$ vs. T_{active} for different values of ρ(rho) at f = 915 MHz and R_{com} = 100 m.

Figure 63 and Figure 64 compare the effects of adjusting T_{active} and ρ on $\overline{T_{lifetime}}$ for both RF transceiver models, where $\overline{T_{lifetime}}$ is given by Equation (3) in Section 2.2.2. Despite the significant difference in P_{Tx} for both transceiver models at R_{com} = 100 m, their expected lifetime results are almost identical when T_{active} is large because idle listening becomes the dominant factor in energy loss over data transmission. As T_{active} is reduced, however, the percentage of active period occupied by data transmission increases, thereby seriously degrades energy efficiency at f = 2.4 GHz because of its higher P_{Tx}. To maintain a certain level of energy efficiency under short active periods, the duty cycle can be lowered as a trade-off. Note that in order to complete the prescribed data exchanges in each active period, T_{active} cannot be reduced past 4 ms at f = 915 MHz, and 0.64 ms at f = 2.4 GHz. Also, even though the results suggest

that nodal lifetime can be extended over 1000 days with appropriate selection of duty cycle, actual implementation results will most probably fail to live up to this projection because of real-life device nonlinearity and constraints that were simplified or ignored by the transceiver model used in this analysis.

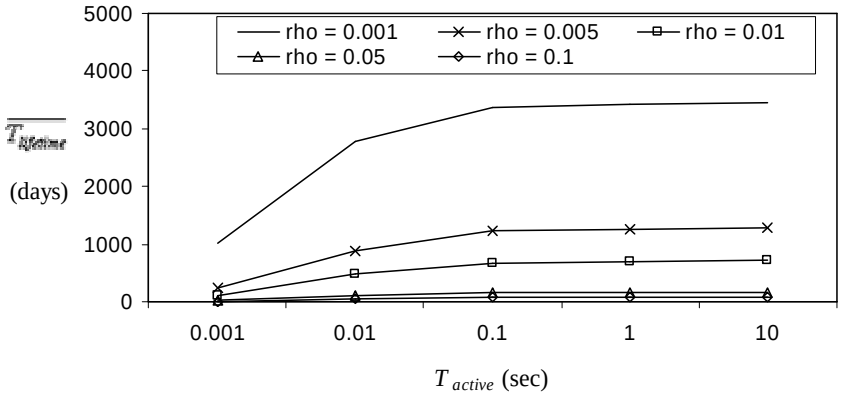

Figure 64 - $\overline{T_{lifetime}}$ vs. T_{active} for different values of ρ(rho) at f = 2.4 GHz and R_{com} = 100 m.

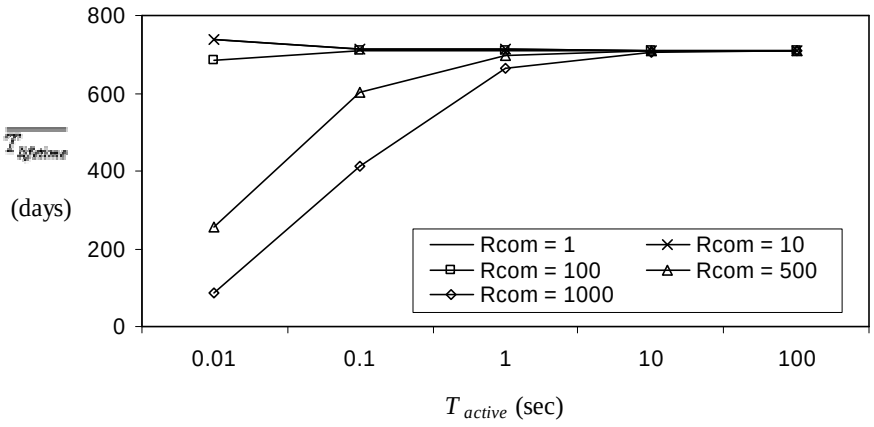

Figure 65 - $\overline{T_{lifetime}}$ vs. T_{active} for different values of R_{com} at f = 915 MHz and ρ = 0.01.

Figure 65 and Figure 66 illustrates the relationship between $\overline{T_{lifetime}}$ and T_{active} when factoring in the effects of transmission range adjustment in both transceiver models, where values of R_{com} are

represented in metres. As T_{active} is shortened, the reduction of expected nodal lifetime in both cases becomes more pronounced when R_{com} is large. Interestingly, a small R_{com} combined with a short T_{active} duration actually yields a better $\overline{T_{lifetime}}$ result because in such cases power lost to idle listening becomes higher than power lost to data transmission due to the short transmission range. Therefore, a short active period would reduce the amount of idle listening and allow less power-consuming data transmissions to dominate, thereby increasing energy efficiency.

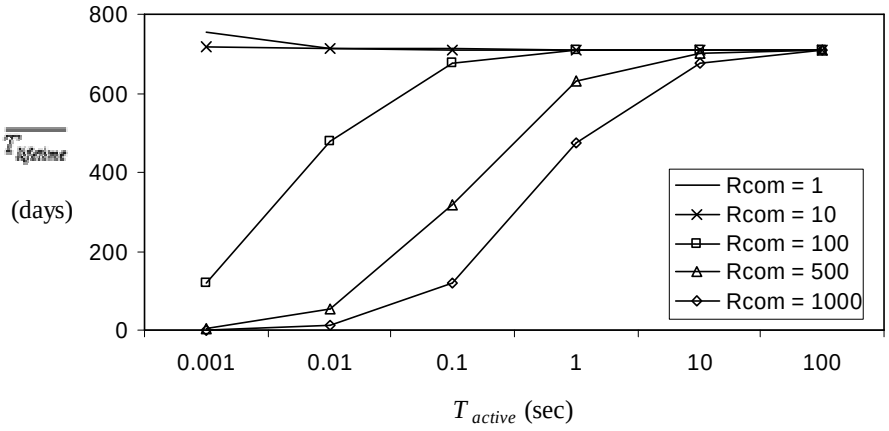

Figure 66 - $\overline{T_{lifetime}}$ vs. T_{active} for different values of R_{com} at f = 2.4 GHz and ρ = 0.01.

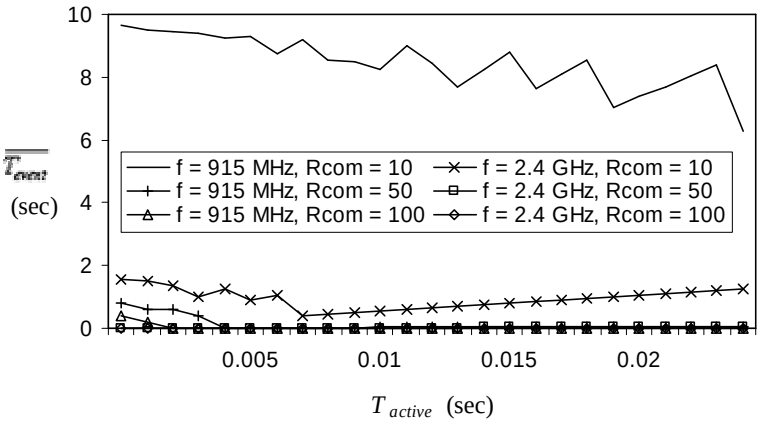

Figure 67 - $\overline{T_{event}}$ vs. T_{active} for both transceiver models at different Tx ranges with ρ = 0.01.

Although setting the transceiver with a short transmission range is the obvious design choice if energy efficiency is the primary concern, its fallout on temporal sensing coverage must not be neglected especially when sleep scheduling under low duty cycles is followed. Given nodes are Poisson distributed over an area of 500 m x 500 m with intensity λ = 1 per 100 m^2, the maximum expected hop count, N_{hop}, is found to be 96, 8 and 4, for transmission ranges of 10 m, 50 m and 100 m, respectively. On the other hand, the expected number of computable SS-Trees, N_{sst}, is projected to be 1, 30 and 123, for transmission ranges of 10 m, 50 m and 100 m, respectively.

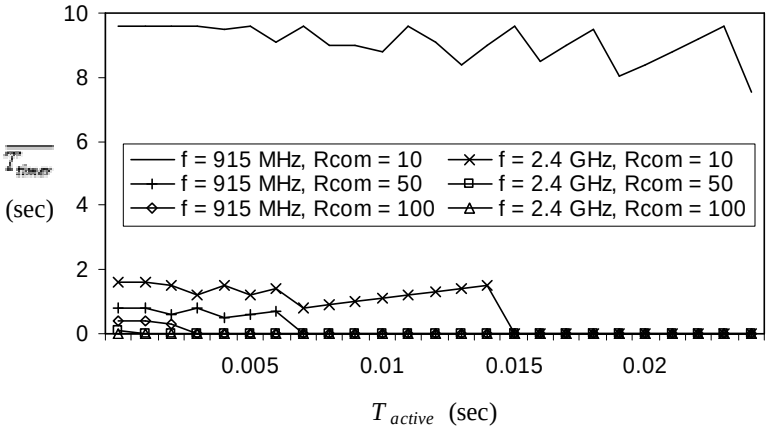

Figure 68 - $\overline{T_{timer}}$ vs. T_{active} for both transceiver models at different Tx ranges with ρ = 0.01.

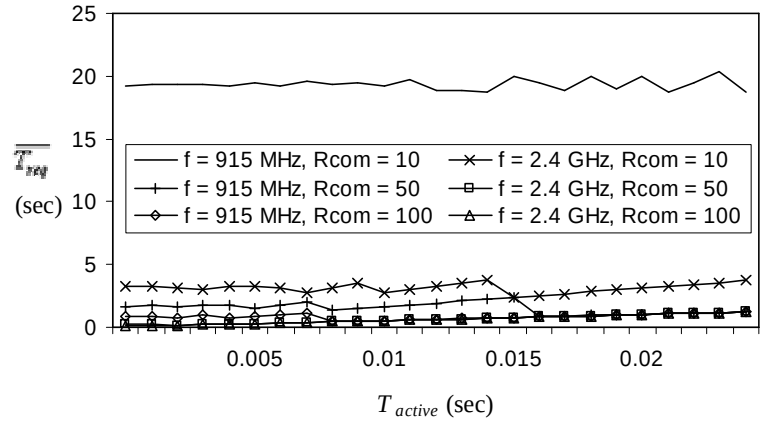

Figure 69 - $\overline{T_{req}}$ vs. T_{active} for both transceiver models at different Tx ranges with ρ = 0.01.

Figure 67, Figure 68 and Figure 69 plot $\overline{T_{event}}$, $\overline{T_{timer}}$ and $\overline{T_{req}}$, respectively, against T_{active} for both transceiver models at different transmission ranges while including the effects of SS-Trees. From the figures, it can be easily seen that high data rates and long transmission ranges provide quicker expected response times for all three types of data reporting. In contrast, shortening the transmission distance increases hop count considerably while limiting the number of available SS-Trees, thereby leading to poor temporal sensing coverage. Note that it is assumed that it takes 1 ms and 0.16 ms to conduct a packet exchange over 1 hop (i.e., T_{hop}) for f = 915 MHz and f = 2.4 GHz, respectively. In comparison, Figure 70 displays the effects on $\overline{T_{event}}$ if SS-Trees are not implemented, and the results suggests that having more SS-Trees will definitely improve timing performance for event-driven data reporting especially when the active period is long, which is in accordance with the observations made in Section 4.3.

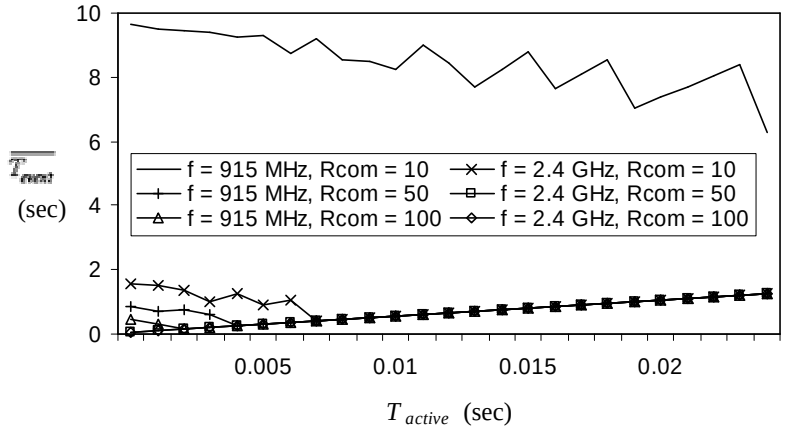

Figure 70 - $\overline{T_{event}}$ vs. T_{active} for both transceiver models without SS-Tree implementation.

4.4.2 A Case Study

As the characteristics of real RF transceivers do not necessarily match the model used in the general example because of the differences in circuit optimization, fabrication techniques, antenna selection, etc., this section will conduct the numerical analysis based on system parameters drawn from data specifications of a selected number of commercially available products as listed in Table 10. Additional parameters used in the numerical analysis of this case study are presented in Table 11. In Table 10, the first two are transceivers, namely XBee-PRO

from Maxstream [60] and Z-Link from Atmel [61], both of which conform to the IEEE 802.15.4/Zigbee standard. In contrast, the Crossbow MICAz [62] is a so-called mote, which is a standalone sensor node with a Zigbee-enabled transceiver installed. For the system lifetime analysis, it is assumed that all three transceivers can be interchanged to work with the same processor on the MICAz for energy consumption measurement purposes. Notice that despite the exponential relationship between transmission power and range, the actual current draw for transmission increases at a much smaller scale compared to transmission range (see comparison between XBee-PRO and MICAz). It is perhaps due to the fact that even for short-range transceivers, a sizable portion of power is consumed for maintaining components such as amplifiers and mixers while only a fraction is transmitted out to the wireless channel. In terms of cost, XBee-PRO is the most expensive option while Z-Link is the most economical. The cost of the transceiver onboard the MICAz module is not given, though it is assumed to be somewhere in between the other 2 transceivers.

Specification	**Maxstream XBee-PRO**	**Atmel Z-Link**	**Crossbow MICAz**
Processor current (active)	-	-	8 mA
Processor current (sleep)	-	-	15 µA
Band	2.4 GHz	915 MHz	2.4 GHz
Data rate	250 kbps	40 kbps	250 kbps
Tx power output	60 mW/18 dBm	16 mW/12 dBm	1 mW/0 dBm
Supply voltage	3.3 V	3.3 V	3.3 V
Tx current	270 mA	60 mA	17.4 mA
Idle/Rx current	55 mA	14.5 mA	19.7 mA
Sleep current	10 µA	1 µA	1 µA
Tx range (indoor/urban)	up to 100 m	up to 30 m	up to 30 m
Tx range (outdoor/LOS)	up to 1.6 km	up to 100 m	up to 100 m
Price	$32 USD/chip	$6.75 USD/chip	$125 USD/unit

Table 10 - Specifications for commercially available transceivers and motes.

The numerical analysis is divided into two cases, outdoor and urban, both of which assume each transceiver is able to transmit at the corresponding range specified in Table 10. In any case, multi-channel selection is disabled and each transceiver can only rely on one radio channel for transmission. Sensing field dimensions and node deployment densities are set different for both cases, though in either case the data sink is assumed to be located in the middle of the sensing field. Otherwise, the rest of the parameters are identical to those used in Section 4.4.1.

Parameter	Outdoor	Urban
L x W	3 km x 3 km	500 m x 500 m
λ	1 per 10000 m^2	1 per 100 m^2
$E_{battery}$	2000 mAh x 3.3 V	2000 mAh x 3.3 V
ρ	0.01	0.01
D_{Tx}	10 bytes	10 bytes
D_{Rx}	10 bytes	10 bytes
T_{sense}	100 µs	100 µs
T_{hop}	5 bytes/B_{tx}	5 bytes/B_{tx}

Table 11 - Additional parameters for numerical analysis in the case study.

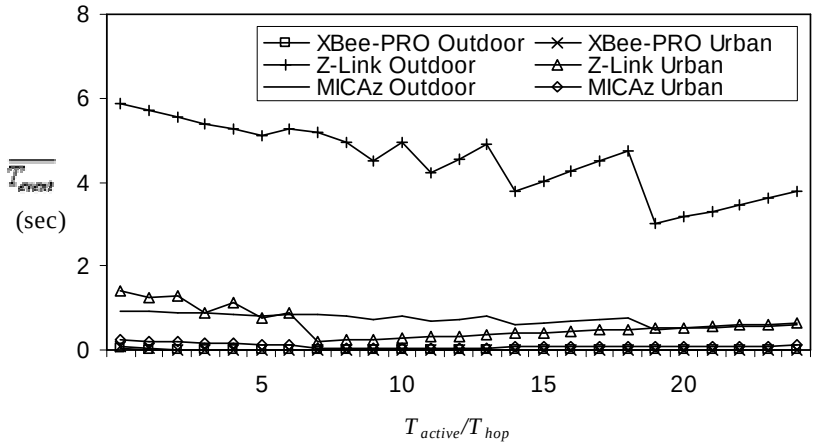

Figure 71 - $\overline{T_{event}}$ vs. T_{active}/T_{hop} for different transceivers at different transmission ranges.

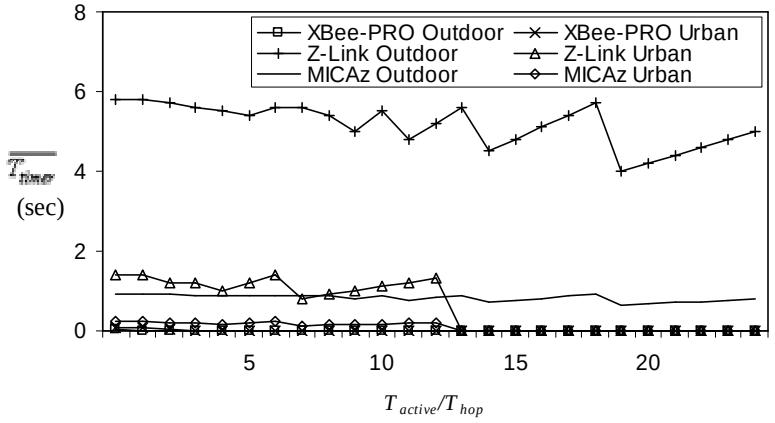

Figure 72 - $\overline{T_{timer}}$ vs. T_{active}/T_{hop} for different transceivers at different transmission ranges.

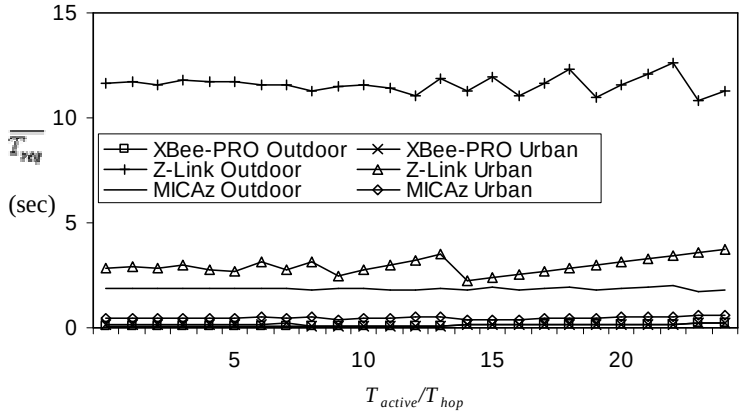

Figure 73 - $\overline{T_{req}}$ vs. T_{active}/T_{hop} for different transceivers at different transmission ranges.

Figure 71, Figure 72 and Figure 73 plot $\overline{T_{event}}$, $\overline{T_{timer}}$ and $\overline{T_{req}}$, respectively, for different transceivers at different transmission ranges while inherently factoring in the effects of SS-Trees. Under the conditions specified in Table 11 regarding sensing area dimensions and node deployment density, the number of SS-Trees computable for shorter range transceivers (i.e., Z-Link and MICAz) is 1 for outdoor and 2 for urban, while it is a remarkable 314 and 24, respectively, for the XBee-PRO. As longer ranges also lead to lower $\overline{N_{hop}}$, it is no wonder that the timing performance for XBee-PRO in all three data reporting types outperforms the other two

transceivers for all T_{active}/T_{hop} ratios in both outdoor and urban environments. On the other hand, because of Z-Link's slower transmission speed, its timing performance mostly registers in the second-level range, which is tens to hundreds of times of that achieved by XBee-PRO and MICAz.

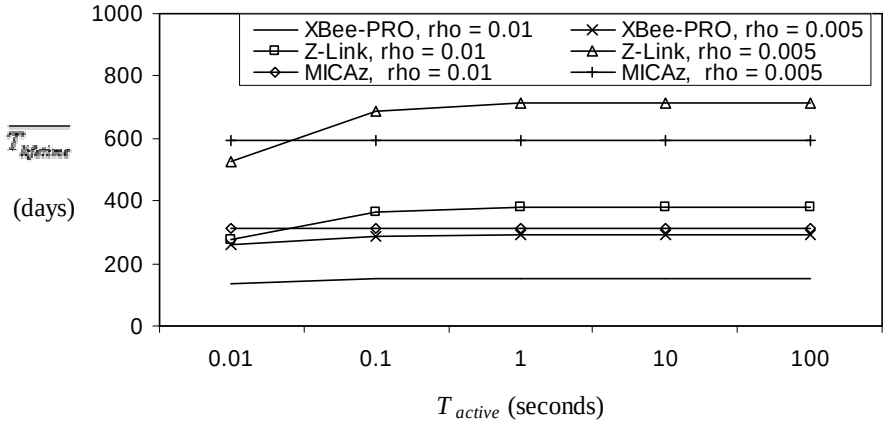

Figure 74 - $\overline{T_{lifetime}}$ vs. T_{active} for the three different transceivers under different duty cycles.

Besides offering excellent timing performance, XBee-PRO also delivers solid energy efficiency results as demonstrated in Figure 74. Although for a given ρ, the expected operational lifetime for Z-Link and MICAz are about 2 times of that of XBee-PRO, the long-range chip can always operate at a lower duty cycle for better energy efficiency with only a minor degradation in data reporting timing performance. As a side note, the tailing off in $\overline{T_{lifetime}}$ as T_{active} decreases for Z-Link and to a lesser extent for XBee-PRO is due to the fact that energy consumption from packet transmission dominates when T_{active} is small. When the difference between transmission power and reception/idle power is as large as in the case for Z-Link and XBee-PRO, power consumption will actually increase as the active period duration gets smaller, thereby reducing expected system lifetime. However, such undesired decrease in $\overline{T_{lifetime}}$ can be more than made up by reducing the duty cycle, which underscores the fact that sleep scheduling is a bigger contributor to energy efficiency than transmission power regulation.

4.5 SS-TREE COMPUTATION

This section evaluates the effectiveness of the proposed SS-Tree computation methods described in Section 3.3 over actual network topologies. The ILP-Dijkstra and ILP-MF formulations for computing SS-Trees are solved using CPLEX 9.0 optimization software, and the iterative algorithmic approach is coded into ANSI C programming language. All three approaches are tested by running simulation cases on a Dell Precision 450 machine with two 2.8 GHz Pentium 4 Xeon processors and 1 GB RAM. The preliminary SS-Tree computation results were performed under the following assumptions:

1. Two types of WSN topology are used: 8-neighbour square grid (*8-N*, see Figure 23(a) in Section 3.1) and 4-neighbour planar square grid (*4-N*, see Figure 27(a) in Section 3.3.2). The number of nodes per grid edge ranges from 3 to 10 (i.e., the number of nodes in the WSN is the square of the number of nodes per grid edge)
2. The number of SS-Trees to be computed on each test topology ranges from 2 to 4 (*2-SST* to *4-SST*).
3. The data sink is represented as a node located at or near the centre of the grid. Specifically, given n is the number of nodes per grid edge, then the coordinates of the data sink is $(0.5n, 0.5n)$ if n is even, $(0.5(n+1), 0.5(n+1))$ otherwise.
4. The link cost between adjacent nodes is 1, which implies that the total shortest path cost from each node to the data sink is simply its corresponding hop count.
5. The maximum number of co-SS-Tree neighbours allowed per node in the ILP-Dijkstra and ILP-MF formulations, $C_{\max}$, is set at 3.
6. The maximum number of nodes allowed per SS-Tree in the ILP-Dijkstra and ILP-MF formulations, $N_{\max}$, is empirically set at $\left\lceil 1.2\frac{|V|}{|K|} \right\rceil$.

4.5.1 ILP-Dijkstra Approach

Figure 75 illustrates the computational complexity involved in solving the ILP-Dijkstra formulation in terms of time used. In the course of the SS-Tree computations, it is evident that CPLEX cannot solve the ILP-Dijkstra formulation quickly when the number of nodes per grid edge increases beyond a certain point. For instance, for test cases with a 8-neighbour grid and 3 SS-Trees (*8-N, 3-SST*), the solution time for a 9x9 grid and 10x10 are so high that there is no

point in including that in the graph. In any case, there exists a close correlation between the density of the WSN topology, the number of SS-Trees involved, and the actual computation time needed. More in-depth analysis into this relationship is needed as part of future work. Note that tests cases with a 8-neighbour grid 2 SS-Trees (*8-N, 2-SST*) are not included in the simulation results because all of them returned with infeasible solutions.

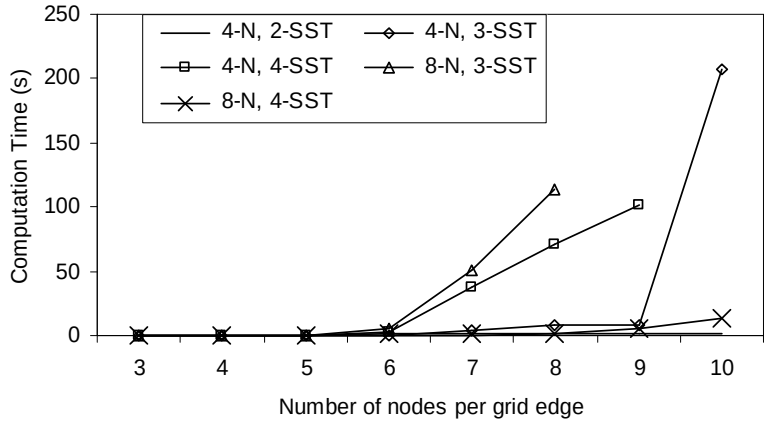

Figure 75 - Amount of computation time used in solving the ILP-Dijkstra Formulation.

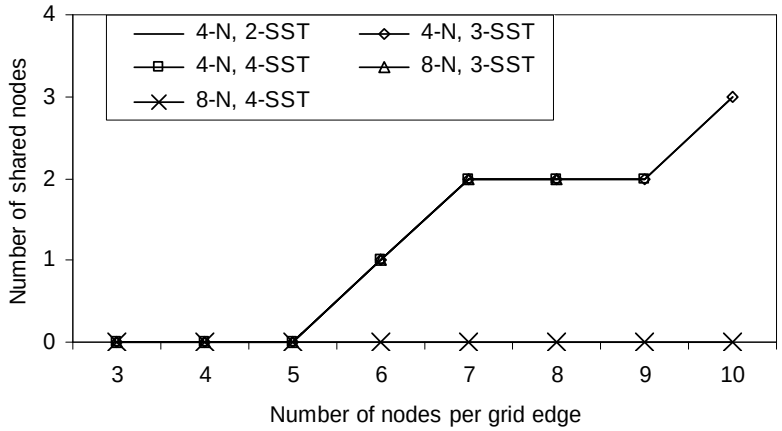

Figure 76 - Number of shared nodes computed using the ILP-Dijkstra approach.

However, long solution times do not necessarily yield the best computation results in terms of minimizing the number of shared nodes. Figure 76 shows the number of shared nodes recorded after a certain number of SS-Trees is computed for a given WSN. Tests cases with a 8-

neighbour grid and 4 SS-Trees (*8-N 4-SST*) successfully eliminated the occurrence of shared nodes, where by aggressive data aggregation and duplicate suppression, the resultant WSNs should be able to offer 2 to 4 times the level monitoring frequency per node at the same communication duty cycle depending how the nodes are assigned to the SS-Trees. On the other hand, other test cases demonstrated interesting consistency in the number of shared nodes computed, but the figures remain very low.

Another measure of how well SS-Trees are computed is the number of *protected* nodes, which refers to nodes with at least 1 non-co-SS-Tree neighbour. This measure is important as it indicates how frequent the event reporting window of each node will be and how well nodes can recover from failures with help from such non-co-SS-Tree neighbours. Figure 77 shows the proportion of nodes protected in each case, and the test cases with 8-neighbour grids all achieved 100% node protection. On the other hand, test cases with sparser 4-neighbour grids and 2 SS-Trees produced the worst protection, though they incurred the fastest computation time. In all test cases, most of the unprotected nodes lie at the fringes of the WSN, and the proportion of protected nodes will be expected to increase as the nodal population gets larger.

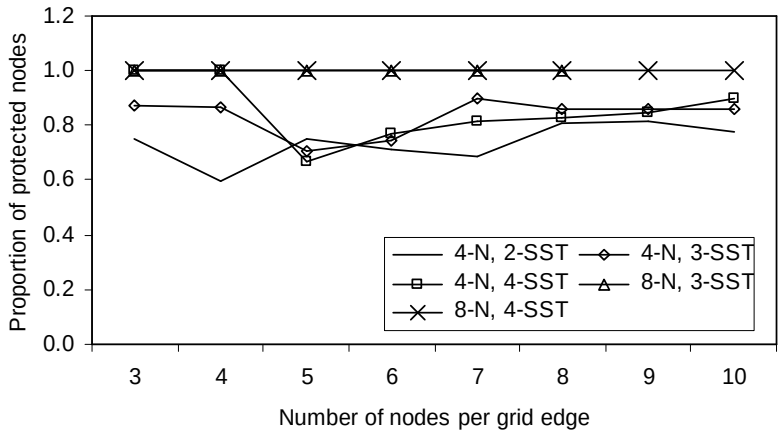

Figure 77 - Proportion of nodes protected using the ILP-Dijkstra approach.

If a protected node is adjacent to neighbours that separately belong to all other SS-Trees, then this node is deemed *fully protected*, which means it possesses the most frequent event reporting

window and it receives the maximum protection from neighbours during failure recovery. Figure 78 shows the proportion of fully protected nodes in each test case. For a given topology, the degree of full protection decreases as the number of SS-Trees to be computed increases. On the other hand for a fixed number of SS-Trees, an increase in network density would also increase the degree of full protection. Therefore, careful design consideration should be given when balancing the optimal nodal deployment strategy, wireless communication range and the number of SS-Trees involved.

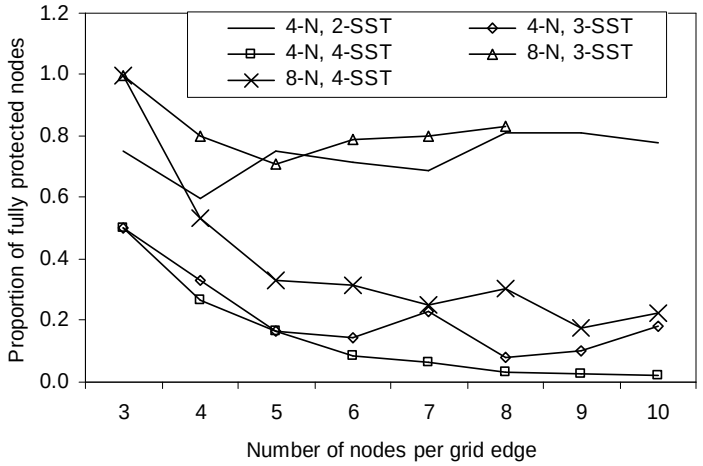

Figure 78 - Proportion of nodes fully protected using the ILP-Dijkstra approach.

4.5.2 ILP-Multicommodity Flow Approach

Throughout the SS-Tree computations, it is clear that CPLEX cannot solve the ILP-MF problem efficiently in all of the test scenarios. Since computation times must be kept within reasonable limits in practical WSN application, a particular test case will be prematurely terminated if it does not return with a solution within 200 seconds. Table 12 lists the best computation times and network size achieved for each test setup before premature stoppage or after the number of nodes per grid becomes 10. Since the number of variables involved in the SS-Tree computation increases along with the number of SS-Trees as shown in Equation (32), it is not surprising to see that computation efficiency degrades when more SS-Trees are to be computed for a given WSN topology. In any case, there exists a close correlation between the density of the WSN topology, the number of SS-Trees involved, and the actual computation time needed.

Test Setup	Best solution time	Nodes per grid edge
4-N, 2-SST	78 seconds	10
4-N, 3-SST	5 seconds	7 (premature)
4-N, 4-SST	93 seconds	6 (premature)
8-N, 2-SST	25 seconds	10
8-N, 3-SST	5 seconds	8 (premature)
8-N, 4-SST	4 seconds	7 (premature)

Table 12 - Computation times for ILP-Multicommodity Flow approach.

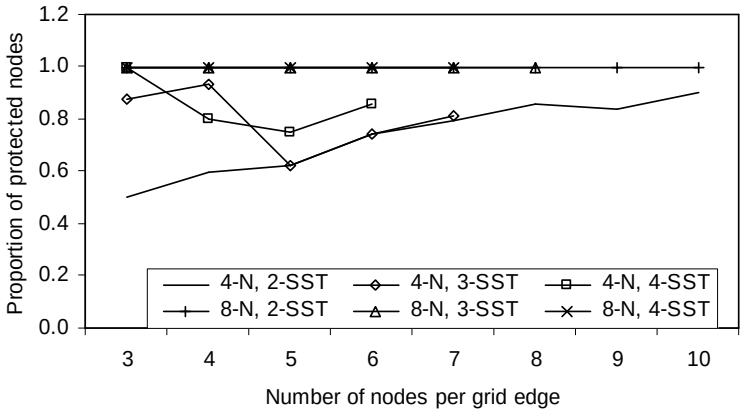

Figure 79 - Proportion of nodes protected using the ILP-MF approach.

Despite its long solution times in general, ILP-Multicommodity Flow formulation does yield the best computation results in terms of minimizing the number of shared node despite its long solution times. In all of the test cases, this approach successfully eliminates the occurrence of shared nodes. With aggressive data aggregation and duplicate suppression, the resultant WSNs should be able to offer a tremendous increase on the amount of monitoring coverage at the same communication duty cycle depending on the number of SS-Trees involved. Another measure of how well SS-Trees are computed is the number of *protected* nodes, which is defined in Section 4.5.1. Figure 79 shows the proportion of nodes protected in each case, and the test cases with 8-neighbour grids all achieved 100% node protection. On the other hand, test cases with sparser 4-neighbour grids and 2 SS-Trees produced the worst protection, though they also incurred the

fastest computation time than other *4-N* counterparts. In all test scenarios, most of the unprotected nodes lie at the fringes of the WSN, and the number of protected nodes will be expected to increase as the nodal population gets larger.

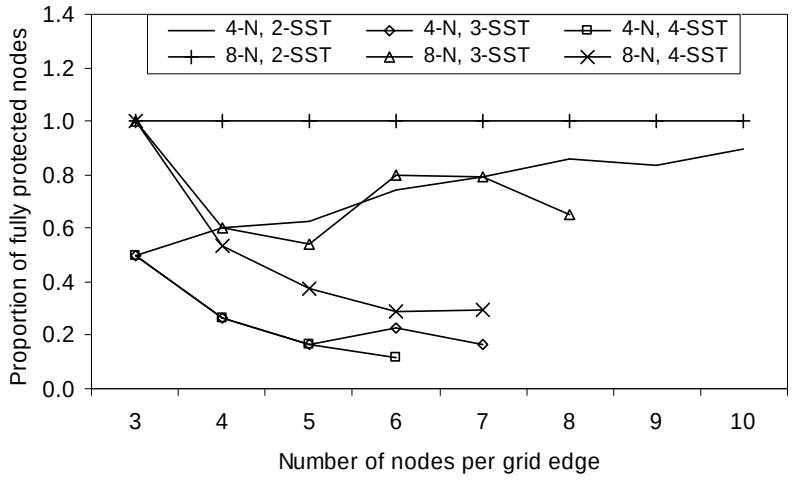

Figure 80 - Proportion of nodes fully protected using the ILP-MF approach.

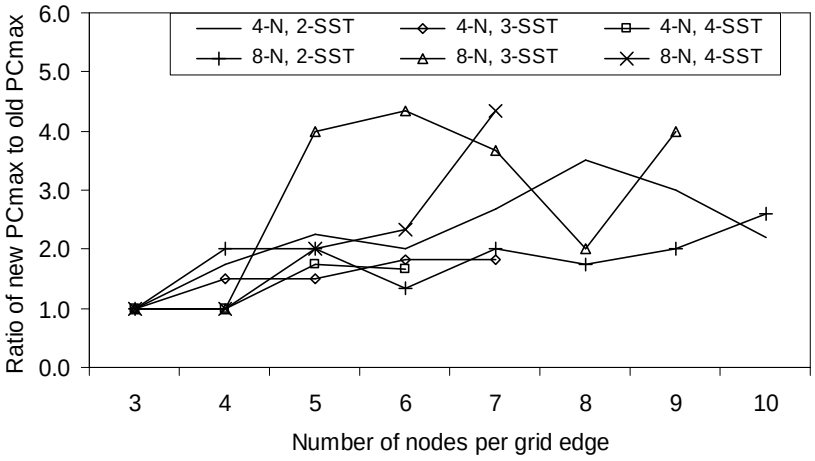

Figure 81 - Increase in path cost after computing SS-Trees using the ILP-MF approach.

Figure 80 shows the proportion of fully protected nodes in each test case. For a given topology, the degree of full protection decreases as the number of SS-Trees to be computed increases. On the other hand for a fixed number of SS-Trees, an increase in network density would also

increase the degree of full protection. Therefore, careful design consideration should be given when balancing the optimal sensing coverage, network connectivity and the number of SS-Trees involved. A good reference on the optimal number of SS-Trees is the bound on the number of domatic partition on a graph as given in Section 1.1, though a more definitive bound is still up in the air because of the affinity of the SS-Tree approach to the much involved MCDP problem.

Besides causing an exponential increase in the solution time, the elimination of shared nodes by the ILP-Multicommodity Flow approach also comes at a price of increased path cost. Assume before computing SS-Trees, the node furthest away from the data sink has a shortest path cost of PC_{max} as determined by Dijkstra's algorithm. Because the ILP-MF formulation does not restrict the path cost of the longest branch on each SS-Tree, the cost of delivering packets from the network fringes to the data sink could become very high under certain conditions. Figure 81 shows such an effect, which plots the ratio of the PC_{max} values before and after SS-Tree computation. As shown in this figure, the highest path cost after computing SS-Trees could reach as high as 4.33 times of the highest path cost recorded on the original WSN topology. An excessively long end-to-end multihop delivery time would profoundly affect how sleep schedules are devised, which eventually leads to a decrease in the monitoring sensitivity provided by the WSN. Therefore intuition suggests that some form of adjustable restriction on path cost should be implemented in the ILP formulation, which will become part of the ongoing research work.

4.5.3 Iterative Algorithm Approach

In terms of computation speed, the iterative algorithm arrives at SS-Tree computation results at a very quick pace, typically in less than 1 second for all of the test cases. As an unpleasant trade-off for achieving faster solution times, however, the algorithm produces a considerable number of shared nodes. Figure 82 shows the number of shared nodes computed in the different test cases using the iterative algorithm. In all of the test cases, the number of shared nodes computed increases more or less in a linear trend corresponding with the number of nodes per grid edge. The slight deviations along the linear trend can be attributed to the shifting of the data sink's coordinates in response to odd/even changes of the number of nodes per grid edge in individual test cases. Out of all test scenarios, cases with a 8-neighbour grid and 4 SS-Trees (*8-N 4-SST*)

produce the most shared nodes, while scenarios with 2 SS-Trees (both *4-N 2-SST* and *8-N 2-SST*) generated the fewest shared nodes on average.

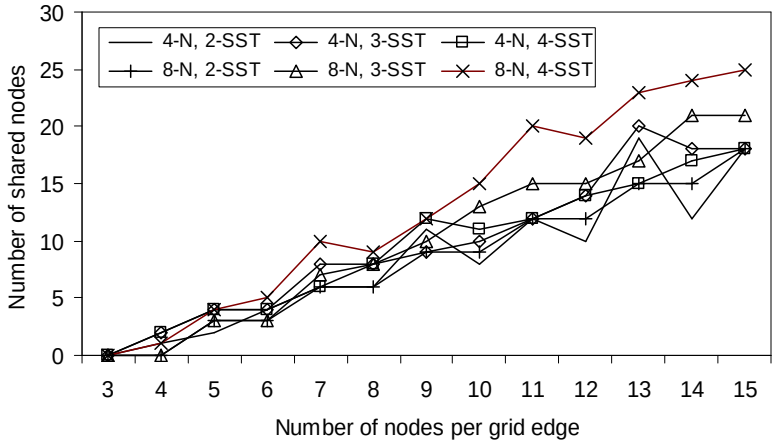

Figure 82 - Number of shared nodes computed using the iterative algorithm approach.

Despite the large number of shared nodes computed, the iterative algorithm does enhance the degree of neighbour-to-neighbour protection and therefore increase the capabilities for event reporting and failure recovery of each node. Figure 83 shows the proportion of nodes protected in each case, and all of the test cases achieved 100% node protection except for a couple of instances.

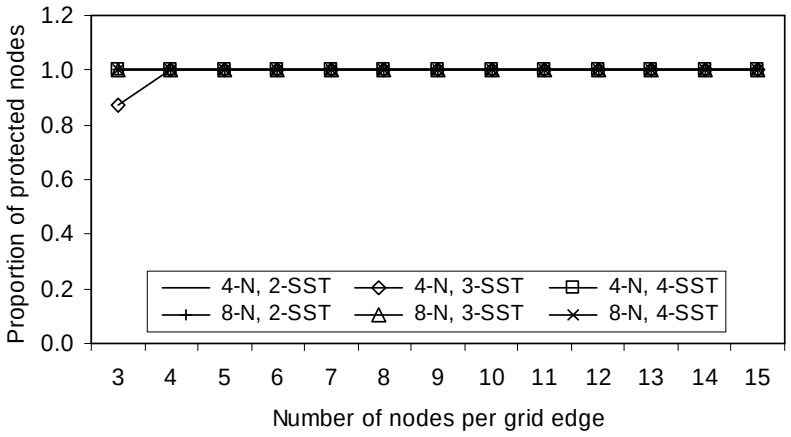

Figure 83 - Proportion of nodes protected using the iterative algorithm approach.

Figure 84 shows the proportion of fully protected nodes in each test case. For a given topology, the degree of full protection decreases as the number of SS-Trees to be computed increases. On the other hand for a fixed number of SS-Trees, an increase in network density would also increase the degree of full protection. Cases of *8-N 2-SST* configuration all achieved 100% protection, though at the expense of each node having multiple co-SS-Tree neighbours. This will lead to increased overhearing and packet collisions during steady state WSN operations. Notwithstanding WSN topologies with small number of nodes, degree of full protection remains quite stable or even increases along with the increase in nodal population. This is especially true for test cases with higher number of SS-Trees, where a larger and denser WSN topology permits better SS-Tree construction. On the other hand, smaller topologies give less leeway in allowing multiple SS-Trees to become neighbours to a particular node. Again, careful design consideration should be given when balancing the optimal nodal deployment strategy, sensing requirements, wireless communication range and the number of SS-Trees involved.

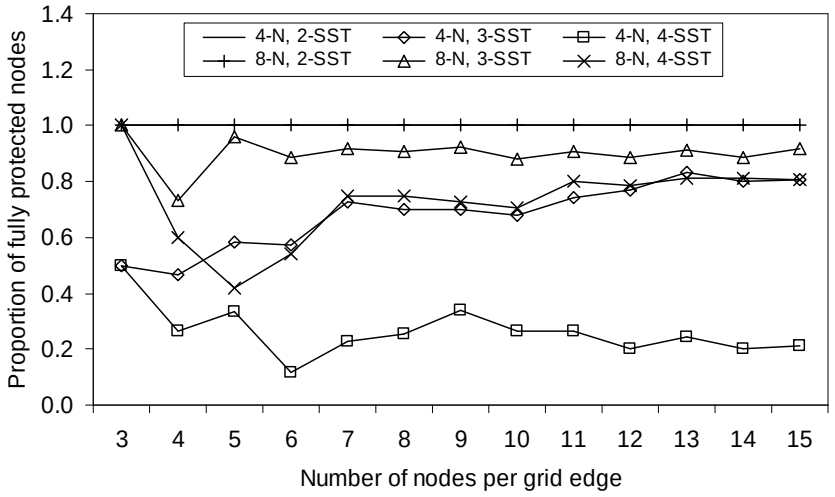

Figure 84 - Proportion of nodes fully protected using the iterative algorithm approach.

In comparison with the two ILP-based approaches, the iterative algorithm offers more promise in practical implementation because of its simplicity and much quicker solution times. Therefore it would be interesting to further investigate how the algorithmic approach affects overall energy efficiency. To evaluate the expected *system lifetime* of a given WSN topology, the normal approach is to first construct a number of SS-Trees and then generate simulated traffic over the

network and measure the energy consumption until network connectivity is no longer sustainable or the sensing coverage becomes inadequate due to nodal failures. However, because of the complexity in building a suitable MAC-level simulation environment that can model system lifetime changes lasting several years for thousands of nodes, detailed packet transactions for message exchanges over the wireless medium cannot be simulated at this time. While a precise measure of the expected system lifetime is not yet available, a rough estimation can still be obtained through relating the expected system lifetime to the number of times, or *rounds*, the iterative SS-Tree computation algorithm can be run consecutively over a given topology. The logic behind this is that after all the shared nodes have drained their batteries, the data sink will have to recompute the SS-Trees based on the revised WSN topology with the shared nodes removed, thereby extending the system lifetime. This SS-Tree recomputation process continues until *less than 50% of all nodes remain connected to the data sink*, which is the benchmark for evaluating the expected operational lifetime of a particular WSN. This definition on system lifetime comes in contrast with node lifetime assumption in Section 4.4, where $\overline{T_{lifetime}}$ is defined to be the time it takes to exhaust the battery supply of each node with no consideration on network connectivity or incidences of shared nodes in SS-Trees

Simulations on determining the expected operational lifetime and other related performance metrics of a given WSN are based the following idealized assumptions:

1. Every node begins operation with the same amount of battery power.
2. Battery depletion is the only cause for nodal failures.
3. The energy cost in recomputing SS-Trees is negligible compared to that incurred during normal WSN operations.
4. Because of efficient data aggregation and duplicate suppression procedures, the energy depletion rate for all nodes across the WSN is approximately the same.

If no shared nodes were ever computed on a given WSN, then theoretically the expected operational lifetime of the WSN is equal to the product of its normal operational lifetime under a particular duty cycle and the number of SS-Trees involved. Therefore for test cases with 4 SS-Trees computed, the upper bound on expected lifetime becomes 4 times that of cases without using SS-Trees. Figure 85 shows the expected increase in WSN system lifetime, and from the

numerical results it can be seen that denser topologies and a higher number of SS-Trees will generally extend System lifetime longer. Still, only tiny WSNs (i.e., 3 x 3 to 4 x 4) are able to achieve the theoretical limit of expected lifetime since their SS-Tree configurations do not contain any shared nodes. On the other hand, a few cases do not yield any system lifetime extension at all because the WSN topology becomes prematurely disconnected below the 50%-threshold in the first round of SS-Tree computation by shared nodes that appear at the right places, which leaves the rest of the nodal population with plenty of battery power left to waste. In fact, shared nodes that are concentrated at a close path distance to the data sink will lead to a shorter expected lifetime since they expedite the occurrence of topology disconnection.

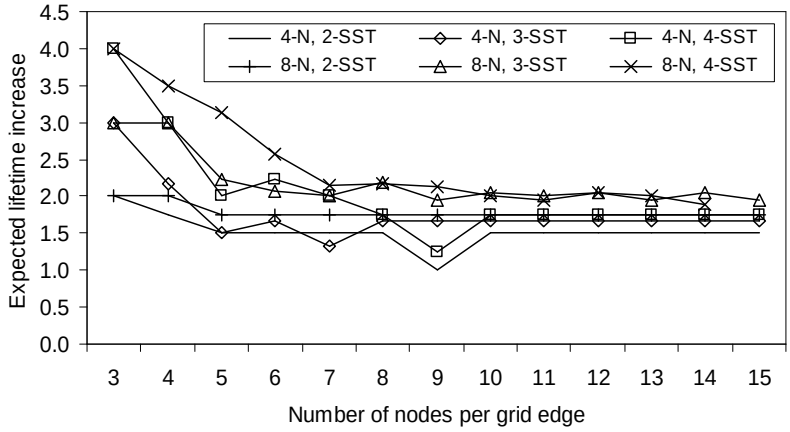

Figure 85 - Expected system lifetime increase through the SS-Trees computation algorithm.

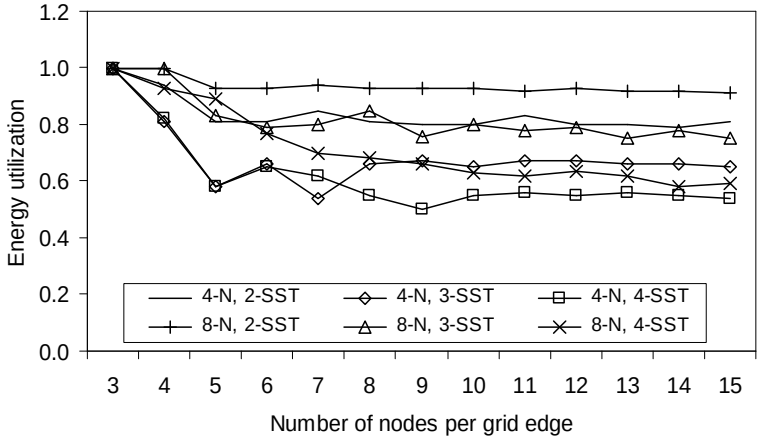

Figure 86 - Energy utilization of the WSN nodes with the SS-Trees computation algorithm.

Another useful metric for evaluating the effectiveness of the SS-Tree computation algorithm is energy utilization, which measures the total amount of energy expended across the WSN versus the total amount initial energy given to every node. Figure 86 shows the degree of energy utilization occurred in the test scenarios, and again small WSNs are able to achieve maximum energy utilization since their SS-Tree configurations do not contain any shared nodes. On the other hand, the test scenario *8-N 2-SST* is able to attain over 90% energy utilization in addition to excelling in extending WSN operational lifetime. Future research work using full-scale WSN data traffic simulations will verify this combination's prowess in computing SS-Trees by taking into account MAC-level effects such as overhearing and packet collisions as well as other degradation effects.

5 CONCLUDING REMARKS

5.1 CONTRIBUTIONS OF THIS WORK

Designing and implementing a battery-operated wireless sensor network application is fraught with challenges with regards to energy efficiency, cost, sensing coverage, communication capabilities, and overall reliability. In this work, the relationship between energy efficiency, temporal sensing coverage, transmission range assignment and communication protocol design is studied in detail under the proposed cross-layer design approach. By exploiting the inherent multihop, multipath routing property of mesh WSNs, multiple SS-Trees can be formed to maximize energy efficiency through sleep scheduling, as well as to increase temporal sensing coverage for critical event-driven data. In the upper layers, the unique traffic profile of event-driven sensing applications is coupled with simplified network structure of each SS-Tree to achieve low control overhead in packet routing and network maintenance. Selected results of this work have been published in or are to be submitted to a number of academic journals and conferences [63]-[68].

Layer	Features/Advantages
Link/MAC	• Minimize energy loss from idle listening via sleep scheduling, overhearing via neighbourhood sleep schedule interleaving, packet collisions via traffic engineering, and control overhead
Network	• Simplify routing procedure through directional flooding on the uplink to the data sink and source routing on the downlink • Balance latency and energy use
Transport	• Rely on hop-by-hop acknowledgement instead of end-to-end acknowledgement
Application	• Increase temporal sensing coverage for event-driven data reporting under a given operational duty cycle

Table 13 - Summary of cross-layer features and advantages in the SS-Tree approach.

Table 13 offers a summary of the cross-layer considerations in the SS-Tree approach. Under the proposed SS-Tree design approach, optimal parameter selection ultimately depends on sensing

application requirements. Summarizing the performance evaluation results, it is found that the combination of a low duty cycle sleep schedule, long active period duration with respect to per hop delivery time, low traffic load, simple MAC signalling, as well as a long transmission range, is able to best balance the various design considerations and provide better performance in extending system lifetime and reducing data reporting latency. The SS-Tree concept plays a significant role in improving timing performance in event-driven data reporting, which is essential in most of the WSN applications. On the other hand, component cost and hardware reliability issues may lead to the selection of a different set of parameters, which suggests trade-offs in sensing application performance and affordability are worth serious deliberation during network planning and pre-deployment stage.

To further improve the timing performance of data reporting, an implicit acknowledgement scheme is proposed in the MAC layer that makes the most of the open wireless channel and multihop routes in WSNs. Compared with standard ACK schemes, sleep scheduling is better served when combined with the IACK/EACK scheme at the MAC layer and the push-pull traffic sequencing concept because of their better adaptation to the application-specific multihop WSN environment. However, it may be necessary to implement full RTS/CTS mechanism in high traffic volume situations, which is assumed to be irrelevant in the current system model.

Three methods are suggested to compute SS-Trees, one based on a depth-first greedy iterative search algorithm and the other two based on integer linear programming techniques. Furthermore, the first two are based on the Dijkstra's shortest path algorithm while the third method is based on the network flow model. The computation of the SS-Trees is closely related to the maximum domatic partition problem, which is NP-Complete, and the objective is to minimize the number of shared nodes among the SS-Trees such that sleep times can be maximized while a certain level of sensing coverage can be guaranteed. In computing the SS-Trees, the ILP-based approaches thoroughly minimize the number of shared nodes, though at a price of exponential solution times, increased path cost and lower inter-node protection. In contrast, the iterative algorithm approach returns a solution quickly with excellent inter-node protection, but it produces a higher number of shared nodes.

5.2 *FUTURE RESEARCH DIRECTIONS*

The SS-Tree concept offers tremendous potential in advancing the research and development of wide-area mesh-based surveillance WSNs. While the bulk of the issues have been dealt with in this work, the following areas will be explored further in the next stage of research:

- *Sleep scheduling with variable transmission range assignment* - Throughout this work, it is assumed that the radio transceivers can only offer fixed transmission range after deployment. Therefore, it would be interesting to evaluate the impact of incorporating the dynamically variable transmission range assignment feature to the overall SS-Tree design. Specifically, if each node is equipped with variable range transceivers, then the minor adjustments to the overall network topology is possible such that the resultant computed SS-Trees enjoy better sensing coverage. On the other hand, the corresponding increase in component cost and control overhead deserve attention as well.

- *Sleep scheduling with spatial sensing coverage considerations* - When computing the sleep schedules, it is assumed that the nodes are distributed in a manner such that the sensing field is thoroughly covered and no coverage redundancy exists. Therefore, it is logical to extend the SS-Tree concept to include spatial sensing coverage considerations in the future. For example, the degree of spatial coverage over a particular area may vary dynamically according to application requirements. Therefore, such issues need to be taken care of when SS-Trees are assigned to the individual nodes by the data sink.

- *More efficient SS-Tree computation methods* - The ILP-based techniques for computing SS-Trees are too computation-intensive for actual WSN implementation, while the sub-optimality of the iterative search algorithm calls for better SS-Tree computation methods in producing fewer shared nodes efficiently. One possible solution to be focused on is the development of a breadth-first search algorithm for computing SS-Trees, which comes in contrast with the proposed greedy depth-first iterative approach.

- *Refined MAC design for expedited packet delivery* - The IACK/EACK scheme is an example of application-specific MAC design in WSNs, and it serves as a good starting

point for protocol refinement in further enhancing the timing performance of data reporting. Incorporating the RTS/CTS mechanism into SS-Tree design can be useful if event-driven traffic load increases to the point where simply interleaving the sleep schedules of neighbouring nodes is not enough to prevent packet collisions.

- *Efficient neighbourhood discovery and failure recovery strategies* - Efficient neighbourhood discovery and failure recovery strategies were not discussed in detail in this work, which deserve in-depth investigation in the future. The extra overhead in neighb

- *Practical implementation of SS-Trees on actual WSN applications* - Idealized assumptions in the system model of the SS-Tree concept may not be completely valid in practical WSN implementations. Therefore, the SS-Tree concept can be best evaluated on actual sensor nodes and WSNs under real-life application scenarios.

REFERENCES

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "A Survey on Wireless Sensor Networks", *IEEE Communications Magazine*, vol. 40, no. 8, August 2002, pp. 102-114.

[2] W. Ye, J. Heidemann, and D. Estrin, "An Energy-Efficient MAC Protocol for Wireless Sensor Networks", *Proceedings of IEEE Conference on Computer Communications*, vol. 3, June 2002, pp. 1567-1576.

[3] C.-Y. Chong and S.P. Kumar, "Sensor Networks: Evolution, Opportunities, and Challenges", *Proceedings of the IEEE*, vol. 91, no. 8, August 2003, pp.1247-1256.

[4] V. Raghunathan, C. Schurgers, S. Park, and M.B. Srivastava, "Energy-Aware Wireless Microsensor Networks", *IEEE Signal Processing Magazine*, March 2002, pp. 40-50.

[5] K. Du, J. Wu and D. Zhou, "Chain-Based Protocols for Data Broadcasting and Gathering in the Sensor Networks", *Proceedings of the International Parallel and Distributed Processing Symposium*, April 2003.

[6] G.J. Pottie and W.J. Kaiser, "Wireless Integrated Network Sensors", *Communications of the ACM*, vol. 43, no. 5, May 2000, pp. 51-58.

[7] *ANSI/IEEE Std. 802.11 1999 Edition, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE, Piscataway, N.J., 1999.

[8] C.E. Perkins, E.M. Royer, S.R. Das, and M.H. Marina, "Performance Comparison of Two On-Demand Routing Protocols for Ad Hoc Networks", *IEEE Personal Communications Magazine*, vol. 8, no. 1, February 2001, pp. 16-28.

[9] S. Lindsey, C. Raghavendra, and K.M. Sivalingam, "Data Gathering Algorithms in Sensor Networks Using Energy Metrics", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 9, September 2002, pp. 924-935.

[10] Q. Li and D. Rus, "Global Clock Synchronization in Sensor Networks", *Proceedings of IEEE Conference on Computer Communications*, March 2004.

[11] W.B. Heinzelman, A.P. Chandrakasan, and H. Balakrishnan, "An Application-Specific Protocol Architecture for Wireless Microsensor Networks", *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, October 2002, pp. 660-669.

[12] J.M. Reason and J.M. Rabaey, "A Study of Energy Consumption and Reliability in a Multi-Hop Sensor Network", *ACM Mobile Computing and Communications Review*, vol. 8, no. 1, January 2004, pp. 84-97.

[13] M.L. Sichitiu, "Cross-Layer Scheduling for Power Efficiency in Wireless Sensor Networks", *Proceedings of IEEE Conference on Computer Communications*, 2004.

[14] J.E. Elson, L. Girod and D. Estrin, "Fine-Grained Network Time Synchronization Using Reference Broadcasts", *Proceedings of the 5th Symposium on Operating System Design and Implementation*, December 2002.

[15] A. Muqattash and M. Krunz, "CDMA-Based MAC Protocol for Wireless Ad Hoc Networks", *Proceedings of ACM International Symposium on Mobile Ad Hoc Networking and Computing*, June 2003, pp. 153-164.

[16] T. van Dam and K. Langendoen, "An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks", *Proceedings of ACM Conference on Networked Sensor Systems*, November 2003, pp. 171-180.

[17] A. Boukerche, X. Cheng, and J. Linus, "Energy-Aware Data-Centric Routing in Microsensor Networks", *Proceedings of ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, September 2003, pp. 42-49.

[18] U. Cetintemel, A. Flinders, and Y. Sun, "Power-Efficient Data Dissemination in Wireless Sensor Networks", *Proceedings of International ACM Workshop on Data Engineering for Wireless and Mobile Access*, September 2003.

[19] J.R. Vig, *Introduction to Quartz Frequency Standards*, tech. report, Army Research Laboratory, Electronics and Power Sources Directorate, October 1992.

[20] Y. Zhang and L. Cheng, "Cross-Layer Optimization for Sensor Networks", *Proceedings of 3rd New York Metro Area Networking Workshop*, September 2003.

[21] A. Manjeshwar and D.P. Agrawal, "TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks", *Proceedings of the 15th International Parallel and Distributed Processing Symposium*, 2001, pp. 2009-2015.

[22] S. Ganeriwal, R. Kumar, and M.B. Srivastava, "Timing-sync Protocol for Sensor Networks", *Proceedings of ACM Conference on Networked Sensor Systems*, November 2003, pp. 138-149.

[23] H. Gupta, S.R. Das, and Q. Gu, "Connected Sensor Cover: Self-Organization of Sensor Networks for Efficient Query Execution", *Proc. ACM MobiHoc*, Jun. 2003, pp. 189-200.

[24] N. Bulusu, J. Heidemann, and D. Estrin, "GPS-Less Low-Cost Outdoor Localization for Very Small Devices", *IEEE Personal Communications Magazine*, vol. 7, no. 5, October 2000, pp. 28-34.

[25] F. Ye, A. Chen, S. Liu, and L. Zhang, "A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks", *Proceedings of the 7th Annual International Conference on Computer Communications and Networks*, 2001, pp. 304-309.

[26] J. Albowicz, A. Chen, and L. Zhang, "Recursive Position Estimation in Sensor Networks", *Proceedings of IEEE International Conference on Network Protocols*, 2001, pp. 35-41.

[27] F. Stann and J. Heidemann, "RMST: Reliable Data Transport in Sensor Networks", *Proceedings of 1st International Workshop on Sensor Net Protocols and Applications*, 2003, pp.102-112.

[28] D.L. Mills, "Internet Time Synchronization: The Network Time Protocol", in Z. Zang and T.A. Marsland, editors, *Global States and Time in Distributed Systems*, IEEE Computer Society Press, 1994.

[29] Y. Zou and K. Chakrabarty, "A Distributed Coverage- and Connectivity-Centric Technique for Selecting Active Nodes in Wireless Sensor Networks", *IEEE Trans. Computers*, vol. 54, no. 8, Aug. 2005, pp. 978-991.

[30] P.E. Black, "Dijkstra's Algorithm", National Institute of Standards and Technology, 2004, http://www.nist.gov/dads/HTML/dijkstraalgo.html.

[31] T. Moscibroda and R. Wattenhofer, "Maximizing the Lifetime of Dominating Sets", *Proc. IEEE IPDPS*, Apr. 2005.

[32] A. Sinha and A. Chandrakasan, "Dynamic Power Management in Wireless Sensor Networks", *IEEE Design and Test of Computers*, vol. 18, no. 2, 2001, pp. 62-74.

[33] G. Xing, C. Lu, Y. Zhang, Q. Huang, and R. Pless, "Minimum Power Configuration in Wireless Sensor Networks", *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, Urbana-Champaign, IL, USA, 25-28 May, 2005, pp. 390-401.

[34] Q. Fang, J. Gao, and L.J. Guibas, "Locating and Bypassing Routing Holes in Sensor Networks", *Proceedings of IEEE Conference on Computer Communications*, 2004.

[35] B. Karp and H. Kung, "GPSR: Greedy Perimeter Stateless Routing for Wireless Networks", *Proceedings of ACM International Conference on Mobile Computing and Networking*, 2000, pp. 243-254.

[36] G. Robins and A. Zelikovsky, "Improved Steiner Tree Approximation in Graphs", *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 2000, pp. 770-779.

[37] R. Min, M. Bhardwaj, S.-H. Cho, N. Ickes, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan, "Energy-Centric Enabling Technologies for Wireless Sensor Networks", *IEEE Wireless Communications Magazine*, vol. 9, no. 4, August 2002, pp.28-39.

[38] J. Pan, Y.T. Hou, L. Cai, Y. Shi, and S.X. Shen, "Topology Control for Wireless Sensor Networks", *Proceedings of ACM International Conference on Mobile Computing and Networking*, September 2003, pp. 286-299.

[39] M. Stemm and R.H. Katz, "Measuring and Reducing Energy Consumption of Network Interfaces in Hand-held Devices", *IEICE Transactions on Communications*, vol. E80-B, no. 8, Aug. 1997, pp. 1125-1131.

[40] A. Ephremides, "Energy Concerns in Wireless Networks", *IEEE Wireless Communications Magazine*, vol. 9, no. 4, August 2002, pp.48-59.

[41] C. Schurgers, G. Kulkarni and M.B. Srivastava, "Distributed On-Demand Address Assignment in Wireless Sensor Networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 10, October 2002, pp. 1056-1065.

[42] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-Resilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks", *ACM Mobile Computing and Communications Review*, vol. 5, no. 4, October 2001, pp. 10-24.

[43] Y. Xu, S. Bien, Y. Mori, J. Heidemann and D. Estrin, *Topology Control Protocols to Conserve Energy in Wireless Ad Hoc Networks*, Technical Report, Center for Embedded Networked Sensing (CENS), CA, 2003.

[44] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. Srivastava, "Coverage Problems in Wireless Ad-Hoc Sensor Network", *Proceedings of IEEE Conference on Computer Communications*, 2001, pp. 1380-1387.

[45] X.-Y. Li, P.-J. Wan, and O. Frieder, "Coverage in Wireless Ad Hoc Sensor Networks", *IEEE Transactions on Computers*, vol. 52, no. 6, June 2003, pp. 753-762.

[46] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated Coverage and Connectivity Configuration in Wireless Sensor Networks", *ACM Transactions on Sensor Networks*, vol. 1, no. 1, 2005, pp. 36-72.

[47] S. Slijepcevic and M. Potkonjak, "Power Efficient Organization of Wireless Sensor Networks", *Proceedings of IEEE International Conference on Communications*, 2001, pp. 472-476.

[48] K. Dasgupta, M. Kukreja and K. Kalpakis, "Topology-Aware Placement and Role Assignment for Energy-Efficient Information Gathering in Wireless Sensor Networks", *Proceedings of the 8th IEEE International Symposium on Computers and Communication*, vol. 1, June 2003, pp. 341-348.

[49] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed Diffusion for Wireless Sensor Networking", *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, February 2003, pp. 2-16.

[50] C.-F. Hsin and M. Liu, "Network Coverage Using Low Duty-Cycled Sensors: Random and Coordinated Sleep Algorithms", *Proc. IPSN'04*, April 2004, pp. 433-442

[51] J. Carle and D. Simplot-Ryl, "Energy-Efficient Area Monitoring for Sensor Networks", *IEEE Computer Magazine*, February 2004, pp. 40-46.

[52] N. Li and J.C. Hou, "Topology Control in Heterogeneous Wireless Networks: Problems and Solutions", *Proceedings of IEEE Conference on Computer Communications*, 2004.

[53] A.J. Goldsmith and S.B. Wicker, "Design Challenges for Energy-Constrained Ad Hoc Wireless Networks", *IEEE Wireless Communications Magazine*, vol. 9, no. 4, August 2002, pp.8-27.

[54] R. Ramanathan and R. Rosales-Hain, "Topology Control of Multihop Wireless Networks using Transmit Power Adjustment", *Proceedings of IEEE Conference on Computer Communications*, 2000, pp. 404-413.

[55] H. Takagi and L. Kleinrock, "Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals", *IEEE Transactions on Communications*, vol. 32, no. 3, March 1984, pp. 246-257.

[56] S. Rhee and S. Liu, "Wireless Sensor Nets Demand Trade-offs", *EE Times*, October 31, 2003.

[57] P.-J. Wan, K.M. Alzoubi, and O. Frieder, "Distributed Construction of Connected Dominating Set in Wireless Ad Hoc Networks", *Mobile Networks and Applications*, vol. 9, issue 2, Kluwer Academic Publishers, Apr. 2004, pp. 141-149.

[58] J. Wu and H. Li, "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks", *Proceedings of International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 1999, pp. 7-14.

[59] U. Feige, M.M. Holldorsson, G. Kortsarz, and A. Srinivasan, "Approximating the Domatic Number", *SIAM Journal of Computing*, vol. 32, no. 1, 2000, pp. 172-195.

[60] Maxstream Inc., http://www.maxstream.net/.

[61] Atmel Corporation, http://www.atmel.com/.

[62] Crossbow Inc., http://www.xbow.com/.

[63] R. Ha, P.-H. Ho and X.S. Shen, "SS-Trees: A Cross-Layer Organizational Approach for Mesh-based Wide-area Wireless Sensor Networks", *Proceedings of IEEE BroadNets*, October 2005, pp. 885-894.

[64] R. Ha, P.-H. Ho and X.S. Shen, "Cross-Layer Organization for Wireless Sensor Networks Using Sense-Sleep Trees", *Proceedings of WirelessCom 2005*, June 2005, pp. 952-957.

[65] R. Ha, P.-H. Ho and X.S. Shen, "Cross-Layer Application-Specific Wireless Sensor Network Design with Single-Channel CSMA MAC over Sense-Sleep Trees", accepted by *Elsevier Journal: Computer Communications*.

[66] R. Ha, P.-H. Ho, X.S. Shen, and J. Zhang, "Sleep Scheduling for Wireless Sensor Networks via Network Flow Model", accepted by *Elsevier Journal: Computer Communications*.

[67] R. Ha, P.-H. Ho and X.S. Shen, "Optimal Sleep Scheduling with Transmission Range Assignment in Application-Specific Wireless Sensor Networks", accepted by *International Journal of Sensor Networks*.

[68] R. Ha, P.-H. Ho and X.S. Shen, "Sleep Scheduling and Temporal Sensing Coverage Issues in Application-Specific Wireless Sensor Networks", to be submitted to *IEEE Transactions on Wireless Communications*.

[69] L. van Hoesel, T. Nieberg, J. Wu, and P.J.M. Havinga, "Prolonging the Lifetime of Wireless Sensor Networks by Cross-Layer Interaction", *IEEE Wireless Communications*, vol. 11, no. 6, December 2004, pp. 78-86.

[70] J.A. Stankovic, T.F. Abdelzaher, C. Lu, L. Sha, and J.C. Hou, "Real-Time Communication and Coordination in Embedded Sensor Networks", *Proceedings of the IEEE*, vol. 91, no. 7, July 2003, pp. 1002-1022.

[71] J. Polastre, J. Hill and D. Culler, "Versatile Low Power Media Access for Wireless Sensor Networks", *Proceedings of SenSys'04*, November 2004.

[72] S. Kumar, T.H. Lai and J. Balogh, "On k-Coverage in a Mostly Sleeping Sensor Network", *Proceedings of ACM International Conference on Mobile Computing and Networking*, September 2004, pp. 144-158.

[73] Y. Li, B. Bakkaloglu and C. Chakrabarti, "A Comprehensive Energy Model and Energy-Quality Evaluation of Wireless Transceiver Front-Ends", *Proceedings of IEEE Workshop on Signal Processing Systems Design and Implementation*, November 2005, pp. 262-267.

[74] P. Chen, B. O'Dea and E. Callaway, "Energy-Efficient System Design with Optimum Transmission Range for Wireless Ad Hoc Networks", *Proceedings of IEEE International Conference on Communications*, April 2002, pp. 945-952.

[75] A.Y. Wang and C.G. Sodini, "A Simple Energy Model for Wireless Microsensor Transceivers", *Proceedings of IEEE Global Telecommunications Conference*, November 2003, pp. 3205-3209.

[76] J. Aslam, Q. Li, and D. Rus, “Three Power-Aware Routing Algorithms for Sensor Networks”, *Wireless Communications and Mobile Computing*, vol. 2, no. 3, Mar. 2003, pp. 187-208.

[77] J.-H. Chang and L. Tassulas, “Maximum Lifetime Routing in Wireless Sensor Networks”, *IEEE/ACM Transactions on Networking*, vol. 12, no. 4, Aug. 2004, pp. 609-619.

[78] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris, “Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks”, *Wireless Networks*, vol. 8, 2002, pp. 481-494.

[79] G. Xing, X. Wang, Y. Zhang, C. Lu, R. Pless, and C. Gill, “Integrated Coverage and Connectivity Configuration for Energy Conservation in Sensor Networks”, *ACM Transactions on Sensor Networks*, vol. 1, no. 1, 2005, pp. 36-72.

[80] M.R. Garey and D.S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*, W.H. Freeman and Company, San Francisco, 1979.

[81] B. Sundararaman, U. Buy, and A.D. Kshemkalyani, “Clock Synchronization in Wireless Sensor Networks: A Survey”, *Ad-Hoc Networks*, vol. 3, no. 3, 2005, pp. 281-323.

[82] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D.E. Culler, and K.S.J. Pister, “System Architecture Directions for Networked Sensors”, *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93-104.

[83] E.W. Weisstein, “Circle-Circle Intersection”, from *MathWorld - a Wolfram Web Resource*, http://mathworld.wolfram.com/Circle-CircleIntersection.html.

Appendix A - List of Abbreviations

ACK	Acknowledgement
ADC	Analog-to-Digital Converter
C/D	Control or Data
CDS	Connected Dominating Set
CDMA	Code Division Multiple Access
CSMA/CA	Carrier Sense Multiple Access/Collision Avoidance
CTS	Clear To Send
DAC	Digital-to-Analog Converter
EACK	Explicit Acknowledgement
FDMA	Frequency Division Multiple Access
FEC	Forward Error Correction
GPS	Global Positioning System
IACK	Implicit Acknowledgement
ID	Identifier
IF	Intermediate Frequency
ILP	Integer Linear Programming
LNA	Low Noise Amplifier
MAC	Medium Access Control
MCDS	Minimum Connected Dominating Set
MDP	Maximum Domatic Partition
OS	Operating System
PA	Power Amplifier
QoS	Quality of Service
RF	Radio Frequency
RTS	Request To Send
SS-Tree/SST	Sense-Sleep-Tree
TDMA	Time Division Multiple Access
WSN	Wireless Sensor Network

Appendix B - Additional Information on Derivations of Equations

Section 2.4.3, Equation (11)

$$\overline{T_{event}} = \begin{cases} \dfrac{T_{active}}{2\rho} + T_{sense} + N_{hop} T_{hop} & \text{for } N_{hop} \le \alpha, \\ \dfrac{T_{active}}{2\rho} + \left\lfloor \dfrac{N_{hop} - \alpha}{\beta} \right\rfloor \dfrac{T_{active}}{\rho} - T_{sense} + \left[\left(N_{hop} - \alpha \right) \bmod \beta \right] T_{hop} & \text{otherwise.} \end{cases}$$

where $\alpha = \left\lfloor \dfrac{T_{active} - T_{sense}}{T_{hop}} \right\rfloor$ is the average number of hops traversable in the initial active period the event is first detected, and $\beta = \left\lfloor \dfrac{T_{active}}{T_{hop}} \right\rfloor$ is the maximum number of hops traversable in 1 active period. With the assumption of uniform probability of event occurrence over time, the $\dfrac{T_{active}}{2\rho}$ term denotes the expected time between an event happening and the subsequent sensing opportunity in the next active period. Note that the node takes a sensor reading at the start of each active period and then shuts off the sensor unit. If $N_{hop} \le \alpha$, then the end-to-end multihop packet transmission from the node to the data sink can be completed within a single active period. Otherwise, the end-to-end transmission has to span more than one sleep cycle. The exact number of extra sleep cycles to be spanned is determined by $\left\lfloor \dfrac{N_{hop} - \alpha}{\beta} \right\rfloor$. On the other hand, the $\left[\left(N_{hop} - \alpha \right) \bmod \beta \right] T_{hop}$ term accounts for the amount of time taken in the last active period to pass the packet over the last hops.

Section 2.4.3, Equation (12)

$$\overline{T_{timer}} = \left\lfloor \frac{N_{hop}}{\beta} \right\rfloor \frac{T_{active}}{\rho} + \left(N_{hop} \bmod \beta \right) T_{hop}.$$

Similar to the previous equation, the total amount of time for completing the end-to-end packet transfer depends on the number of sleep cycles to be spanned, which is given by $\left\lfloor \frac{N_{hop}}{\beta} \right\rfloor$. On the other hand, the $\lfloor N_{hop} \bmod \beta \rfloor T_{hop}$ term accounts for the amount of time taken in the last active period to pass the packet over the last hops.

Section 2.4.3, Equation (14)

$$\overline{T_{req}} = \begin{cases} P_1\left[\frac{(1-\rho)T_{active}+\rho T_{hop}}{2\rho} + \left\lfloor \frac{2N_{hop}}{\beta} \right\rfloor \frac{T_{active}}{\rho} + (N_{hop} \bmod \beta)T_{hop}\right] & \text{for } N_{hop} \le \frac{\gamma}{2}, \\ \quad + 2P_2 N_{hop} T_{hop} & \\ P_1\left[\frac{(1-\rho)T_{active}+\rho T_{hop}}{2\rho} + \left\lfloor \frac{2N_{hop}}{\beta} \right\rfloor \frac{T_{active}}{\rho} + \lfloor (2N_{hop}) \bmod \beta \rfloor T_{hop}\right] & \\ \quad + P_2\left[\left\lfloor \frac{2N_{hop}-\gamma}{\beta} \right\rfloor \frac{T_{active}}{\rho} + \lfloor (2N_{hop}-\gamma) \bmod \beta \rfloor T_{hop}\right] & \text{otherwise,} \end{cases}$$

where $P_1 = \frac{(1-\rho)T_{active}+\rho T_{hop}}{T_{active}}$ is the probability that the data request arriving at the data sink has to be processed in the next active period, $P_2 = \frac{\rho(T_{active}-T_{hop})}{T_{active}}$ is the probability that the data request arriving at the data sink can be processed instantaneously, and $\gamma = \left\lfloor \frac{T_{active}+T_{hop}}{2T_{hop}} \right\rfloor$ is the average number of hops traversable in the initial active period the data request is received.

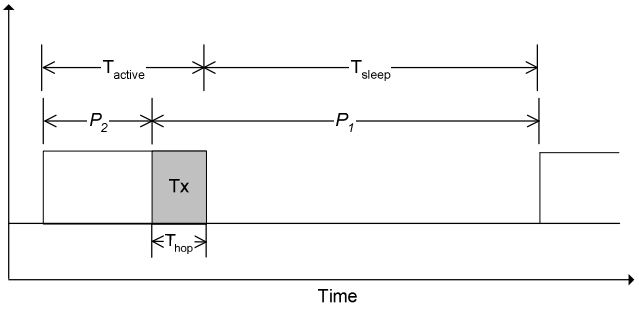

Figure 87 - Roles of P_1 and P_2 in request-driven data reporting.

Figure 87 helps to illustrate the relationship of P_1 and P_2 with the other timing parameters in Equation (14). The $\frac{(1-\rho)T_{active} + \rho T_{hop}}{2\rho}$ term gives the expected time between a data request generated and the start of the next active period.

Appendix C - MAC Simulator Description

Purpose:

Find out the approximate end-to-end propagation and processing time for push-pull traffic sequencing with respect to different parameters in network topology, packet length, and packet loss probability for both IACK and EACK schemes. The MAC protocol details are described in Section 3.4.3 and simulation results have been presented in Section 4.2.

Variable definition:

T_{ack} - time for EACK

T_{cd} - time for C/D data

p - probability of packet delivery success

N_{hop} - number of hops

T_{sim} - total simulation time to be used

Network initialization:

The network topology is assumed to be a linear chain between the data sink and the leaf node, with N_{hop} values ranging from 2 to 14 hops.

Packet fields:

Each packet is loaded with the following fields:

- Packet type
- Packet time
- Sent timestamp
- Packet sequence ID
- Packet source node
- Packet destination node

Approach for treating packet loss from channel errors:

- Determined by packet receiver.
- Receiver replies a negative ACK (NACK) back to sender to indicate packet loss.
- After receiving NACK, sender changes packet timestamp and retransmits packet.
-

Approach for treating packet loss from packet collision:

The following pseudocode demonstrates how to handle packet loss from the collision of 2 packets:

```
if (pkt1_timestamp >= pkt2_timestamp)
       // packet 1 sent earlier than packet 2
       packet collision = true if (pkt1_timestamp - pkt2_timestamp) < pkt1_duration
else
       // packet 2 sent earlier than packet 1
       packet collision = true if (pkt2_timestamp - pkt1_timestamp) < pkt2_duration
```

State transition diagrams for EACK only and IACK/EACK schemes:

The MAC simulator will perform packet transmission over a multihop path with a choice of either ACK scheme. The state transition diagrams for EACK only and IACK/EACK schemes are shown in Figure 88 and Figure 89, respectively.

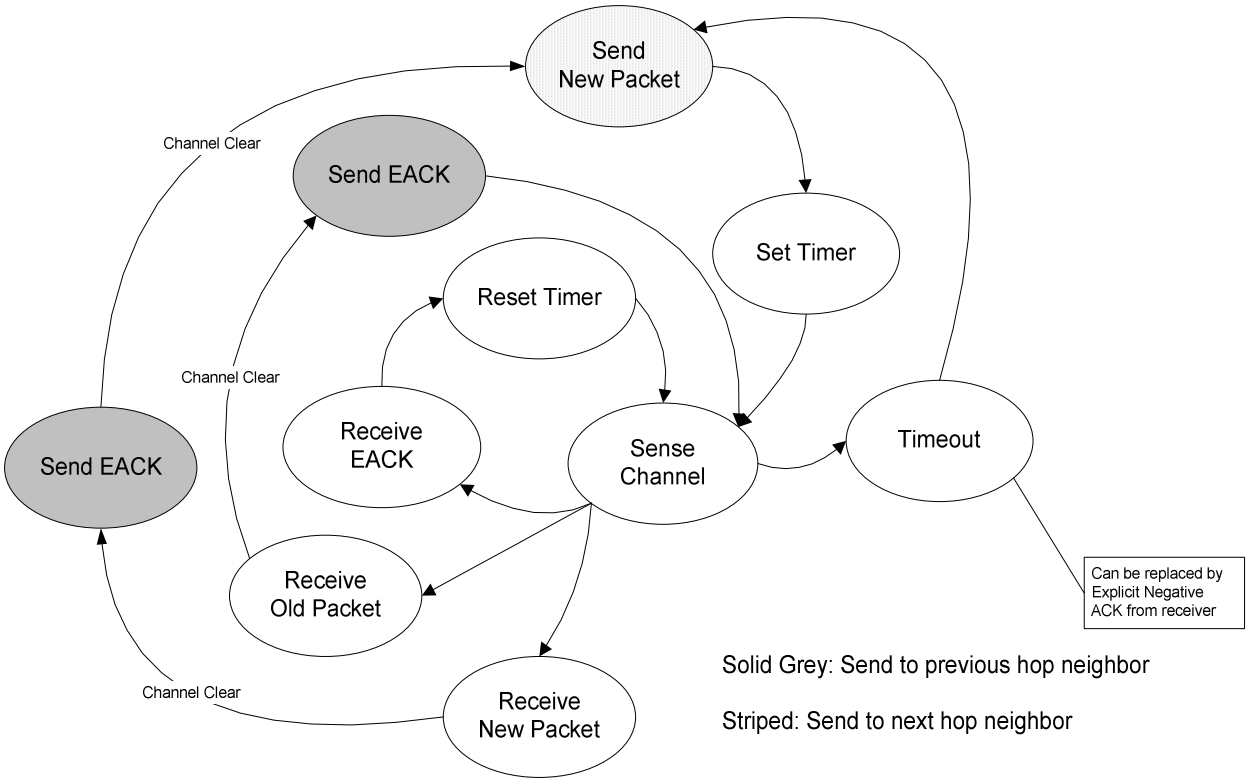

Figure 88 - State transition diagram for EACK only scheme.

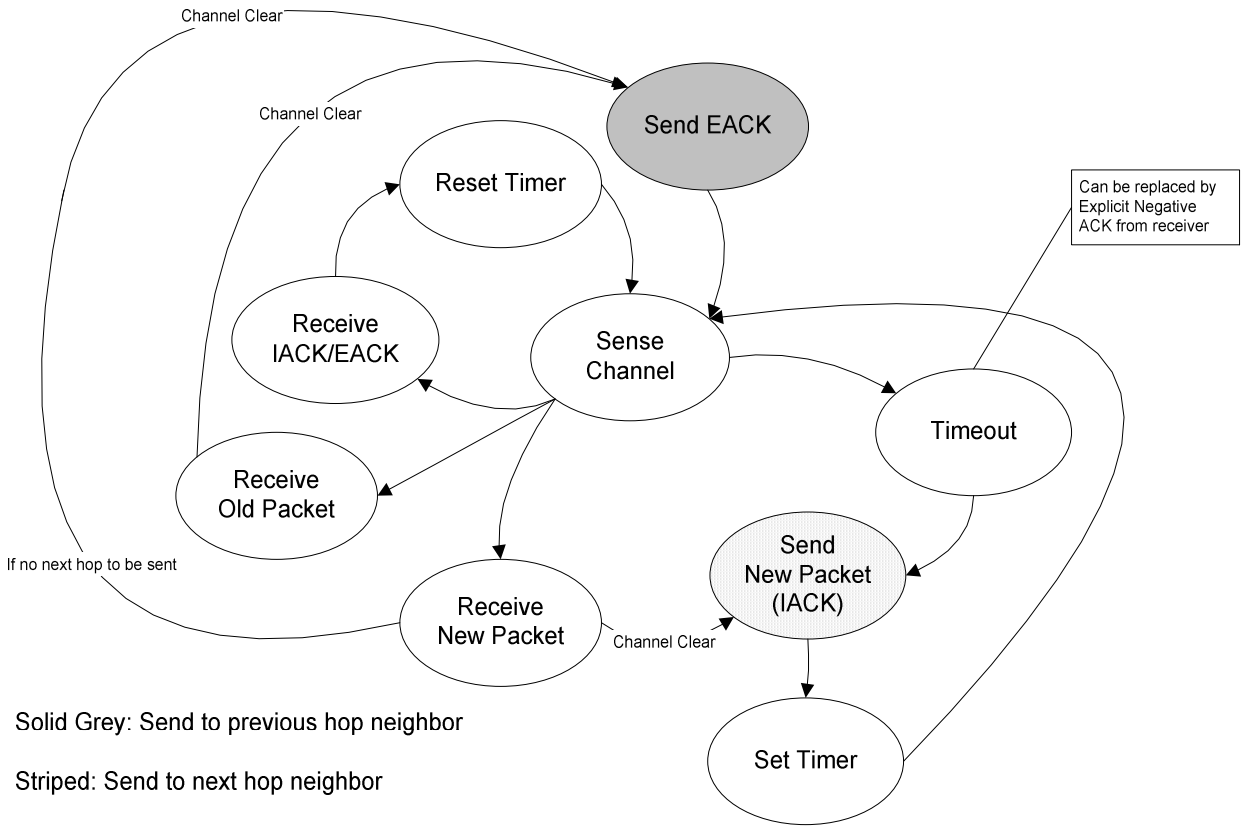

Figure 89 - State transition diagram for IACK/EACK scheme

Pseudocode for node operations (EACK only scheme):

```
variables:
type_to_send              - type of packet to send
address_to_send           - address of node to send packet
timeout                   - next timeout (sent_packet_list element)

sent_packet_list          - list of sent packets waiting for ACKs
send_packet_list          - list of packets waiting to be sent after channel clear
last_received_packet_ID   - ID of previously received packet
time_begin_send           - upcoming time to send next packet
time_end_receive          - time of end of receiving packet
time_end_send             - time of end of sending packet
channel_clear             - "Sense Carrier" state

last_received_packet_ID = -1


while (wait receive) {
      if received file is from time keeper {
            // advance time marker by 1
            current_time = time from time packet
            if (current_time == time_end_send) {
                  // finished sending
                  time_end_send = 0
                  if (time_end_receive >= time_end_send) {
```

```
                // still receiving packet
                // do nothing
        }
        else {
                // channel clear
                channel_clear = TRUE
        }
}
if current_time == time_end_receive {
        // packet received
        channel_clear = TRUE
        if ((packet_collision == FALSE) && (destination_ID == node_ID)) {
                // packet received successfully without packet collision
                // accept packet only after passing the BER test
                accept packet if probability = p
                if (packet accepted) {
                        // process packet
                        if (packet_type = C/D) {
                                // control/data packet received
                                // reply ACK
                                last_received_packet_ID = received_packet_ID
                                next_packet_ID = received_packet_ID
                                next_packet_type = ACK
                                next_packet_source_node = node_ID
                                next_packet_destination_node
                                                = received_packet_ID
                                next_packet_duration = ACK duration
                                time_end_send = current_time + ACK duration
                                send packet to every neighbor
                                channel_clear = FALSE

                                    if(received_packet_ID!=last_received_packet
                                        _ID) {
                                        // new packet received
                                        // construct next packet
                                        next_packet_ID = received_packet_ID + 1
                                        next_packet_type = C/D
                                        next_packet_source_node = node_ID
                                        next_packet_destination_node
                                        = next_neighbor_packet_ID
                                        next_packet_duration = C/D duration
                                        time_begin_send = time_end_send
                                        append packet to send_packet_list
                                        // timeout = C/D timeout
                                }
                        }
                        else if (packet_type = ACK) {
                                parse sent_packet_list
                                if (packet found with same packet_ID) {
                                        // correct ACK
                                        remove packet from sent_packet_list
                                        remove packet from send_packet_list
                                }
                        }
                }
        }
}
parse send_packet_list
if current_time == time_begin_send {
        if (channel_clear == TRUE) {
                // channel clear to send
                channel_clear = FALSE
                packet_timestamp = current_time
```

```
                        send new packet to every neighbor
                        time_end_send = current_time + packet_duration
                        if packet_type = C/D {
                              timeout = current_time + C/D timeout
                        }
                        place packet from send_packet_list to sent_packet_list
                  }
                  else {
                        // channel is busy, wait until next time marker
                        time_begin_send = current_time + 1
                  }
            }
            parse sent_packet_list
            if current_time == timeout  {
                  // timeout of packet previously sent
                  if (channel_clear == TRUE)
                        // resend packet
                        // channel clear to send
                        channel_clear = FALSE
                        packet_timestamp = current_time
                        send new packet to every neighbor
                        time_end_send = current_time + packet_duration
                        if packet_type = C/D {
                              timeout = current_time + C/D timeout
                        }
                        place packet from send_packet_list to sent_packet_list
                  }
                  else {
                        // channel is busy, wait until next time marker
                        time_begin_send = current_time + 1
                        place packet to send_packet_list
                        set timeout = -1
                  }
            }

            send “finished” packet back to time keeper
      }
      else if received packet is from neighbor {
            // incoming packet
            // channel not clear

            if (channel_clear == TRUE) {
                  // not sending or receiving other packets
                  // no channel collision yet
                  // append packet to buffer
                  channel_clear = FALSE
                  packet_collision = FALSE
                  time_end_receive = received_packet_timestamp
                                          + received packet_duration
            }
            else {
                  // currently sending or receiving other packets
                  // channel collision
                  packet_collision = TRUE
                  if(time_end_receive = received_packet_timestamp
                  + received packet_duration) {
                        // the newest packet will be received the latest in time
                        time_end_receive = received_packet_timestamp
                                                + received packet_duration
                  }
            }
      }
}
```

Pseudocode for node operations (IACK/EACK scheme)

```
variables:
type_to_send              - type of packet to send
address_to_send           - address of node to send packet
timeout                   - next timeout (sent_packet_list element)

sent_packet_list          - list of sent packets waiting for ACKs
send_packet_list          - list of packets waiting to be sent after channel clear
last_received_packet_ID   - ID of previously received packet
time_begin_send           - upcoming time to send next packet
time_end_receive          - time of end of receiving packet
time_end_send       - time of end of sending packet
channel_clear             - “Sense Carrier” state

last_received_packet_ID = -1


while (wait receive) {
       if received file is from time keeper {
              // advance time marker by 1
              current_time = time from time packet
              if (current_time == time_end_send) {
                     // finished sending
                     time_end_send = 0
                     if (time_end_receive >= time_end_send) {
                            // still receiving packet
                            // do nothing
                     }
                     else {
                            // channel clear
                            channel_clear = TRUE
                     }
              }
              if current_time == time_end_receive {
                     // packet received
                     channel_clear = TRUE
                     if ((packet_collision == FALSE) && (destination_ID == node_ID)) {
                            // packet received successfully without packet collision
                            // accept packet only after passing the BER test
                            accept packet if probability = p
                            if (packet accepted) {
                                   // process packet
                                   if (packet_type = C/D) {
                                          // control/data packet or IACK received
                                              if (received_packet_ID=last_received_packet
                                                  _ID +1){
                                                  // IACK received
                                                  parse sent_packet_list
                                                  if (packet found with same packet_ID+1)
{
                                                         // correct ACK
                                                         remove pkt from sent_packet_list
                                                         remove pkt from send_packet_list
                                                  }
                                          }
                                          else if (received_packet_ID >
                                                 last_received_packet_ID) {
                                                 // new packet received
                                                 // construct next packet
                                                 next_packet_ID = received_packet_ID + 1
```

```
                                        next_packet_type = C/D
                                        next_packet_source_node = node_ID
                                        next_packet_destination_node
                                                = next_neighbor_packet_ID
                                        next_packet_duration = C/D duration
                                        time_end_send = current_time + C/D
                                                duration
                                        send packet to every neighbor
                                        channel_clear = FALSE
                                        send packet to every neighbor
                                        timeout = current_time + C/D timeout
                                        place packet to sent_packet_list
                                }
                                else {
                                        // old packet
                                        // reply ACK
                                        last_received_packet_ID =
                                            received_packet_ID
                                        next_packet_ID = received_packet_ID
                                        next_packet_type = ACK
                                        next_packet_source_node = node_ID
                                        next_packet_destination_node
                                                = received_packet_ID
                                        next_packet_duration = ACK duration
                                        time_end_send = current_time + ACK
                                                                duration
                                        send packet to every neighbor
                                        channel_clear = FALSE
                                }
                        }
                        else if (packet_type = ACK) {
                                parse sent_packet_list
                                if (packet found with same packet_ID) {
                                        // correct ACK
                                        remove packet from sent_packet_list
                                        remove packet from send_packet_list
                                }
                        }
                }
        }
}
parse send_packet_list
if current_time == time_begin_send {
        if (channel_clear == TRUE) {
                // channel clear to send
                channel_clear = FALSE
                packet_timestamp = current_time
                send new packet to every neighbor
                time_end_send = current_time + packet_duration
                if packet_type = C/D {
                        timeout = current_time + C/D timeout
                }
                place packet from send_packet_list to sent_packet_list
        }
        else {
                // channel is busy, wait until next time marker
                time_begin_send = current_time + 1
        }
}
parse sent_packet_list
if current_time == timeout  {
        // timeout of packet previously sent
        if (channel_clear == TRUE)
```

```
                        // resend packet
                        // channel clear to send
                        channel_clear = FALSE
                        packet_timestamp = current_time
                        send new packet to every neighbor
                        time_end_send = current_time + packet_duration
                        if packet_type = C/D {
                               timeout = current_time + C/D timeout
                        }
                        place packet from send_packet_list to sent_packet_list
                  }
                  else {
                        // channel is busy, wait until next time marker
                        time_begin_send = current_time + 1
                        place packet to send_packet_list
                        set timeout = -1
                  }
            }

            send “finished” packet back to time keeper
      }
      else if received packet is from neighbor {
            // incoming packet
            // channel not clear

            if (channel_clear == TRUE) {
                  // not sending or receiving other packets
                  // no channel collision yet
                  // append packet to buffer
                  channel_clear = FALSE
                  packet_collision = FALSE
                  time_end_receive = received_packet_timestamp
                                          + received packet_duration
            }
            else {
                  // currently sending or receiving other packets
                  // channel collision
                  packet_collision = TRUE
                  if (time_end_receive = received_packet_timestamp
                                       + received packet_duration) {
                        // the newest packet will be received the latest in time
                        time_end_receive = received_packet_timestamp
                                                + received packet_duration
                  }
            }
      }
}
```