Packet Aggregation in TelosB WSNs: Design, Implementation and Experiments

by

Ayyaz Mahmood
Salekin Imran

Supervisor
Frank Yong Li

Department of Information and Communication Technology
Faculty of Engineering and Science
University of Agder
Norway

Grimstad, May 16, 2016

Abstract

WSN is an extensive field of research and a core technology which is adopted for monitoring and data assembling, used in various applications. Traditionally in a WSN, communication is performed in the fashion of single packet per transmission which produces high energy consumption and longer delay. Therefore, we introduce a novel approach by designing a system which applies packet aggregation in TelosB sensor motes using Contiki platform. In this approach, we assemble multiple packets together and send them in an aggregated frame towards the sink which can reduce the number of transmissions, energy consumption and delay per transmission unit. Accordingly, in this thesis report, we perform aggregation at relay node with diverse set of topologies and schemes and examined this technique through numerous experiments and testing for temporal and spatial aggregation topologies. Additionally, we design and implement three packet aggregation schemes and perform a comprehensive set of experiments to evaluate these schemes. These experiments are capable to demonstrate how packet aggregation proposes compelling performance boost in terms of less transmission count and how these schemes strongly actuate the energy level due to the reduction in transmission count. This report concludes with a brief summary of these experiments and recommends some potential approaches to affirm and further expand these outcomes. We have a belief that this research will be effective for better understanding of packet aggregation approach in WSN for future pioneers.

Keywords: *WSNs, Packet aggregation, Contiki, TelosB, Spatial and temporal, Delay, Energy consumption, Design and implementation.*

Preface

Entrance devotion and inspiration have always turned up as an essential part in the achievement of any project. This thesis is submitted in partial fulfilment of the requirements for module IKT-590 Master Thesis. This thesis has 30 ECTS and is done at the spring semester from January 4, 2016 to May 15, 2016. It is performed at the Department of Information and Communication Technology, University of Agder, campus Grimstad.

We impart our immense gratefulness to Professor Frank Y. Li for encouraging and guiding us to finish this project earlier. Additionally, we are grateful to Mr. César Asensio for his help in programming and feedback during fulfilment of this task.

Ayyaz Mahmood
Salekin Imran

Grimstad
May 16, 2016

Contents

List of Figures

List of Tables

List of Abbreviations

ACK	ACKnowledgement
AVR	Alf and Vegard's RISC processor
AODV	Ad-hoc On-demand Distance Vector
CDA	Concealed Data Aggregation
CH	Cluster Head
CCA	Clear Channel Assessment
CSMA/CA	Carrier Sense Multiple Access/ Collision Avoidance
CS	Compressive Sensing
CPDA	Cluster-based Private Data Aggregation
CDMA	Code Division Multiple Access
DC	Duty Cycling
DTMC	Discrete Time Markov Chain
DCF	Distributed Coordination Function
DIFS	DCF Inter Frame Space
DSR	Dynamic Source Routing
DSDV	Destination Sequenced Distance Vector
DSSS	Direct Sequence Spread Spectrum
FEC	Forward Error Correction
FDMA	Frequency Division Multiple Access
FIFO	First-In-First-Out
HTTP	Hyper Text Transfer Protocol
IoT	Internet of Things
IEEE	Institute of Electrical and Electronics Engineers
*i*PDA	Integrity Protecting Private Data Aggregation
IP	Internet Protocol
LEACH	Low-Energy Adaptive Clustering Hierarchy
LQI	Link Quality Index
LR-WPANs	Low-Rate Wireless Local Personal Area Networks
MAC	Medium Access Control
PHY	Physical
RSSI	Received Signal Strength Index
RN	Reference Node
RREQ	Route Request
RF	Radio Frequency
RX	Receiver
RAM	Random Access Memory

SMART	Slice Mix AggRegaTe
S-MAC	Synchronized Medium Access Control
SHIMMER	Sensing Health with Intelligence Modularity, Mobility and Experimental Re-usability
SPOT	Small Programmable Object Technology
TX	Transmitter
TCP	Transmission Control Protocol
USB	Universal Serial Bus
UDP	User Datagram Protocol
WSN	Wireless Sensor Network

Chapter 1

Introduction

WSN is a promising technology which consists of an enormous number of tiny sensors deployed in an ad-hoc fashion. Due to this fact, it has extensive dimensions in numerous applications such as tracking, remote monitoring, surveillance, etc. Therefore, it is becoming more adopted and fascinating field of research nowadays. More often, sensor nodes deployed in a WSN are narrow in terms of resources, for instance, energy, bandwidth, packet size, etc. Accordingly, sensors nodes need to cooperate inside a network in terms of data accumulation to decrease the number of transmission units to preserve these resources. This in-network accumulation technique is referred as packet aggregation. These resource parameters such as bandwidth, number of transmission units and energy consumption for each node along with defined topology protocol between sensor nodes have forthcoming effects over the network's performance.

This chapter contains background knowledge of this topic as with concise description of problem statement and what was the driving force which boosted us to embrace this topic.

1.1 Background and Motivation

In WSN, all the sensor nodes are used to perform monitoring and processing of the gathered data by transmitting this data towards the sink node. Afterwards, sink node collects the data from all nodes and follow the predefined protocol for data inspection to achieve the desired outcomes. Unlike ad-hoc networks where nodes follow any-to-any topology, WSN is always in the favour of many-to-one transmission pattern. Eventually, this many-to-one pattern causes energy and resource outflow. Therefore, all real world sensor deployment scenarios are energy-limited and resource constrained. The key concern in this context is how to enhance the energy conservation of these sensor nodes using in-network resources during the process of data collection.

To solve the energy consumption problem, several clustring methods and routing protocols are proposed which enhance the energy conservation for WSN. For instance, [1] proposes energy-efficient communication protocol for wireless micro-sensor networks which focuses on a clustering-based protocol, called LEACH. This protocol adopts the random circulation of CHs to uniformly assign the energy consignment among all the sensor nodes in a network. Data gathering algorithms in sensor networks using energy metrics is presented in [36] which is based on construction of $energy \times delay$ matrix and tries to create an equilibrium between energy and delay in WSN. However, adequate techniques for various deployment of protocols are needed to meet this task. Accordingly, there are many factors which has huge impact towards resource conservation of WSNs. In this manner, packet-aggregation is noticeable which is based on a principle that during monitoring and data gathering, relay nodes can execute packet-aggregation by combining and appending multiple packets from different nodes into one aggregated frame. Afterwards, this frame is transmitted towards the intended sink node in one transmission unit.

Some efficient algorithms to increase the life span of sensors by data gathering and aggregation is discussed in [2]. In [3], a design architecture for clustering and aggregation is presented where CH in each cluster gathers the data from all over the cluster and performs aggregation. During the process of aggregation, keeping data privacy is also a key factor which is resolved in [3] by providing a privacy-preserving data aggregation. It uses CPDA and SMART protocol by implementing algebraic characteristics of polynomials to execute this task. An iPDA, given in [4], is also related with data integrity where data slicing and assembling is performed in collaboration with disjoint aggregation paths building for data gathering to ensure integrity. Another efficient technique for data collection and aggregation is presented in [5] which explains a clustering-based approach to implement aggregation. Similarly, [6] considers an energy-aware spanning-tree algorithm for aggregation which is called E-Span. This algorithm is based on source configuration, when a root is elected from all the source nodes on the basis of highest residual energy while rest of all the nodes select their parent node on the basis of residual energy and distance towards root. An aggregation-tree is constructed in [7] which is helpful for data-centric routing by switching off the radio of leaf nodes and allowing no-leaf nodes in the topology to control the aggregation. Most of the algorithms adopt in-networking processing technique when new packets are appended with partially processed packets and forwarded towards the sink node. Furthermore, considerable amount of research is available which specifically or generally related with packet aggregation. In [8], another technique is introduced , called CS. This technique attains lowest sampling rate for each sparse sensed packet to decrease the transmission count.

Although these investigations brought many noticeable outcomes but these efforts are not enough as TelsoB based packet aggregation on Contiki platform is not done yet, to the best of our knowledge. This essential reason prompts our motivation towards real life implementation of packet aggregation in TelsoB sensor motes. Moreover, our intention is to address some of the lacking topics and also bring up some potential work for future research. Just as, previous investigations are somehow bounded by lack of knowledge about some constraints like transmission count, time per transmission, energy, delay, etc. while performing aggregation in real life. Therefore, we think these inspections are insufficient to combat this problem in real life as some specific algorithms are needed which can appropriately use all the resources with better performance in terms of transmission count and energy utilization. In our experiments, we built a network of multiple sensors which are implemented in a way that sources are sending data packets towards relay node which is behaving as an aggregator. After aggregation, relay node sends the aggregated packets towards the sink where sink can either store that information for further processing or it can simply discard that information. The key concern in this process is how efficiently data is collected and how flexibly it is aggregated. Moreover, how received data can be stored at relay node for a short duration of time and how to perform a homogeneous selection of data according to its service type. Our testing and experimental results explain the implication and significance of the given topologies and schemes over the preceding ones in order to improve the resource conservation. We hope that this research can attain a desirable attention and a broad concern from the research community in this definite field which can be helpful for the eventual expedition.

1.2 Problem Statement

This project contemplates the concept of packet aggregation in WSNs using TelosB sensor motes. It consider two topologies, accompanying with several schemes which are designed for packet aggregation in order to decrease the resource utilization. This thesis acknowledges following goals:

- *How to design an aggregation scheme which is more suitable to TelosB and single/multi-hop in order to scale down the number of transmission?*
- *How to implement aggregation at relay node with different parameters? Additionally, how to employ packet aggregation which do not comprises of high energy consumption and delay?*
- *How to evaluate packet aggregation by an adaptive testing and experimental approach to deliver specified outcomes?*

1.3 Approach

In almost every research stream, there are always certain limitations which become apparent while dealing with the implementation of new techniques. Likewise, WSN also suffers certain challenges when it comes to design and implementation. Some of the major problems regarding this area are:

- Precise synchronization between source relay and the sink node due to the processing delay generated by the packet aggregation in WSNs.
- Strict energy confinement and narrow in-network holdings of the sensors motes.
- Buffer and queue management for intended incoming and outgoing packets.
- Evaluation and compilation of multiple nodes simultaneously and assessment of the performance of each node analytically.

This thesis is mainly concerned about conservation of in-network resources for different nodes in the network topology. This whole idea is established by using some existing aggregation methods and proposes several schemes and scenarios in this environment. For instance, how to aggregate packets on the relay nodes using unicast transmission protocol. How to generate real time data while using temporal and spatial aggregation and what is the effect of this aggregation in terms of transmission count and energy level of each node. How to initialize sequence number and source ID in each packet in order to make the transmission more reliable and accurate.

In WSNs, several processes are happening simultaneously so every process needs to be selected homogeneously according to its service type. In our case, sensor nodes are generating real time data such as temperature, light, humidity etc. Hence, it is elemental to set s selective aggregation method for each data type to enhance the flexibility of the network. For attainment of these requirements, we use several methods and techniques to accomplish the prescribed tasks in specified time. Subsequently, going through all the research material, we approach the conclusion that in-network resources can be conserved by performing concatenation at relay node and select the data according to the requirements. In order to carry out the experiments, we start with simple HELLO messages and move towards the real time data. This real-time behaviour of a sensor node is very important in a way that it looks more convincing for any researcher as every physical development is followed by a research. Therefore, instead of sending raw data or any beacon message, real-time values are always preferable. Afterwards, we execute all the real time testing and experiments to produce some numerical results. The throughout system architecture revolves around certain points such as decrease the number of transmission to an optimum number, introduce less possible delay between each transmission while maintaining the QoS as there should not be any possibility of packet drop. Furthermore, every experiment is executed by assuming ideal conditions for the channel. In contrast with previously proposed schemes, our proposed packet aggregation schemes are more adaptable, reliable, future oriented and thus more appropriate for energy and resource-limited scenarios.

1.4 Report Organization

The rest of the report is enumerated in the following manner:

- The second chapter contains some background familiarity about the concept of packet aggregation, how it is being used in WSN, various techniques used for aggregation and some elemental information of TelosB motes and Contiki platform.
- The third chapter is composed of design for different topologies, platform building, hardware and software architectures for the implementation of each scheme and topology.
- The fourth chapter is related to the attained experimental results, analysis and compilations from different test scenarios and schemes.

- The fifth chapter is about discussion of the results, provided in chapter 4 along with a brief description of unsuccessful attempts to provide some potential basis for future research.
- Eventually, the sixth chapter is conclusions and also endorses some new dimensions as future works.

Chapter 2

Enabling Technologies and Related Work

In WSNs, sensors are deployed in large geographical areas and these networks are implemented in multi-hop fashion with certain resource constraints and limited sensing range. Therefore, instead of direct transmission, packets are sent through relay nodes. These relay nodes are also responsible for processing and aggregation. In order to conserve these resources, optimal number of transmissions in less possible transmission time is necessary. Given the fact that local calculation and processing bring less cost in terms of energy and time, packet aggregation in WSN has become an interesting field of research. It is due to the fact that packet aggregation is intended to extend the lifespan by excluding avoidable transmission units [33]. However, WSNs are always energy-limited and memory constrained, this aggregation technique sometime causes more complications and dilemma while doing inside the network.

This chapter contains of a brief knowledge about packet aggregation, multi-hop networks and related work done previously. Additionally, It explains about Contiki platform and TMote Sky sensors and their relation with our thesis.

2.1 Enabling Technologies

WSNs are appearing as one of the most promptly growing field of research involving dynamic platforms and support for various effective operations. However, several technical barriers are there which have to overcome to attain the required outcomes. This section compiles some trending features and configurations by concentrating on some basics of WSN and multi-hop networks as an highly compelling permissive technique for WSN.

2.1.1 WSN essentials

In sensor networks, all the sensor nodes are aligned to perform certain tasks such as monitoring, data collection, etc. Afterwards, collected data is transmitted towards the sink via some relays. Sink node is responsible for gathering all the monitored and processed data and ends up with final report [3]. An important feature in WSN is that a sensor mote has the ability to perform some tasks like fusion, correlation and processing/saving of data into its memory, in addition to its conventional communication feature. Moreover, sensors motes can communicate with other sensor motes from a different geographical area, as shown in Figure 2.1, using Internet via sink nodes [9].

In WSN's operations, sometime it does not require any predefined position or topology. Due to this fact, WSNs are sometime treated as certain form of wireless ad-hoc networks. WSN faces certain challenges in many cases, if a sensor node is already deployed, it should remain unattended. When a device is capable to adopt all the environmental and topological variations and behave according to these variations, its called self-managing. This self-management contains many tiny features like when a sensor node is equipped with an ability to configure its technical specifications like RSSI, transmission power, connectivity, LQI etc. If any change is noticeable in the system state, its called self-organization. Next

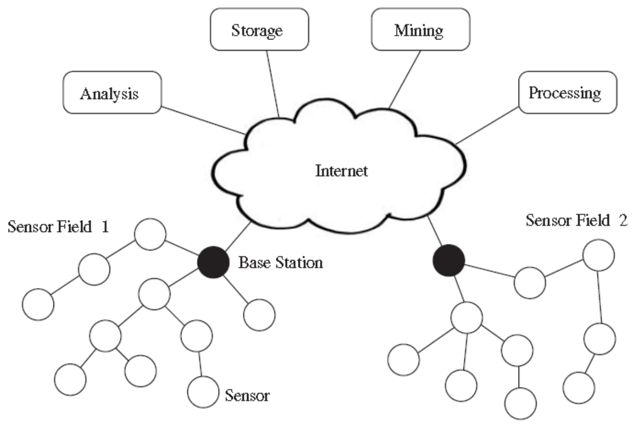


Figure 2.1: Communication between different sensor fields via sink nodes [9].

feature is called self-optimization, when a node is able to optimize its resources according to the requirement. In self-protection, a device can detect any possible threat and can recover from that attack. Lastly, in self-healing, a sensor node affords to find, detect and react to network interruptions [9]. In terms of communication, WSN nodes follow IEEE 802.15.4 protocol which is refereed for LR-WPANs, usually named as Zigbee which is a logical network. Zigbee is build on physical radio IEEE 802.15.4 standard and is used for networking in sensor devices, while deployed at remote locations [10]. In many cases, this standard apply frequency band of 2.4 GHz. This band has a indoor working range of 30 m and outdoor working range of 100 m while transmitting packets with the rate of 250 kbps.

Scalability and less power utilization are the key features for an optimal sensor mote. Additionally, it is able to gather data rapidly with great efficiency and consistency. Therefore, adoption of any sensor node before starting any task is very important to attain desirable outcomes [11]. The signal conditioning block is re-programmable and even alterable to accommodate different sensors inside the mote. Accordingly, radio link is also exchangeable, depends on the intended operation and requirement of the protocol. In memory section, flash memory is there to hold all the given instructions or gathered information for the short period of time. Moreover, an adaptive embedded firmware is used which follows the wireless network for the updates. The micro-controller is responsible for data employment, energy management, sensing to physical layer consolidation and organisation of radio network protocol. However, energy management in WSN is not an easy task while dealing with uncertain events. Therefore, an event-sensitive approach with appropriate hardware architecture is needed to manage the required energy consumption [11]. For the network architecture, various topologies are implemented in WSNs including, star network, mesh network, hybrid star-mesh network, etc. which will be discussed in the coming sections.

2.1.2 Multi-hop networks

As discussed earlier, sensor nodes are typically programmed to transmit data packets towards the sink node which behaves as a gateway between other networks. In a simple topology, when sender and sink node are close enough to each other that sender can directly send the data to sink without requiring any relay node, this type of network is usually referred as single-hop network. Nonetheless, when nodes are deployed in large geographical areas and nodes are far away from each other, single-hop communication is not suitable. This type of scenario is suitable for multi-hop network which is designed with multiple sensor nodes, behaving as relay, between sender and sink node. These relay nodes perform monitoring and collection of their own data and also forwarding of other nodes data simultaneously.

In Figure 2.2, single-hop and multi-hop models are shown where in single-hop (left), all the sensors nodes are attached with the sink node without requiring any relay node. Therefore, communication be-

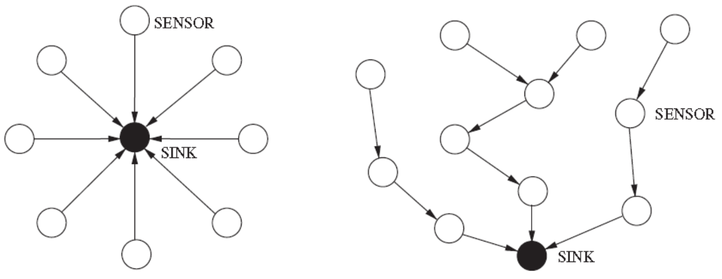


Figure 2.2: Single-hop vs multi-hop communication [9].

tween all nodes to sink node is direct. In contrast, nodes are indirectly sending data to sink node via some relay node, as shown in Figure 2.2 (right), by following a routing path to the sink node. The key challenge in this design is to determine this path despite the fact that WSN has energy constrain and memory limitations. Accordingly, in case of any variations in topology, these routing paths should be volatile and alterable. However, global addressing schemes like IP addressing etc. are not feasible to implement in some scenarios so WSN has its own routing protocols which are categorized in terms of network organisation, route discovery and protocol operation [9].

While we are talking about the identifying the routing path, the most commonly used approach is minimum hop, that is, finding the shortest possible path from source to sink by using smallest possible hops. Here, every link between the two nodes has a fixed cost and shortest path is selected by estimating the total aggregated cost of all the links till the destination. This approach eventually decrease delay and energy consumption by involving less possible relays but may not be ideal in terms of delay and energy optimization. Moreover, in terms of energy efficiency, several estimations are there which consist of minimum energy consumed per packet, maximum time to network partition, minimum variance in node power levels, maximum (average) energy capacity and maximum minimum energy capacity [9].

2.1.3 Packet aggregation

In WSN, nodes are energy-constrained and resource-limited and in the process of monitoring and gathering data, each node has a data packet which is suppose to be sent to the sink. Packet aggregation in this context is used to reduce the protocol overhead by appending multiple packets together and send them in a single transmission towards the sink which will reduce the number of transmissions and overhead, associated with each transmission unit. Aggregation schemes depends on the topology of every network. For instance, in cluster based networks, aggregation is usually performed at CHs while inside a cluster, relay nodes behave as aggregators [13]. In [11] some fundamental features are presented which include latency and energy consumption. At first, latency is actually a duration required to finish the aggregation between aggregator and the source node. It includes distance between nodes, packet size and channel state. CH follows the multi-hop pattern by identifying the shortest possible path towards the destination. Therefore, to decrease latency, backup paths are followed for aggregation purpose. Accordingly, in working mode of data aggregation, aggregator waits for all the data coming from various sources to execute the aggregation. Accordingly, the cut off for residual energy is examined regularly to elect the new aggregator with maximum possible energy within one cluster.

Heretofore, a lot of research has been done in the context of packet aggregation in WSNs by focusing on different aspects of aggregation, for instance, data gathering algorithms using energy metrics in [36], energy aware data aggregation in [12], energy efficient recoverable concealed data aggregation in [13], etc. In this section, we are going to briefly analyse some of the techniques and schemes about aggregation, proposed by academia. Additionally, there are some techniques which not only discuss aggregation but

add more value by providing some favourable algorithms like packet-compression, encryption, com-

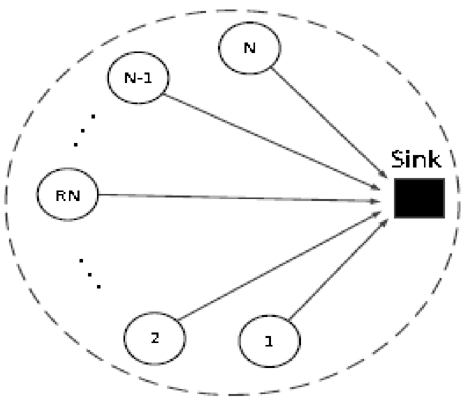


Figure 2.3: Network model for DC aggregated transmission [14].

pressed sensing, etc.

2.2 Related work

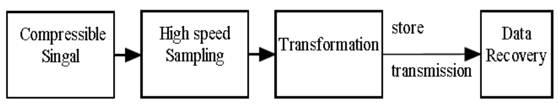
This section consists of a encyclopaedic summary of some of proposed techniques and schemes by various researchers. We are going to explain some of those techniques to give a brief knowledge to readers about the the efforts which are already presented.

Packet aggregation in Duty-Cycled WSNs

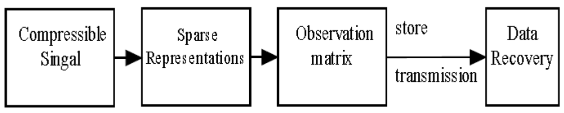
As DC is a well known technique which involves wake up and sleep intervals for sensor nodes to save energy. In [14], instead of defining a duty cycle for an individual packet, it is preferable to deploy a duty cycle for an array of aggregated data packets to save time and energy. Furthermore, [14] considers a three-dimensional DTMC model which includes a retransmission mechanism to analyse the transmission of this aggregated array of packets with respect to state of queue, retransmission and progression of alive nodes in a network. This approach utilizes S-MAC as its MAC protocol. The network model for this approach is described in Figure 2.3, where a cluster of N sensor nodes is examined and sink is one hop away from each node in the cluster. In operation, algorithm chooses one RN node forthwith and assumes that sink behaves as a receiver only. One node in the cluster operates on FIFO and acts as a buffer. Accordingly, the node analyses the collected packets in buffer executes aggregation process and builds a frame. This frame can only be transferred by the RN if it achieves success in competition to access the medium. Eventually, transmission for all the packets in frame is only possible if the number of gathered packets in buffer is less than maximal capacity *F* of frame. In contrast, frame with only *F* packets is sent. In case of fruitful transmission , packets in the buffer are downsized according to the frame capacity.

Efficient aggregation scheme to maximize WSN life span

An adequate energy-aware scheme is presented in [2] where a network is formed by implementing multiple sensors which are systematically gathering information and forwarding this data towards the sink. At the same time, these sensors are behaving as aggregators due to their capability of executing data aggregation within the network. Consequently, this aggregation process decrease the time required to send data from source till destination which is referred in [2] as sensor's life span. In the described technique, called MLDA algorithm, a nearly ideal polynomial time algorithm is presented with some clustered based advance technique to enhance the lifespan. After all the experiments, results showed that MLDA perform 1.15 - 2.32 times better than the current aggregation schemes in limited area networks. Additionally, clustered based approach gained 2.61 times better efficiency in life span than the typical ones.



(a) Typical compression



(b) Compressed sensing

Figure 2.4: Packet compression vs compressed sensing [8].

Energy-efficient data aggregation

Many techniques and algorithms have been presented to conserve energy and resources in WSNs. In this section, three techniques are described to decrease the energy consumption in WSNs. In [15], such an approach is given which proposes a framework, acts as a middleware for data aggregation inside a network in SPIN. It also shows a method to send minimal data with less possible error. In this scenario, nodes are forwarding packets towards sink, at first, there is broadcast message about packets from *A* for all the neighbouring nodes and on reception of request from *B*, packets are transmitted towards the node *B* which is intended to receive the packets. On the basis of minimum hop count, node *A* identifies the shortest path towards *B*. When packet arrives at *B*, algorithm matches the received packet with the packets formulated by *B* itself. If the packets are unchanged, received packets are discarded. Otherwise, received packets are aggregated with *B*'s packets and forwarded towards the next node. At next destination, which in this case is *C*, data is matched with the earlier packets from all nodes. Therefore, in case of real time data like temperature, humidity, light, etc. unchanged values are useless to consider which saves energy and time required for further processing. Consequently, statistics revealed that 750 nJ energy can be conserved at 4th round and 400 nJ is amount of minimal energy conserved at 2nd round. Another important aspect of energy-aware aggregation is discussed in [12] where an aggregation architecture is presented. This scheme proposes more flexibility in aggregation process by saving packets in RAM memory of a sensor node. This algorithm performs aggregation in certain conditions which are given below:

- If sensor's memory is packed and there is no space for incoming samples.
- If the saved packets amount exceed the limit of standard payload size of the intended outgoing packet.
- If any node requests to send data towards it.

Additionally this model provides an error correction algorithm FEC to control packet error ratio. In sensor's memory management, various temperature samples are being stored for short term before further processing. Additionally, a counter is introduced which counts each added sample into the database and computes the capacity of payload to accommodate these samples for aggregation. Thereupon, results reveal that proposed algorithm decreases the network packet loss ratio down to 90 % and enhances the life span upto 50% which are considerable ratios to boost the network performance. CDA is another approach discussed in [13] which introduces encryption and recover ability in data packets to save networks from external attacks. This recover ability factor provides authentication and integrity but instead of typical flexible implementation, this scheme sends diverse packets towards the CHs so these are responsible for data aggregation by providing more secure transmission through CDA.

This scheme saves energy by bypassing the redundant data while sending concurrently. The network model is distributed among various clusters while each cluster has *w* nodes. One node form all *w* nodes is elected as CH which is responsible for processing and aggregation of received data of all nodes from that cluster. Data transmission throughout this scheme is made confidential by applying *mykletun et al.'s* encryption. Therefore, redundancy is removed which results a noticeable decline in consumed energy. Furthermore, another clustered-based approach for aggregation is presented in [6] which considered E-Span as a protocol to allows to elect a root node among all nodes with higher residual energy. Remaining nodes decide their parent nodes from neighbourhood on the basis of residual energy and distance towards root. Therefore, instead of directed diffusion, this model helps to enhance the average life time of a node. In [11] presents an other idea of distributing aggregation protocols on the basis of structure, search-based and time based manners instead of structure free which also enhance efficiency and energy-conservation.

Compression and aggregation

Compression is another technique which is used to reduce overhead, produced by large amount of data travelling. Normally, compression is done after gathering of monitored data from sensor nodes to reduce the size of data packet. However, [8] introduces a different technique which is called CS which operates on less sampling rate for infrequent signals. In this context, signals are reconstructed in energy-conserved way where data is sent in narrow compressed way rather than transmission of whole batch of measured data towards the base station. In terms of aggregation, when CS is implemented, size of collected data at relay node is compressed apart from detaching any information. In Figure 2.4, a comparison between traditional compression and CS is shown as in typical compression, Figure 2.4a, N samples are needed to calculate the pack of transform coefficients and in case of CS, one signal is pitched continuously at low dimensional space to reconstruct the high probability signal with large dimension, illustrated in 2.4b. Eventually, due to the absence of sensing signal, traffic overhead comes down with decrease in the size of packets which is beneficial for energy conservation. By the same token, another highly effective and reliable minimum-energy compressed data aggregation scheme with CS is conferred in [41], which implements diffusion wavelets to discriminate the spatial correlations for reliable recovery of data. From the analysis of the outcomes, it is revealed that highly reliable recovery of data is possible by the appropriate plotting of inadequate basis which is an energy efficient way for data gathering.

2.2.1 MAC protocols in WSNs

In WSNs, numerous nodes are trying to transmit their packets. MAC protocol allows nodes to acquire the common medium for transmission. Accordingly, it is duty of the data link layer to develop a system which grant this access among all nodes evenly. MAC layer is a sub layer of data link layer an its responsibility is to organise a fair competition between all competing nodes to avoid contention in network. MAC protocols are further dispersed in contention free and contention based protocols. In contention free protocols, different techniques are used to grant access to only one node in the meantime. In contradiction, other category allows nodes to acquire the medium concurrently while assuring the fact that environment is collision free [9]. In some protocols where features of both categories are common are referred as hybrid protocols. In contention free, several protocols are used to decrease collision probability among different nodes. These protocols are categorised on the basis of their division in frequency, time etc. From [9], it yields a short description of these protocols, given below.

- ***FDMA*** splits the frequency band into multiple small bands which are allotted in a way that each band is given to each transmission between a pair of nodes.
- ***TDMA*** breaks the time into small slots for different nodes while using same frequency band. One time slot is assigned to utmost one node.
- ***CDMA*** grants concurrent access to multiple nodes by adopting several codes. In case of orthogonal codes, multiple nodes can communicate using same frequency band at the same time slot where FEC is used at receiver end to avoid interference.

On the other side, contention based protocols accord nodes to compete with each other to access the medium concurrently. However, it provides some liberty to reduce collisions or even recover ability from these collisions. These protocols include ALOHA, slotted- ALOHA, Carrier Sense Multiple Access (CSMA) etc. which are used for WSNs.

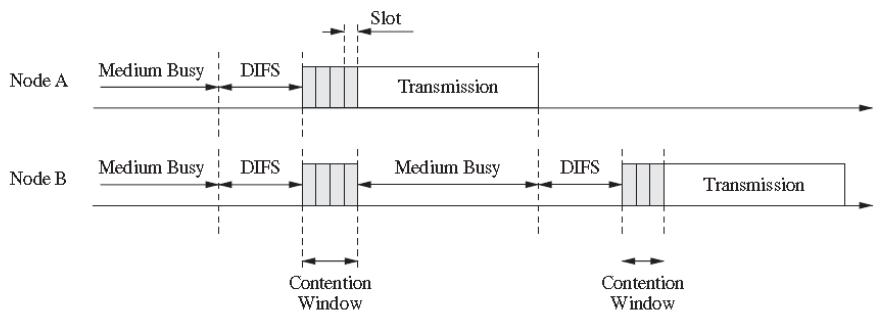


Figure 2.5: CSMA/CA mechanism for medium access [9].

CSMA/CA

CSMA allows node to sense the medium before further access which is feasible to avoid collision. Medium is continuously sensed by all nodes whether it is idle or busy. If medium is idle, a node intended to transmit packet, send instantly. Additionally, CSMA/CA is used to enhance the ability of a network to evade from collisions. Accordingly, when a node wants to transmit, first, it has to check whether channel is idle for DIFS and an interval of random back-off plus multiple of slot size. In this back-off algorithm, shown in Figure 2.5, a node with less back-off value wins the competition, as *A* has delay of $DIFS + 4 \times slotsize$ and *B* has $DIFS + 7 \times slotsize$. Therefore, *A* wins the competition and is allowed to send its packet while *B* has to wait till the next competition in addition to the interval of DIFS [9]. In every designed protocol for medium access, contention window size is an important factor, as if the values selected by various nodes are approaching to same size, collisions will occur. On the other side, if values are too large, it will bring unnecessary delay in transmission. Back-off timer is used in this context to introduce fairness in CSMA/CA. When a node selects a random time in contention window and it loses the first competition, it freezes its back-off counter and waits for one DIFS interval to take part in next competition. When back-off timer expires, it grabs the channel. Therefore, rejected nodes in first competition will not choose any random back-off value in next turn rather they will count down there timers. In this context, they only have to wait for their remaining slots to get access to the medium [16].

802.15.4 / ZigBee

The IEEE 802.15.4/ZigBee is a standard which is established to embrace to CSMA/CA as medium access approach to deal with the requirements of Personal Area Networks (PAN) with supported data rates of 20, 40 and 250 kbps. ZigBee standard is more divided into two sub topologies which are called star and peer-to-peer. In the first topology, PAN coordinator is responsible for every kind of transmission. However, peer-to-peer scheme offers liberty to communicate without any limitation after getting permission from the PAN coordinator [9]. The IEEE 802.15.4 consists of three characteristics which are number of back-off, content counter length. At first stage, back-off is at zero and it is only incremented by one if channel is found busy by CCA. However, there are always some constraints to keep back-off inside certain limit to abstain from overhead. Moreover, in IEEE 802.15.4, CSMA/CA is distinct in a way that back-off counter decrements the value without bothering the channel state [17].

2.2.2 Addressing in WSNs

In WSNs, instead of using typical global addressing approaches like IP addresses, etc. there are different ways to address nodes via several routing protocols.Another research from [9], describes various routing methods, based on design, working and path finding, etc. used in WSNs. However, we are organising protocols on the basis of proactive and reactive behaviour. Reactive routing protocols are on-demand protocols which are used when there is a need of transmission between a sender and a receiver without any predefined path between them. Protocol overhead is less as there is no existence of predetermined routes. Therefore, flooding is used to renew the route information if its required. Route discovery procedure is carried out through RREQ packets. The DSR and AODV are the two basic types of reactive protocols.

Proactive routing protocols discover the path from source till destination before physical transmission, which produces unnecessary delay in network. In proactive approach, every node has its own routing table which contains the route information towards all other nodes in the network. This information contains destination address, number of hops, etc. Each routing information in the table is remembered with a sequence number, generated by the destination. The DSDV is an example of this approach. Furthermore, if packets are transmitted from source towards destination, this categorical routing is known as node-centric routing like unicast, multicast or broadcast. However, in data-centric routing, when there is no catogorical transmission but destination is defined by certain parameters. For instance, if destination is asked to send only humidity then, instead of all sensors, only respective nodes are concerned in this case [9].

2.3 Adopted Tools

WSNs have extensive dimensions from simple monitoring and data gathering to packet aggregation and further towards more advance fields like IoT. Therefore, variety of tools and architectures are used to meet the requirements of these areas. This section demonstrates about different sensor motes, their classification and also shows a brief overview about Contiki-OS and rime stack.

2.3.1 Sensor motes

Wireless sensor motes are organised on the basis of storage, energy consumption, computation ability, communication protocols, etc. Given below, sensor mote classification is provided on the basis of processing power, cost, support, compatibility and size from [18].

- **Mica2/Micaz** are the second and third generation mote technologies from *CrossBow Technologies* which adopt similar power/battery composition like TelosB/TMote sky sensors.
- **TelosB/TMote** Sky mote are developed by *University of California, Berkeley* and are currently available from *MEMSIC Inc.* Furthermore, these motes require a pair of independent AA batteries with voltage range of 2.1 V to 3.6 V.
- **SHIMMER** requires 250 mAh power source at input.
- **IRIS** is the latest mote from *Crossbow Technologies* which also required a pair of AA batteries as a power supply.
- **Sun SPOT** is from *Sun Micro Systems* which are power-driven by a unified on-board rechargeable source.
- **Waspmote** is an open-source hardware and software based sensor platform from *Libelium Communications* which utilizes Li-Ion power, solar panels or USB as power sources.

For this thesis work, we decided to work with TelosB platform to execute our required experiments, details about TelosB is given in Section 2.3.2. Admitting the fact that we have many other options to choose but these nodes are selected as these were conveniently available. Therefore, to avoid long delays from ordering to shipping, we chose TMote by considering the short time span of this semester.

2.3.2 TelosB/Sky mote

TelosB mote, shown in Figure 2.6, is an ultra low power wireless module which is being used as an experimental device in WSNs. Essentially, it has the ability to monitor various real life events as it has integrated humidity, temperature and light sensors. Additionally, detailed characteristics [1] are given in following part .

- IEEE 802.15.4 enabled
- Data rate of 250 kbps
- TI MSP430 Micro controller with 10 KB RAM
- Unified on-board antenna
- Data gathering and programming by way of USB interface
- Open-source operating system
- Unified temperature, light and humidity sensors

This module affords small power consumption which allows batteries to be alive for longer duration likewise fast switching between wake up and sleep states.

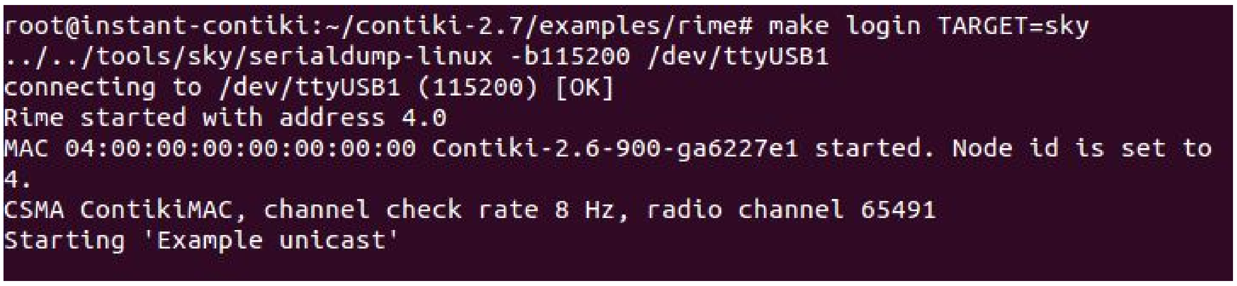
CC2420 radio

TelosB mote has the Chipcon CC2420 radio which is an IEEE 802.15.4 enabled radio, affording the PHY and some MAC functions for wireless communication. These distinct features afford reliable wireless communication. Moreover, it is highly compatible with other application using universal radio configuration. The CC2420 is MSP430 micro-controller driven module which operates using the SPI port along with series of digital I/O lines and interrupts. Some of the key features of CC2420 are given below [19].

- The sole-chip 2.4 GHz IEEE 802.15.4 enabled RF transceiver with baseband modem and MAC support.
- DSSS baseband modem with 2 MChips/s and 250 kbps data rate.
- Capability of working with both RFD and FFD.
- Less current consumption (RX: 18.8 mA, TX: 17.4 mA)
- Optimal supply voltage (2.1 – 3.6 V) with unified voltage regulator.
- Optimal supply voltage (1.6 – 2.0 V) with external voltage regulator.

[1] www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

Figure 2.6: TelosB sensor mote [20].



```
root@instant-contiki:~/contiki-2.7/examples/rime# make login TARGET=sky
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB1
connecting to /dev/ttyUSB1 (115200) [OK]
Rime started with address 4.0
MAC 04:00:00:00:00:00:00:00 Contiki-2.6-900-ga6227e1 started. Node id is set to
4.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
Starting 'Example unicast'
```

Figure 2.7: Initialization of a simple unicast programme in Contiki with predefined configuration.

2.3.3 Contiki-OS and its features

In WSNs, the gigantic task is to discover balanced and reliable architecture that will furnish bulky and excessive compilations in a smooth and swift environment while remaining inside the circumference of software and hardware boundaries. Contiki is such a platform which is developed to afford this liberty to execute dynamic loading and unloading of codes smoothly. Contiki is established with an event-driven kernel which accommodates selective preventive multi-threading for each exclusive process. It is established on C language which supports multiple environments to configure several micro controller architectures containing MSP430, Atmel AVR, now based on ESB platform. The ESB adopts MSP430 with 2kb RAM and 60 kb ROM, operating at 1 MHz. The MSP430 is capable of performing optional reprogramming of on-chip flash storage [2]. Figure 2.7 shows the initialization of program on Contiki before any operation.

TinyOS vs Contiki

In terms of operation, both TinyOS and Contiki are used to program wireless sensor nodes. However, there are apparent diversities which are essential to understand while dealing with various tasks in WSNs. From another research [9] describes this differentiation in terms of functional and non-functional features. Functional features include programming archetype, scheduling, building blocks, memory allocation and system calls. Both, TinyOS and Contiki have event-based paradigm but Contiki has an optional multi-threading ability. In block architecture, TinyOs has components, interfaces, and tasks while Contiki affords services, service interface stubs, and service layer. *FIFO* is used as a scheduling approach in both platforms. The major difference appears in memory allocation where TinyOS has static memory where Contiki has dynamic memory. Finally, system calls are not available in TinyOS. On the other hand, Contiki offers runtime libraries to call upon in runtime environment. Furthermore, non-functional features contain minimum system overhead, dynamic reprogramming, portability, etc. System overhead in TinyOs case is 332 bytes which is less than in Contiki which injects 810 bytes of data as system overhead. Dynamic reprogramming is fully supported in Contiki unlike TinyOS which requires external software to perform reprogramming [9].

[2]www.ti.com/lit/ds/symlink/cc2420.pdf/

Contiki features

Some of the key features of Contiki are illustrated below [32].

- **Memory allocation** is an efficient way to provide a mechanism for memory allocation.
- **Full IP networking** affords a full IP network stack, with standard IP protocols such as UDP, TCP and HTTP.
- **Dynamic module loading** dynamic loading and linking of modules at run-time is supported.
- **Power awareness** operates in extremely low power systems which are designed to run for years on a pair of AA batteries.
- **The Cooja network simulator** makes simulation tremendously easier by providing a simulation environment.
- **The Rime Stack** supports simple operations such as sending a message to all neighbours or to a specified neighbour, as well as more complex mechanisms such as network flooding and address-free multi-hop semi-reliable scalable data collection.
- **The Contiki shell** provides an optional command line shell with a set of commands that are useful during development and debugging of Contiki systems.
- **Protothreads** is a mixture of the event driven and the multi-threaded programming mechanisms. With protothreads, event handlers can be made to block, waiting for events to occur.

A running Contiki system consists of the kernel, libraries, the program loader, and a set of processes. In this context, a process can be an application program or a service, while a service implements functionality used by more than one application process. All processes, can be dynamically replaced at run-time. Communication between processes always goes through the kernel. The kernel is not able to afford a hardware abstraction layer but allows device drivers and applications communicate directly with the hardware. A process is defined by an event handler function and an optional poll handler function. The process state is held in the process's private memory and the kernel only keeps a pointer to the process state. On the ESB platform, the process state consists of 23 Bytes. All processes share the same address space and do not run in different protection domains [21].

In this chapter, we elucidated the basis of packet aggregation in WSN in order to improve energy efficiency and resource conservation. Additionally, we studied multi-hop networks and took an overview of some previously proposed methods and techniques to implement packet aggregation in context of WSN. Later, we explained our adopted tools including TelosB and Contiki and how these are related with our thesis work in order to achieve the desired outcomes according to the requirement specifications. We hope that after reading this chapter, one can easily understand the theoretical and factual background behind this thesis work.

Chapter 3

Network Design and Protocol Implementation

Wireless sensor nodes have several sub-functions such as sensing, processing, communication and energy distribution. Therefore, in order to assess the efficiency of packet aggregation, it is an important task to answer how to build system architecture, how to put together sensor nodes into a unified topology to execute all the sub-functions appropriately. Furthermore, how to design the protocol to perform these tasks according to the requirement specification. However, there are certain challenges for instance low data rates, narrow payload size, and crappy packet exchange performance, etc. Thus, to deal with all these challenges, a systematic approach is required which will provide a set of guidelines about design and protocol considerations and afford a framework for their implementation.

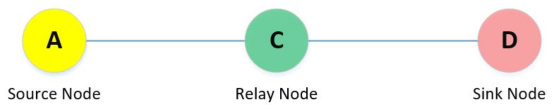
In this chapter, we propose several network topologies with programming layout which explains how protocols are designed and how well these are implemented, in order to assign packet aggregation in typical sensor networks. Additionally, these topologies are composed to assess the efficiency of packet aggregation in TelosB motes with respect to different schemes and algorithms.

3.1 Network Design

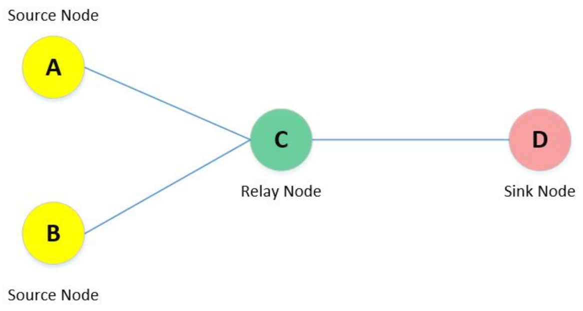
In terms of network design, we propose two basic topologies which are systematized as spatial and temporal aggregations. Spatial aggregation topology is employed when we are sending data packets from two different sources towards the aggregator where in temporal characterization, we analyse series of consecutive packets from a single source to the aggregator. The idea behind these topologies is to implement packet aggregation with different schemes and evaluate the impact of aggregation on sensor nodes against delay, number of transmissions and energy consumption. In order to perform this evaluation, we develop three aggregation enabled schemes which contain homogeneous aggregation, heterogeneous aggregation and variable frame-length aggregation. Moreover, the entire set of schemes include sequence number and ACKs for reliability and time-stamp to measure end-to-end delay. Accordingly, these parameters like end-to-end delay, number of transmissions and energy consumption are examined for the proposed schemes with aggregation in comparison to the schemes without aggregation. In this section, we provide a brief description of the proposed network model along several schemes to implement aggregation. Throughout the design and implementation, we used intermediate or relay as interchangeable name for a same node.

3.1.1 Network topologies

Consider a WSN cluster which contains multiple sensor nodes. These nodes are connected with each other and source communicate with other nodes via single or multiple hops. Nonetheless, nodes have some limitations in energy, memory, computation and processing. According to the topologies shown in Figure 3.1, node A and B are behaving as source nodes where C is operating as a relay node or aggregator. Node C executes packet aggregation based on the packets acquired in its buffer. Accordingly, the batch of



(a) Network topology for temporal aggregation.



(b) Network topology for spatial aggregation.

Figure 3.1: Network toplogies for temporal and spatial aggregation.

these concatenated packets is mentioned as a *frame*. Typically, transmission of this frame is only possible if there are at least two packets, aggregated in its payload. However, we also introduce another scheme which is referred to as variable aggregation where C periodically sends a frame without any packet-length limitations, explained in coming Sections. Node D is intended destination or sink which receives this frame. The wireless channels among these nodes are considered to be ideal and all the communication links between adjacent nodes are comparable. We start with simple temporal aggregation scheme, shown in Figure 3.1a, with three nodes A, C and D. The idea behind this scheme that A generates consecutive packets and transmits towards node C for aggregation and waits for specified time interval to send next packet. The relay node C receive and store the incoming packet in its memory and for duration and waits for next packet from A. After the predefined interval, A sends next packet towards C while C receives this packet, concatenates two packets into one frame and forwards this frame towards node D. However, in spatial aggregation, we have two sources A and B and both are generating real-time data. In this scheme, shown in Figure 3.1b A and B send packets towards C with difference in time slot to avoid collisions. It is important to mention that as buffer is very limited in size so some packets may be dropped because of buffer overflow. In our proposed network topologies, there are certain assumptions which are important to mention here. i) The channel is assumed to be ideal and error free such as channel state is certain and there is no interference or noise. ii) While calculating end-to-end delay, distance between all nodes is kept small enough to ignore propagation delay iii)Queuing delay is ignored because C can only receive one packet at a time. For certainty, the variable aggregation scheme is described in Algorithm 3 which demonstrates the behaviour of node C. In this algorithm, a variable aggregation scheme is illustrated as value of *i* is zero. After receiving first packet, algorithm inspects the previous state if there is any packet which is already received in *c[i]* . If there is any received packet, it simply concatenates the recently received packet with the previous packet and transmits towards the sink. However, if the memory of *c[i]* is blank then the incoming packet is stored in the memory and counter is incremented by one. The Algorithm 3 holds for both topologies, shown in Figure 3.1a and Figure 3.1b, as node C behaves identical in both cases.

3.2 Design of Packet Aggregation Schemes

In order to explain aggregation in TelosB sensor nodes, we proposed several schemes. Therefore, in this section, we discuss the design of different schemes for packet aggregation which are implemented and tested on TelosB motes. These schemes are organised with respect to spatial and temporal aggregation

Algorithm 1 Homogeneous aggregation

```
1:  i = 0
2:  while A packet is received do
3:      if The packet type == Seclected type then
4:          d[i] ← Received packet
5:          if i == 1 then
6:              e ← d[i] + d[i-1]
7:              Forward the aggregated frame from e
8:          else
9:              f[j] ← Received packet
10:             Forward the non-aggregated packet from f
11:         end if
12:         i = i+1
13:     end if
14: end while
```

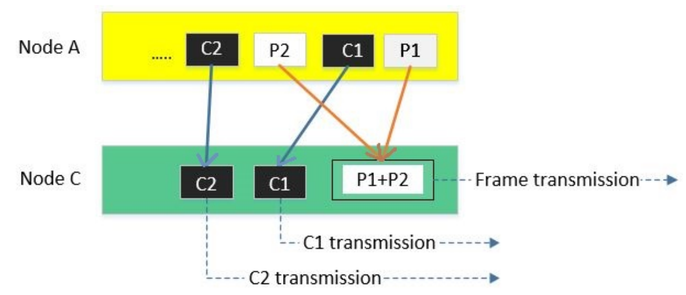
modes. A summary of all the proposed schemes is given in Table 3.1.

Table 3.1: An overview of the proposed schemes with respect to temporal and spatial aggregation with respect to source

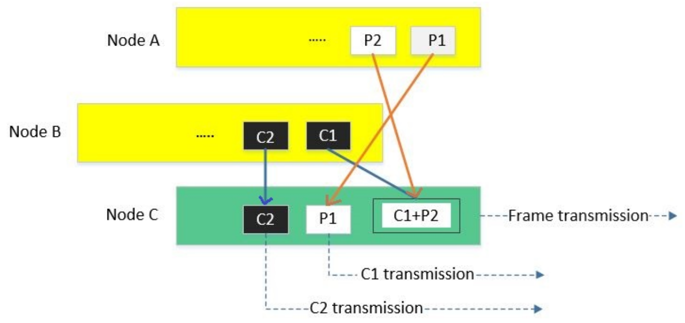
Proposed schemes	Temporal aggregation	Spatial Aggregation	Frame size
Homogeneous	*One source with two different data types*	*Two sources with two different data types*	*Fixed*
Heterogeneous	*One source with two different data types*	*Two sources with two different data types*	*Fixed*
Fixed frame-length	*One source with two different data types*	*Two sources with two different data types*	*Fixed*
Aaptive frame-length	*One source with three different data types*	*Three sources with three different data types*	*Adaptive*

3.2.1 Homogeneous aggregation

Apparently, the homogeneous aggregation involves a scenario where only packets from the same class are operated and their operation can not be compromised to the one with other classes. As sensor nodes are mostly used for monitoring and surveillance purpose, security and safety can not be compensated with any other task. Therefore, in addition to the typical aggregation, this scheme is proposed to afford more flexibility in aggregation as this approach offers selective packet aggregation at relay node. This selective packet aggregation is used to provide priority to the most critical information. While the rest of information is simply forwarded without performing aggregation. In Algorithm 1, design principle is more obvious as when packet is received at relay node, its priority level is checked by the protocol and if it is highly prioritized the it is stored in *d*. Afterwards, if there are already two packets in the memory, both of these packets are aggregated into a frame, stored in *e* and transmitted. However, in case of rest of the classes, packets are simply stored in *f[j]* and forwarded without further processing. This proposal is useful to distinguish between high and low priority class identification in order to reduce energy, consumed by



(a) Homogeneous scheme for temporal aggregation.



(b) Homogeneous scheme for spatial aggregation.

Figure 3.2: Homogeneous scheme for temporal and spatial aggregation.

unworkable and idle data sensing. Moreover, frequency of low-priority forwarding can be reduced from relay node which sufficiently decrease the data traffic in the network. Just as, a sensor node which is deployed in network and only dedicated to sense temperature data once every 10s and sends towards relay node while another node is only sending light value once every 15s. In this scenario, if temperature is assigned as the high priority class and sensed value of light as low priority then we can reduce the frequency of light sensing and only temperature data can be concatenated and forwarded periodically. For better analysis, consider the Figure 3.2, where scenarios of temporal and spatial aggregations are given with priority enabled aggregation. In Figure 3.2a, node A is generating temperature packets P1 and P2 and humidity packets C1 and C2 rapidly. However, we set priority of temperature while humidity packets are with less priority. Therefore, after receiving at node C, P1 and P2 are concatenated into a frame for combined transmission. On the other hand, C1 and C2 are simply forwarded gradually. Likewise, in case of spatial aggregation, shown in Figure 3.2b, where priority of any class is defined according to its data type. However, it is easy to distinguish this time because we have two different sources for two different types of data. As a consequence, P2 and C1, packets from node A and B are aggregated in a frame while P1 and C2 are delivered without aggregation.

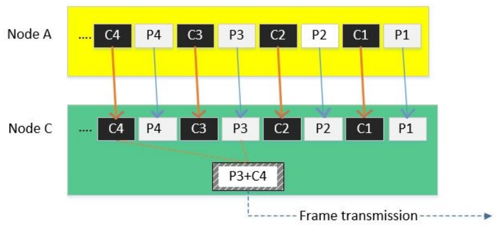
3.2.2 Heterogeneous aggregation

This scheme offers a novel aggregation model that uses sensor resources to store information. In this scheme, shown in Algorithm 2, sensor saves packets in the database for short duration rather transmitting urgently to the sink node. Packets stored in the database are then concatenated and sent according the receiver initiated protocol. The size of database buffer is kept sufficiently large to accommodate an optimal number of packets. Accordingly, after each storage cycle, previous packets are discarded and buffer clear its memory to start a new storage cycle as incoming packets can easily fill this buffer in short amount of time. In Algorithm 2, when packet is received, it is directly transferred to the buffer *d[j]*. After receiving the preferred packet numbers for concatenation, for instance *k* and *z*, these two packets are taken from the memory and aggregated to build a frame. Afterwards, this frame is transmitted towards the destination.

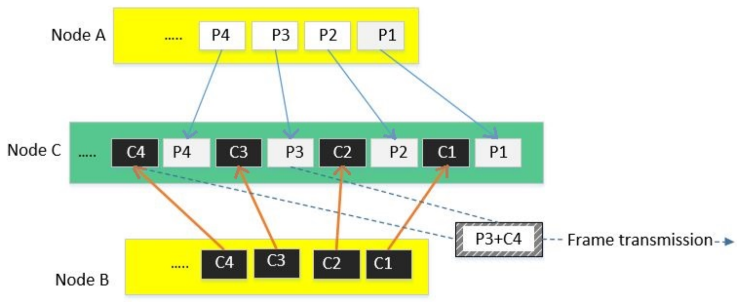
Algorithm 2 Heterogeneous or receiver initiated aggregation

```
1: i = 0
2: j = 0
3: k = 0,1,2,3,.....
4: z = 0,1,2,3,.....
5: while A packet is received do
6:     if The received packet is from node A then
7:         e[i] ← Received packet
8:         i = i+1
9:     else
10:        d[j] ← Received packet
11:        j = j+1
12:    end if
13:    if The preferred packets for aggregation are e[k] & d[z] then
14:        f[i] ← e[k] + d[z]
15:        Forward the aggregated frame from f
16:    end if
17: end while
```

This scheme is proposed by the fact that although conventional aggregation itself is an amazing technique to reduce the energy consumption and protocol overhead. However, the use of history and database in sensor nodes provide an appealing way to save number of transmissions and by that energy consumption. The database at relay node is managed and it is capable of storing incoming packets upto a considerable amount as there is a counter which counts every instant a new packet is injected into the database. For better understanding, consider the the Figure 3.3. Again, two types of aggregation scenarios, temporal and spatial, are illustrated. Starting with temporal aggregation mode, shown in 3.3a, node A is sensing temperature and humidity successively. All the packets from node A are transmitted successfully



(a) Heterogeneous scheme for temporal aggregation.



(b) Heterogeneous scheme for spatial aggregation.

Figure 3.3: Heterogeneous scheme for temporal and spatial aggregation.

and stored in the database, created in the memory of node C. During this storage cycle, if a request is initiated to aggregate two specific packets, P3 and C4, node C collects the requested packets from the database and aggregates for further transmission. Similarly, for spatial aggregation, as Figure 3.3b depicts, two types are packets are coming from node A and node B. Afterwards. packets are stored into node C's history, while waiting for the request to aggregate the desired packets, P3 and C4 in this case.

Algorithm 3 Fixed frame-length aggregation

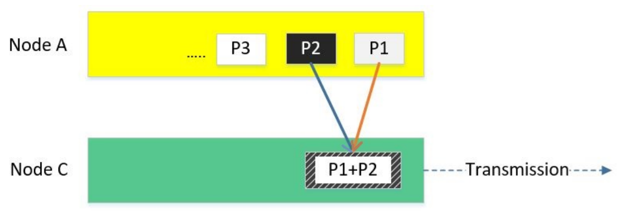
```
1: i = 0
2: while A packet is received do
3:     c[i] ← Received packet
4:     if i == 1 then
5:         d ← c[i] + c[i-1]
6:         Forward the aggregated frame from d
7:     end if
8:     i = i+1
9: end while
```

3.2.3 Variable aggregation

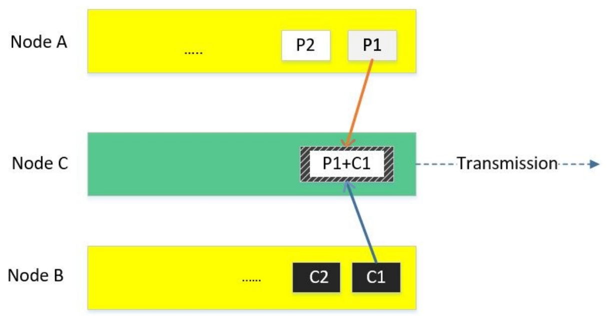
In this section, we propose a set of two schemes which contains fixed frame-length aggregation and adaptive frame-length aggregation. These two schemes are differentiated on the basis of aggregated frame length at relay node.

Fixed frame-length aggregation

In this scheme, aggregation approach is implemented in both spatial and temporal aggregation modes. Starting with the temporal mode, Figure 3.4a depicts the scenario where we have only source node A, generating data packets successively. After receiving P1 and P2 from node A, node C directly concatenates both received packets and formulates an aggregated frame which is transmitted towards the destination. However, we have a limitation here as only two packets can be concatenated to build a frame. Therefore, after receiving first packet, relay node has to wait for the second packet to perform aggregation. Accordingly in case of spatial aggregation, illustrated in Figure 3.4b, we have two source nodes A and B which are transmitting packets individually. When packets P1 is received, node C waits for C1. After receiving C1, it performs aggregation regularly and delivers to sink. This network is formed to sense real world data such as temperature, humidity and light. Therefore, in temporal aggregation approach, node A is sending temperature and humidity information consecutively. Correspondingly, in typical aggregation, there is no differentiation of data packet at node C as aggregated frame can be made of two different or same data types. However, in spatial approach, we specified each source according to the data type. for instance, A is only sensing and transmitting temperature while B is sending measured humidity values. From Algorithm 3, which is also designed for the fixed frame-length aggregation scheme, it is also noticeable that in this scheme, node C always has to wait for atleast two packets to build a frame. Further, in this scheme, C can concatenate two packets only, irrespective of the data type. The sufficient time interval between each packet transmission to avoid collision is defined by the protocol. As this scheme is only used for typical aggregation, there are certain assumptions which are made as i) This scheme is only flexible for aggregation in case two packets are received ii) There is no flexibility with respect to data types is available to provide simplicity. In order to remove these assumptions, more variable schemes are needed which we provide in next section.



(a) Variable scheme for temporal aggregation.



(b) Variable scheme for spatial aggregation.

Figure 3.4: Variable scheme for temporal and spatial aggregation.

Adaptive frame-length aggregation

The adaptive frame-length scheme affords certain flexibility in aggregation mechanism as it can concatenates all received packets within certain amount of time. In this scheme, relay node performs periodic aggregation after a specified interval, defined by the protocol. Therefore, instead of receiving one packet and waiting for atleast one more to build a frame, an relay node can concatenate as many packets as received until the frame length exceeds the maximum packet size which is 128 bytes according to the standard. We try to use Algorithm 3 to explain this scenario, although it is typically designed for fixed frame aggregation. when packet is received, it is stored in d and node counts down the time interval to perform aggregation. When the timer is expired, it performs aggregation and forwards the frame towards the sink. However, it is important to mention that there should be atleast one packet in d to build a frame. Therefore, the predefined timer is synchronized enough so that node can receive minimum one packet during this time. Furthermore, Figure is still valid for this scheme such as minor variations in frame length does not affect the overall design of this scheme. Thus, reader can still adopt Figure 3.4a and Figure 3.4b as an interpretation to examine the spatial and temporal aggregation modes in this context, in Table 3.1, a summary of the proposed schemes is presented which shows an overview for quick understanding.

3.3 Implementation of Packet Aggregation Schemes

Previous sections briefly explained the proposed topologies with associated schemes for aggregation with respect to various criterion such as flexibility, prioritization, receiver initiation, etc. In this section, our focus is to explain each and every aspect which is directly or indirectly related to the implementation of these proposed schemes. For instance, how platform is built, what are the measurement techniques used to compute delay and energy consumption in aggregation enabled network. Additionally, this section contains the software and hardware based protocol implementation of these proposed schemes with their requirement specifications, functionalities and limitations. Figure 3.5 illustrates an overview of the implementation process.

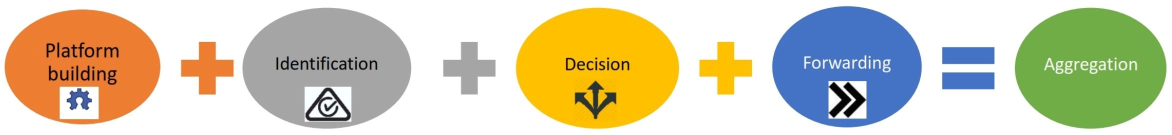


Figure 3.5: Implementation of packet aggregation: an overview.

3.3.1 Platform building

The implementation of these schemes requires an efficient assembling of framework with respect to software and hardware. We have accomplished this whole mechanism in Contiki operating system using TelosB sensor motes, shown in Figure 3.6. However, this sensor module can be easily implemented with any other operating system like TinyOS or LiteOS.

Hardware and software implementation

The hardware based implementation in Contiki does not involve any complex platform building as TelosB sensor mote is a fully developed module, without any external hardware requirements. All nodes are powered by two AA batteries so that the supply voltage is almost constant and close to 3 V. On the other hand, software implementation is carried out through various steps and procedures. In this context, all nodes are programmed using the 2.7 version of the Instant Contiki (VMWare). This version of Contiki is a easily downloadable environment which includes required libraries, compilers and some basic examples for initial configuration of sensor motes. However this environment runs on Ubuntu linux which can be virtually used through VMplayer machine execution environment [27]. In Contiki-MAC, mechanisms are asynchronous without any signalling messages and extra packet headers. An efficient wake-up process in Contiki is obtained by accurate timer where Contiki itself uses quick sleep escalation. This duty cycle utilizes regular wake-ups to listen from all other nodes in the network. For instance, during a transmission cycle, if a packet is sent while this wake-up cycle is going on, receiver is initiated to collect this packet. After a successful reception, a hardware based ACK is sent towards the transmitter to acknowledge the successful packet transmission [22]. We implement ContikiMAC on TelosB mote supported by MSP430

Figure 3.6: Implementation of packet aggregation: an overview.

microcontroller. The MSP430 contains an 8MHz, 16-bit RISC CPU, 48K bytes flash memory (ROM) and 10K bytes RAM. This architecture provides 27 instructions and 7 addressing modes. The CPU also provides sixteen 16-bit registers. The first four are dedicated for special-purpose, such as program counter, stack pointer and status register. The rest of twelve are available for general use. The ContikiMAC execution is done using Contiki 2.7, which adopts the Contiki real time timers (rtimer) to schedule regular wake-ups to make sure that its behaviour is reliable [22]. The real-time timers preempt any Contiki process at the exact time at which they are scheduled.

Rime stack

The Rime communication stack affords a set of communication primitives ranging from best-effort anonymous local area broadcast to reliable network flooding. All the protocols in the Rime stack are organised in different layers and these are arranged in an order such as more complicated protocols are appointed using the less complex protocols. In this experiment, we decided to choose communication primitives in the Rime stack based on the basic requirements of sensor network protocols. All the protocols executing on top of the Rime stack connect at any layer of the stack and utilize any of the communication primitives. One of the key factor which is imporatant to mention here that the Rime stack is compatible with all kind of communication primitives whether its a single hop or multi hop. In multi hop primitives, it is not necessary to mention the route of packets through network. Therefore, when any packet is transmitted across the network, the application or upper layer protocol is enforced at every node to select the next hop neighbour [23]. Following lines show the basic libraries which are required to implement simple unicast.

```
#include "net/rime.h"
#include "net/rime/unicast.h"
#include "<string.h>"
```

Clock synchronization

This clock library [1] is a connection between Contiki and platform based clock. It characterizes a macro, CLOCK_SECOND, to alter seconds into the tick resolution of the platform [24].

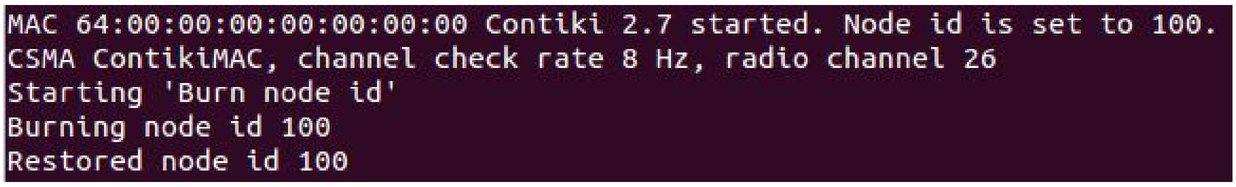
```
// clock initialization.
void  clock_init (void)


// Obtain the ongoing clock time
CCIF clock_time_t  clock_time (void)
// Obtain the on going amount of platform seconds
CCIF unsigned long  clock_seconds (void)
// Adjust the value of platform seconds
void  clock_set_seconds (unsigned long sec)
// Delay of given ticks
void  clock_wait (clock_time_t t)
// Delay a provided value of microseconds
void  clock_delay_usec (uint16_t dt)
```

However, inter-arrival time or inter-transmission time in our protocol is defined by the following way.

```
// Declare  a timer
static struct etimer et;
```

[1] www.contiki.sourceforge.net/docs/2.6/files.html

```
rimeaddr_t addr;
// Timer initialization
etimer_set(&et, CLOCK_SECOND*7);
// Timer is expired
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
```



```
MAC 64:00:00:00:00:00:00:00 Contiki 2.7 started. Node id is set to 100.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Starting 'Burn node id'
Burning node id 100
Restored node id 100
```

Figure 3.7: Process of burning node ID.

3.3.2 Node identification

This step involves the process of identifying incoming packet at the intermediate node on the basis of node ID, MAC address and sequence number. For instance, in homogeneous aggregation, the preferred node id is 4 which is assessed by identification block. This block is further described in the following sections.

Node ID and Mac address

In TelosB modification on Contiki, node ID can be burned manually to the node flash memory which can be seen at the time of node initialization. However, MAC address is hard coded in rime. To modify the node id, in directories, Contiki/examples/rime and run the following command, as shown below. Digit 0 can not be used at the start of the address because a compiler flag will show an error.

```
// Set 100 as node id
make TARGET=sky burn-nodeid.upload nodeid=100 nodemac=100
```

ACK implementation

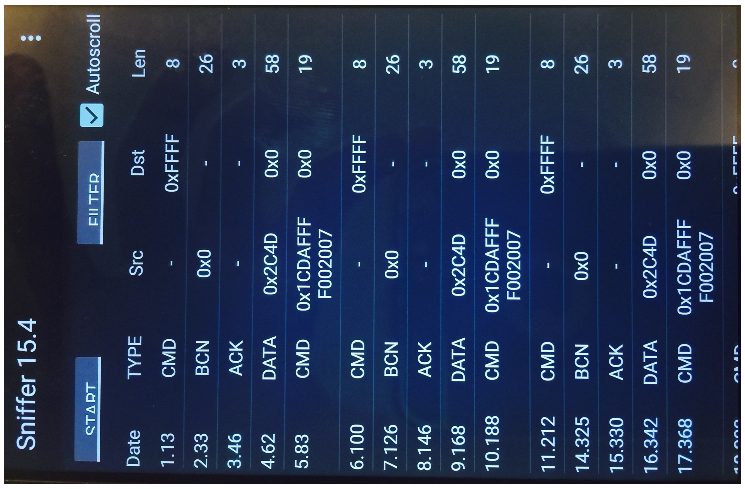
In terms of ACK implementation, a hardware based implicit ACK is implemented by setting CC2420_CONF_AUTOACK value to 1. This implementation can be done in Contiki-2.7/platform/sky/contiki-conf.h.

```
#ifndef CC2420_CONF_AUTOACK
#define CC2420_CONF_AUTOACK              1
#endif /* CC2420_CONF_AUTOACK */
```

Using Figure 3.8, where Sniffer 15.4 is operated with TelosB sensor node, we can actually capture the ongoing transmission between two nodes to make it certain whether CC2420_CONF_AUTOACK is enabled or not.

Sequence number and source ID

In WSN, sequence number is used to remove the redundancy and duplicity of received packets. Additionally, in case of spatial aggregation where number of packets are received at sink from multiple sources, it is not an easy task to separate these packets accordingly. Therefore, in our experiments, every outgoing packet from source node is equipped with a sequence number and corresponding ID of that source node. Hence, sequence number and source ID is implemented in the following way.



Date	TYPE	Src	Dst	Len
1.13	CMD	-	0xFFFF	8
2.33	BCN	0x0	-	26
3.46	ACK	-	-	3
4.62	DATA	0x2C4D	0x0	58
5.83	CMD	0x1CDAFFFF002007	0x0	19
6.100	CMD	-	0xFFFF	8
7.126	BCN	0x0	-	26
8.146	ACK	-	-	3
9.168	DATA	0x2C4D	0x0	58
10.188	CMD	0x1CDAFFFF002007	0x0	19
11.212	CMD	-	0xFFFF	8
14.325	BCN	0x0	-	26
15.330	ACK	-	-	3
16.342	DATA	0x2C4D	0x0	58
17.368	CMD	0x1CDAFFFF002007	0x0	19

Figure 3.8: Sniffer 15.4 capturing a transmission between two nodes where ACK can be seen easily.

```
// Sequence number increase every
send_packets=send_packets+1;  time a packet is sent.
// Source node ID
rimeaddr_node_addr
// Packet structure
sprintf(s,"a:%d:s:%d:Temp:%d \n",rimeaddr_node_addr,send_packets,get_temp());
```

3.3.3 Scheme implementations

As shown in the Figure 3.5, third and fourth step involve decision making and forwarding blocks. Therefore, in this section, protocol implementation and working principle is described in order to evaluate the proposed schemes. However, results and numerical outcomes are described in Chapter 4.

Homogeneous aggregation

We already described in the design section about this scheme which only aggregates packets from same class while packets from other data classes are simply forwarded without aggregation. Figure 3.9 shows a spatial aggregation scenario using homogeneous approach where we consider two source nodes A and B, which are sending data packets towards node C. Node C first receives packet D1 and sends ACK A1 towards node A. After this transmission, node B transmits D2 towards node C and receives A2. Meanwhile, D2 is simply forwarded towards sink without any further processing as data from node B is not prioritized in this scheme. After this forwarding, node C receives A3 from node D. Moreover, node A sends another packet D3 to C and node B sends D4 to C. Therefore, D4 is simply forwarded while D1 is concatenated with D3 and forwarded towards D. Similar approach is used in temporal aggregation when one source is sending two different types of packets, therefore, only one class of packets are aggregated while other packets are delivered without performing aggregation. It is important to mention that Figure 3.9 might show some collisions, however, we adjust the etimer_set(&et, CLOCK_SECOND) precise enough to avoid any possible collisions during this whole mechanism. Therefore, this scheme involves two forwarding mechanisms and packets identified on the basis of source id information which exists into payload. The content of received packet is checked at the first index of buffer *f* by *f[0]* = A. Afterwards, *p[0]*= 0 is applied to check whether *p* is empty or not.

In addition, strcpy(p, f)is used to copy the contents of *f* to *p* as *f* should be empty to receive the next incoming packet. In case the received packet is from node B, it is copied to *c*.

```
if (f[0] == '\0')
```

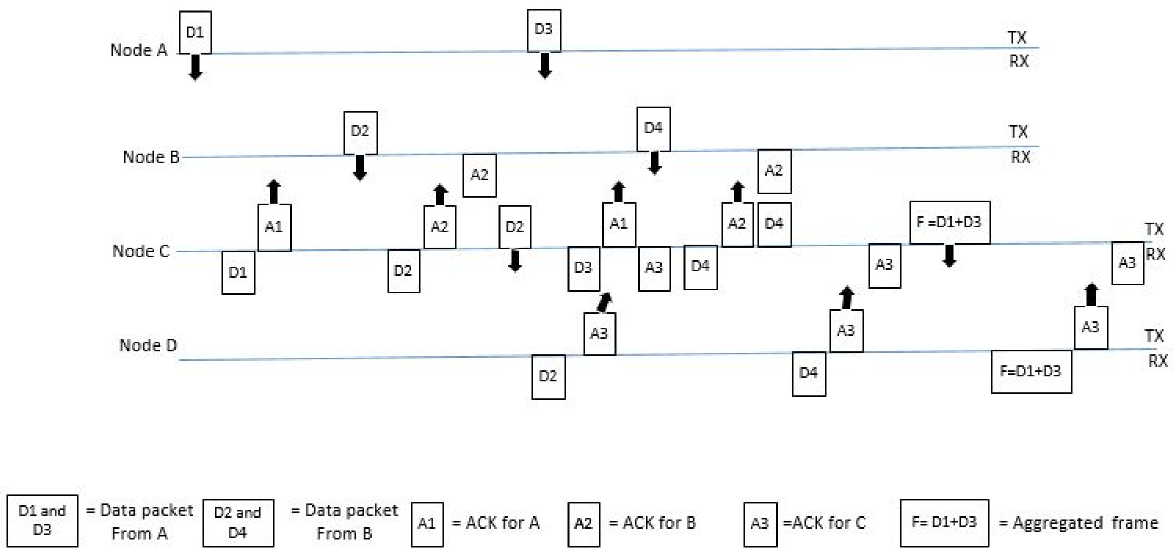


Figure 3.9: Homogeneous scheme for spatial aggregation.

```
(char *)packetbuf_copyto(f);
if (f[0]== 'A')
if (p[0] == '\0'){
strcpy(p, f);
else
strcpy(q, f);}
if  (f[0]== 'B'){
(char *)packetbuf_copyto(s);
strcpy(r, s);}
// Check if both packets from the same class are received
if (p[0] != '\0' && q[0] != '\0' ) {
// Perform aggregation
c=strcat(p,q);
```

Heterogeneous aggregation

To explain this scheme, we refer to Figure 3.3 which illustrates this scheme in an appropriate way. In both spatial and temporal heterogeneous aggregations, the implementation is nearly similar. Therefore, we consider spatial aggregation only for better understanding of the reader. In this context, when packets are coming from node A and B towards C, node C assigns two external buffers *f[i]* and *g[j]* with sufficient size to accommodate maximum packets in its memory. In our case, *e[i]* is allocated to node A while *g[j]* is used to accommodate date from B.

```
// Store packets from A to f[i]
if (from->u8[0]== A) {
i=i+1;  i<=20;
packetbuf_copyto(f[i]);}
// Store packets from B to g[j]
else if (from->u8[0]== B){
j=j+1; l<=20;
packetbuf_copyto(g[j]);  }
```

```
if (  f[3]!='\0' && g[4]!='\0' ) {
strcpy (m,f[3]);
strcpy (n,g[4]);
// Checks the content of f[3] and g[4]
printf ("contents of f3 :'%s'\n" ,f[1]);
printf ("contents of g4 :'%s'\n" ,g[2]);
// Aggregation
c=strcat(m,n);
```

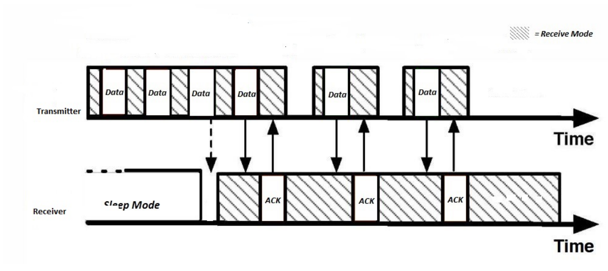
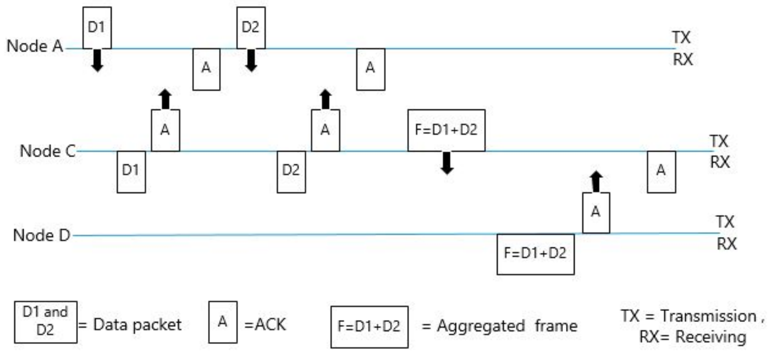
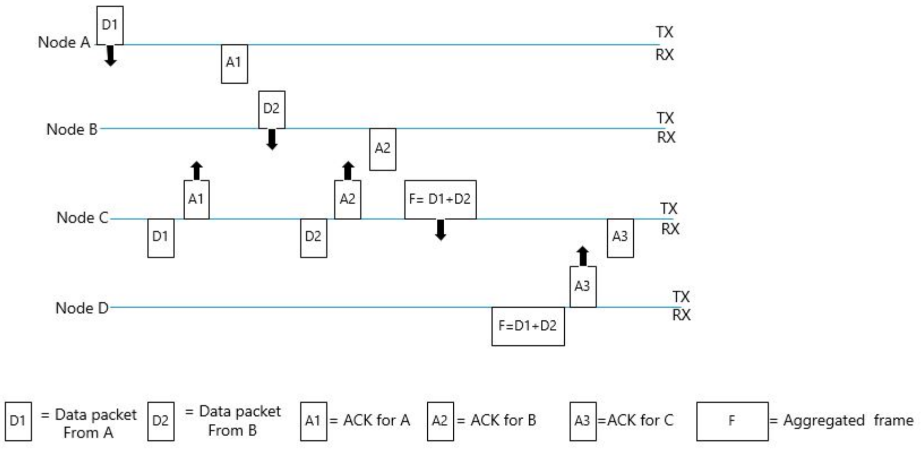


Figure 3.10: ContikiMAC working principle.



(a) Variable scheme for temporal aggregation.



(b) Variable scheme for spatial aggregation.

Figure 3.11: Protocol implementation of variable scheme for temporal and spatial aggregation.

We set a limit to store 20 packets for both *f[i]* and *g[j]*. When the limit exceeds, buffers are cleared using strcpy(f, ""),strcpy(g, ""). During packet reception, receiver initiates a request to transmit some specific packets which are already stored in the memory. In this scenario, node A is generating temperature data while B is generating humidity. Accordingly, if receiver initiated request is for 3rd packet of temperature and 4th packet of humidity, as in Figure 3.3b, it follows the mechanism and checks whether *e[3]* and *g[4]* are empty or not. In case that *f[3]* and *g[4]* are not empty, packets are copied to the variable *m* and *n* to perform aggregation. Before aggregation, *printf* is used to check the contents of *f[3]* and *g[4]* to verify that these packets fulfill the required request. After performing aggregation, frame is stored in *c* which is further used for forwarding.

Fixed frame-length scheme for both spatial and temporal

In the temporal aggregation scheme, Shown in Figure 3.11a, sensor node A sends data packet D to relay node C. After receiving data packet, C sends an ACK back towards A and waits for an other packet. In the next transmission, A sends another packet towards C and receives an ACK as a response from C. After receiving two packets, C concatenates both packets into a frame F and forwards towards D. Hence, C is receiving packets and forwarding these packets to D so it is behaving as sender and receiver at the same time. After receiving data packet from relay node, D sends an ACK back towards C. Similarly, in Figure 3.11b, where spatial aggregation is going on and A sends packet D1 to C and receives and A1. Afterwards, B sends another data packet D2 and attains A2 in response. After receiving both D1 and D2, C perform aggregation and forwards the frame F towards node D. After receiving this frame, D sends A3 towards C and through this whole process, this topology performs aggregation. All of these experiments are working on CSMA ContikiMAC protocol in Contiki 2.7. ContikiMAC is built for static networked nodes and depends on unicast and broadcast transmissions. According to the basic working principle, shown in Figure 3.10, when unicast is executed, the transmitter transmits the packet that consists of the payload, header part and the destination address. In contrast, the intended receiver wakes-up to sample the medium for packet transmissions from its neighbours regularly. Once a packet arrives while wake up period is going on, the receiver keeps the radio ON to receive the packet. When the reception is completed the receiver sends ACK. On the other side, when broadcast is executed, receiver do not sends any ACK. Rather, source node periodically sends the data packet all along the wake-up interval to make sure that every node received it properly. The detailed timer adjustments among different nodes are achieved through our programme. Node D is directly connected to a laptop via USB. The orientation of all nodes is certain enough such that every node can directly communicate with each other without any interference or human interruption. For better understanding, basic coding parts of this scheme for each node are explained here.

```
// Returns temperature value
static int get_temp(void)
// Returns humidity value
static int get_humidity(void)
// Sensing and conversion
((sht11_sensor.value(SHT11_SENSOR_TEMP)
  // Packet structure includes sequence number, source id and data
  sprintf(s,"a:%d:s:%d:Temp:%d \n",rimeaddr_node_addr,send_packets,get_temp());
  // Timer adjustments
  static struct etimer et;
  etimer_set(&et, CLOCK_SECOND);
  // Split string into tokens
  char *strtok(char *restrict s1, const char *restrict s2);
  //Concatenate two strings
  strcat(f,g);
  // Activate sensor modules
  SENSORS_ACTIVATE(sht11_sensor);
  SENSORS_ACTIVATE(light_sensor);
  //  Call may take up to 210ms
```

These functions, mentioned above, are basics which are used in typical aggregation schemes. As node A is assigned to sense temperature only so function get_temp(void) is defined to return temperature value. Similarly, B is using get_humidity(void) to return humidity. At node C, strcat(f,g) is used to concatenate two strings if both strings are saved in f and g. Further, aggregation with adaptive frame-length is also implemented in the same manner. However, node perform periodic aggregation with requirement of at least one packet in it frame.

```
if (f[0] == '\0'){

// Packet received from A
printf("unicast message received from %d: '%s'\n",
from->u8[0],(char *)packetbuf_dataptr());
(char *)packetbuf_copyto(f);
// To clear in internal buffer
packetbuf_clear(); }
if (f[0] != '\0'){

// Packet received from B
printf("unicast message received from %d: '%s'\n",
from->u8[0],(char *)packetbuf_dataptr());
(char *)packetbuf_copyto(g); }
if (g[0] != '\0'  ){
h=strcat(f,g);
```

At node C, three external buffers *f*, *g* and *h* are used. after receiving packet address from $->u8[0]$, node ID is identified and packetbuf_copyto() copies the content to external buffers *f* or *g* accordingly.

```
// Matches intended destination node with itself
if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)){
unicast_send(&uc, &addr);
// Forwards the concatenated frame towards the sink
printf("Forwarding Concatenated Packet to %d: '%s'\n" ,addr.u8[0],(char *)c);  }
```

Moreover, *h* is used to save and forward the frame towards the intended sink node. !rimeaddr_cmp compares the current node ID with the destination ID to remove the effect of duplicity.

Adaptive frame-length aggregation

Furthermore, in variable aggregation with adaptive frame-length, we use three source nodes which are sending temperature, humidity and light packets towards the relay node. So protocol for the source nodes remain same. However, we make certain changes at relay and sink nodes. After first packet, as shown in Figure 3.11, node C periodically performs aggregation after a certain time. Three source nodes are set to transmit packets with a very small time difference to remove unnecessary delay at relay node. Therefore, at relay node, whatever is received from these source nodes, it will be aggregated into frame F. However in case of temporal aggregation, only one source node is generating three different kinds of packets.

```
if (f[0] == '\0'){
(char *)packetbuf_copyto(f);}
{ if  (from->u8[0]== A)
(char *)packetbuf_copyto(g);
if  (from->u8[0]== B)
(char *)packetbuf_copyto(h);}
if (h[0] != '\0')
// Concatenation of all received packets
snprintf(m, 80, "%s%s%s", f, g, h);

// Clear the external buffers
strcpy(f, "") ;
strcpy(g, "") ;
strcpy(h, "");
```

Again, we use three external buffers *f*, *g* and *h* to differentiate among different source nodes. Initially, when any packet is received, it is packetbuf_copyto() copies this packet into *f*. Afterwards, if the sender is A, it is saved to *g* and in case of B, it is saved into *h*. As all the source exceptions are there so if packet is coming from another source, it remains in *f*. When temporal aggregation is performed, packets are identified on the basis of their content. At sink node, when frame is received, it is de-aggregated using strtok_r() function which separates three strings. Accordingly, packets are identified on the basis of source ID, which is initially inserted into the payload.

```
// Frame reception
printf("concatenated message received from %d.%d
with data length %d & total length %d :'%s'\n",
from->u8[0],from->u8[1],packetbuf_datalen(),
packetbuf_totlen(),(char *)packetbuf_dataptr());
packetbuf_copyto(d);
```

Conclusively, this chapter explained our several network topologies and schemes proposed by us with respect to network design and implementation. Furthermore, each scheme is illustrated with the help of pseudo code and diagrams which provides all necessary details in order to understand the architecture and employment.

Chapter 4

Test Scenarios and Experimental Results

Packet aggregation has been appeared as a fundamental approach for resource conservation and energy efficient data gathering in WSNs. Whilst performing aggregation enabled communication in TelosB sensor motes using Contiki platform, experimental results and measured outcomes can be acquired using several methods and approaches. For this thesis, we perform test-bed experiments and execution of proposed schemes to assess the performance of aggregated forwarding over non-aggregated forwarding. Finally, we initialize system clock library to determine average end-to-end delay and transmission count.

In this chapter, we proceed to discuss various testing scenarios for each scheme accompanying performance parameters such as energy consumption and average end-to-end delay. Additionally, how these parameters are overwhelmed by node employment and designated inter-departure time. We describe the techniques which are used to measure energy consumption, average end-to-end delay and packet capturing. Later, we present details of implemented schemes and analogous numerical results of each attended scenario with the help of tables & graphs for better analysis. In addition, we show the captured packets through Sniffer 15.4 and analyse these capturing through Wireshark.

4.1 Test Scenarios and Performance Parameters

For performance assessment, proposed technologies are examined on the basis of several performance parameters for instance energy consumption, delay and count. In this chapter, initially the different compiled scenarios considered are explained. However, as mentioned in Chapter 3, performance parameters are sometime dependent on relay node as this node is behaving as an aggregator in all schemes. Therefore, while conducting the energy measurements, we mostly focus on relay node as number of transmissions vary from here. In this section, we are going to discuss some design objectives which were under observation till the end of this task. Furthermore, we explain the measurement techniques which are adopted to estimate the energy consumption and delay. Afterwards, we describe each scheme with graph illustration and measured parameters and compare the overall energy and delays in one big picture. Afterwards, the obtained numerical results for each scenario and transmission scheme is presented via graphs. A description of the behaviour of observed results is also presented.

4.1.1 Scenarios

Following are the scenarios for our proposed aggregation schemes. Table 4.1 presents an overview of all the proposed schemes with a brief description. At first, we have S1 which contains homogeneous scheme for temporal and spatial aggregation. This scheme allows aggregation for one service type either its temperature or light while other types of packets are simply forwarded. S2 belongs to heterogeneous scheme where a database is made at the relay node and packets from different classes are stored. After initiation of a request from the receiver, relay performs aggregation according to the request based on different service types. Moving further, S3 comprises fixed frame-length scheme which has the maximum ability to aggregating two packets inside one frame. There is no limitations about any specific service type for aggregation. Finally, S4 contains an adaptive frame-length scheme which can aggregates as many packets as a frame can accommodate within its payload. Additionally, this aggregation is performed periodically with the limitation of having at least one packet inside an aggregated frame. In all scenarios, we assume that source nodes are working independently and packets are arriving periodically. It is important to mention that although we are considering fixed frame-length and adaptive frame-length separately , however, we explain the experimental results for both scenarios in Section 4.4 as these both schemes fall under one umbrella of variable aggregation.

Table 4.1: An overview of all scenarios for spatial and temporal aggregation

Scenario	**Name)**	**Explanation**
S1	*Homogeneous scheme*	*Aggregation based on same service type*
S2	*Heterogeneous scheme*	*Aggregation based on different service type from a data base*
S3	*Fixed frame-length scheme*	*Maximum occupancy of two packets*
S4	*Adaptive frame-length scheme*	*Flexible occupancy with minimum one packet*

4.1.2 Performance parameters

Performance parameters are the criterion which evaluate any protocol or scheme whether it is efficient or not. Following are the performance parameters to evaluate the design and efficiency of our proposed schemes.

- *Energy consumption*

  Sensor nodes are always energy limited. Therefore, the proposed model attains energy conservation by reducing number of transmissions from relay node towards sink node.

- *Number of frame transmissions*

  As number of transmissions is directly related with energy consumption and system overhead. Accordingly, by applying aggregation schemes, number of frame transmissions can easily be reduced to an optimal number which eventually reduces the energy and resource consumption associated with each transmission.

- *Delay*

  The end-to-end delay is the amount of time taken by a packet, to be transmitted from a source node till sink. Thus, by reducing number of transmission, overall time related with these transmissions also changes. We calculated end-to-end delay for one packet and accumulated delay for 100 packets.

- *Packet drop*

  A packet drop is a phenomena which happens when packets are lost during the routing process between sender and the receiver due to several reasons. We examined the packet drop in all aggregated schemes with respect to varying distances.

4.1.3 Performance measurements

This section comprises of some methods which by which we measure the energy and delay in sensor nodes and how these techniques are used in our scenarios in order to measure node energy and average end-to-end delay per packet.

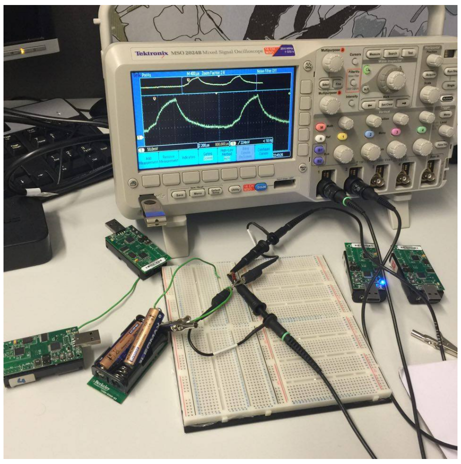


Figure 4.1: Real life experimental set up to measure power consumption in TelosB sensor motes using oscilloscope.

Energy consumption measurements based on oscilloscope

WSN energy usage experiments require accurate measurement of power consumed by a sensor over time. However, measurement is actually comprised of both hardware and software deployment. In hardware part, an efficient technique that we adopted is interfacing of TelosB mote with an oscilloscope while measuring process is carried out. Accordingly, in software implementation, coding based delay profiling is used. Furthermore, TelosB operates at constant supply voltage. To measure the power levels, all the nodes are connected to oscilloscope. The input voltage level is set to 3 V. It is also noticeable that 2.1 V is the minimum required voltage for the CC2420 [1] radio to operate accurately. Except one node at each time who is taking power from the USB of a PC, all others are powered using standard AA batteries. Hence, power consumption measurements can be taken, by measuring the drawn current and input voltage, shown in Figure 4.1. Afterwards, energy can be calculated by multiplying the measured value of power with the associated delay. These measurements are usually performed during normal operation of the platform while observing the power consumption when different components of the board are active, and its always time dependent. In these kind of experiments, the voltage measurement has its own significance on accuracy along with the tolerance on the shunt resistor value, the stability of the supply voltage, and the measurement rate, which should match the analysed experience. It is also noticeable that in these kind of circuits, the voltage drop across the resistor reduces the supply voltage which is powering the TelosB mote. In contrast, low values of resistor lead to reduced values of voltage difference, consequently, affects lower measurement sensitivity. Accordingly, power can be calculated by this formula:

$$P = VI \tag{4.1}$$

Where, V =Input voltage, I = Current consumption. Moreover, energy can be easily calculated from these power values by multiplying with calculated delay.

$$E = P \times T \tag{4.2}$$

[1] www.ti.com/lit/ds/symlink/cc2420.pdf

In Eq. (4.2), T can be transmission, reception or processing delay and Tx and Rx is used to represent transmission and reception respectively.

$$TxPower = TxCurrent \times InputVoltage \tag{4.3}$$

$$RxPower = RxCurrent \times InputVoltage \tag{4.4}$$

Furthermore, using values from Eq. (4.3) & Eq. (4.4) overall energy of a node for *N* packets can be calculated by the following way.

$$E = \sum_{i=1}^{N} (Pti \times Tt + Pri \times Tr + Psi \times Ts) \tag{4.5}$$

Where Tt and Tr represents the transmission and reception delay respectively in Eq. (4.5). In addition, Ts is used as sleep delay. However, as we are not considering any sleep interval so sleep energy is negligible. Inside a network, the total energy for 100 packet from source node till relay can be calculated as,

$$E = \sum_{i=1}^{100} Pti \times T1 + \sum_{i=1}^{100} Pti \times T2 + \sum_{i=1}^{200} Pri \times Tr \tag{4.6}$$

After taking measurements, Eq. (4.6) is used to compute transmission and reception energy of a TelosB motes. Transmission and reception current are 20.36 mA and 23.63 mA respectively with input voltage 3V. These values are measured by us through an early project before this thesis [25]. It is noticeable that energy, calculated from Eq. (4.6) is not the total energy of the whole topology rather it is the energy till packets reach the relay node. To be more specific, in most of the scenarios in WSNs, relay nodes create energy hole as these nodes has to deal with a lot of traffic going towards the sink. Therefore, relay deplete energy very fast and dies immaturely. Considering this fact, we only focus on the relay node while calculating energy consumption in our schemes.

Delay measurements based on clock library

Average end-to-end delay is the time duration which a packet spends from source node till destination. It consists of propagation delay, processing delay, queuing delay and transmission delay. However. in our experiments, propagation delay is negligible as there is not much distance between nodes during transmission cycle. As the clock cycle among all nodes is not synchronized so all source nodes can transmit packets successively. On the other hand, receiver can only receive one packet at the time. The queuing delay is considered negligible here. Transmission delay can be calculated where *T* is the time consumed by one packet of length *L* bits through a channel with bandwidth of *B*. Accordingly, in our scenarios, channel bandwidth 250 kbps and typical size of a packet without aggregation varies between 30 bytes to 40 bytes. Thus transmission delay can be easily calculated in Eq. 4.7. Nevertheless, the mechanism to calculate the processing delay of all nodes for each scheme is slightly complicated. Although, there are various ways to calculate the time duration employed by a software to execute the required operations. Apart from this fact, Contiki provides a set of real-time timers which can measure the processing time precisely. Utmost resolution of 0.4069 microseconds is afforded by the primary clock *MCLK* if current cycle is performed at 2.4576 MHz . However, *MCLK* is not able to work in less power environment and Contiki has the probability of 0.25 to work in available low power environments. Accordingly, and additional clock is available, called *ACLK* which can perform with the frequency of 32768 Hz utilizing crystal [24]. Therefore, we use *ACLK* as a primary clock for our delay measurements.

```
start=clock_time();
ecc_make_key ();
diff=clock_time() - start;
num_seconds=(double)diff (ticks)/CLOCK_SECOND;
```

```
CLOCK_SECOND = 128 ticks
t1=RTIMER_NOW();
ecc_make_key ();
t2=RTIMER_NOW();
printf("Ticks= %u\n", t2-t1);
  printf("Completion time  %lu / %u \n",
  (unsigned long)clock_time() - start_time, CLOCK_SECOND);
```

At first, initialization of clock _ init() takes place and value of incremented ticks is stored by the function clock _ time(). Therefore, the value of the initial tick counter and the final counter after code loading and compilation is compared to measure the processing time it took to perform this task.The consumed time is in number of ticks. To get the time in seconds we divided the consumed number of ticks by total number of ticks in one second (128 ticks). This library is defined in Contiki-2.7/cpu/cc2430/8051def.h. Accordingly, average end-to-end delay can be calculated by adding transmission delay (Tt), processing delay (Tpr), queuing delay (Tq) and propagation delay (Tpg), as shown in this formula,

$$Te2e = Tt + Tpg + Tpr + Tq \tag{4.7}$$

As in our network topologies, nodes are closely situated with each other and each packet is sent with a specified time interval, therefore in our case we neglect the propagation delay and queuing delay. In addition, receiving delay is considered as equal to the transmission delay. Moving further, transmission delay can be calculated as,

$$Ttrans = \frac{L}{B} \tag{4.8}$$

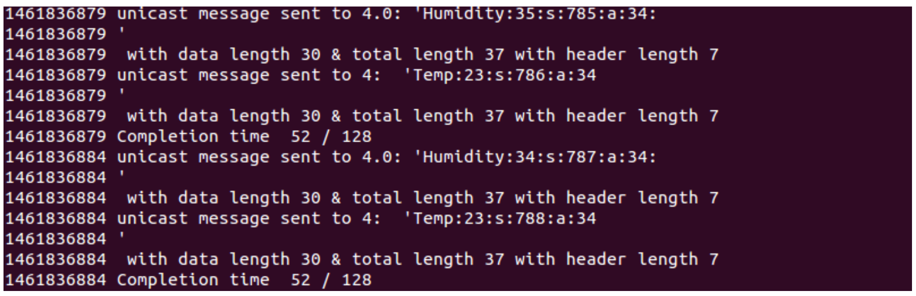
Where *L* is size of transmitted packet in bits and *B* is representing the data rate of the transmission channel.

Accordingly, end-to-end delay in our schemes is calculated in Eq. (4.8).

$$Te2e = Tt(source) + Tpr(source) + Tpr(relay) + Tt(relay) + Tpr(sink) \tag{4.9}$$

4.2 Experimental Results for S1: Homogeneous Aggregation

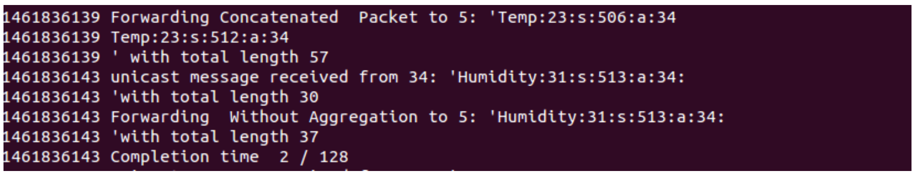
To evaluate the performance of proposed schemes, we examined aggregated and non-aggregated transmissions on the basis of energy consumption of the relay node, average end-to-end delay of a single packet, number of transmissions and overall delay according to the number transmissions. Moreover, the homogeneous aggregation is examined for both temporal and spatial aggregation modes. Here, we are only



```
1461836879 unicast message sent to 4.0: 'Humidity:35:s:785:a:34:
1461836879 '
1461836879  with data length 30 & total length 37 with header length 7
1461836879 unicast message sent to 4:  'Temp:23:s:786:a:34
1461836879 '
1461836879  with data length 30 & total length 37 with header length 7
1461836879 Completion time  52 / 128
1461836884 unicast message sent to 4.0: 'Humidity:34:s:787:a:34:
1461836884 '
1461836884  with data length 30 & total length 37 with header length 7
1461836884 unicast message sent to 4:  'Temp:23:s:788:a:34
1461836884 '
1461836884  with data length 30 & total length 37 with header length 7
1461836884 Completion time  52 / 128
```

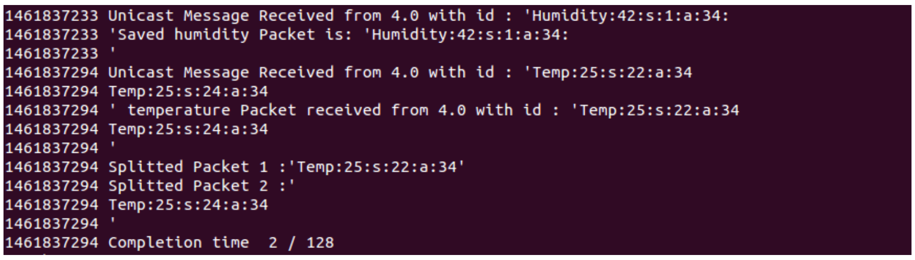
Figure 4.2: Packet transmission from the source node in the homogeneous scheme for temporal aggregation.

showing the figures for some temporal aggregation and spatial aggregation modes however numerical results are included for all temporal and spatial aggregation modes. Source identification can be seen in



```
1461836139 Forwarding Concatenated  Packet to 5: 'Temp:23:s:506:a:34
1461836139 Temp:23:s:512:a:34
1461836139 ' with total length 57
1461836143 unicast message received from 34: 'Humidity:31:s:513:a:34:
1461836143 'with total length 30
1461836143 Forwarding  Without Aggregation to 5: 'Humidity:31:s:513:a:34:
1461836143 'with total length 37
1461836143 Completion time  2 / 128
```

Figure 4.3: Packet transmission from the relay node in the homogeneous scheme for temporal aggregation.



```
1461837233 Unicast Message Received from 4.0 with id : 'Humidity:42:s:1:a:34:
1461837233 'Saved humidity Packet is: 'Humidity:42:s:1:a:34:
1461837233 '
1461837294 Unicast Message Received from 4.0 with id : 'Temp:25:s:22:a:34
1461837294 Temp:25:s:24:a:34
1461837294 ' temperature Packet received from 4.0 with id : 'Temp:25:s:22:a:34
1461837294 Temp:25:s:24:a:34
1461837294 '
1461837294 Splitted Packet 1 :'Temp:25:s:22:a:34'
1461837294 Splitted Packet 2 :'
1461837294 Temp:25:s:24:a:34
1461837294 '
1461837294 Completion time  2 / 128
```

Figure 4.4: Packet reception and de-aggregation at the sink node in the homogeneous scheme for temporal aggregation.

both figures where *a:32* represents the associated source ID in both figures. Furthermore, Figure 4.2 illustrates a temporal transmission scenario for a source node where node 34 is sending sensing temperature and humidity data from the environment and sending towards node 4 which is a relay node and behaving as an aggregator. These packets contain sequence number, sensed data and source ID as *s* indicates sequence number and *a* indicates source ID. According to the Figure 4.3, node 4 receives these packets and checks the predefined class priority for aggregation and simple forwarding. Afterwards, it simply concatenates two temperature packets into one frame and forwards towards sink which is node 5. However, as humidity packets belong to low priority class, these packets are simply forwarded without performing aggregation. Moving further, Figure 4.4 depicts another scenario where node 5 is receiving the concatenated packets. Humidity packets are simply received and saved into a buffer for further processing while temperature packets are de aggregated after reception and then saved into node's memory for further analysis. By using this mechanism, homogeneous aggregation is achieved in our topologies. It is important to mention that above mentioned figures are here for just illustration of the whole process as these are not taken from a same transmission so sequence numbers are different in each figure.

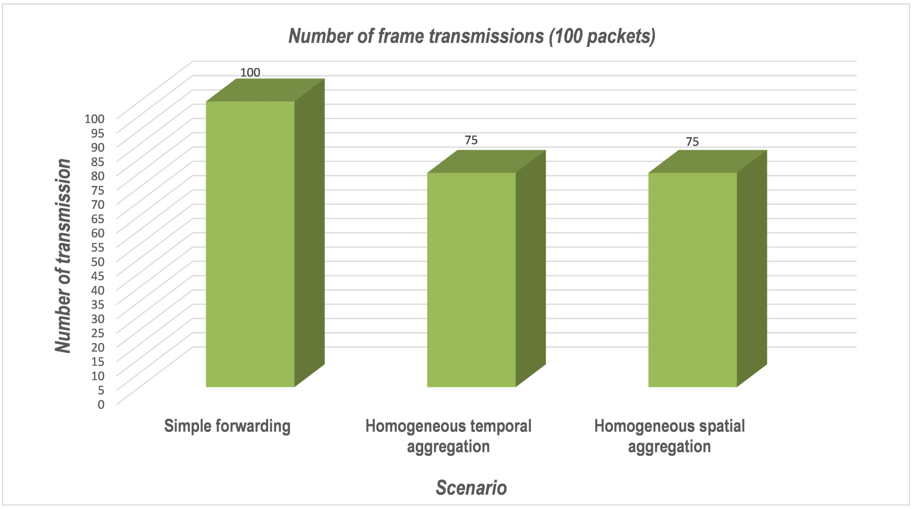


Figure 4.5: A comparison of frame transmission count from the relay node between homogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

4.2.1 Results for number of frame transmissions

Figure 4.5 illustrates a comparison of average transmission count from relay node for both scenarios. These values are taken while transmitting 100 packets from the relay node. As we already described about the working principle of homogeneous aggregation where different types of data is prioritized. For that reason, only temperature packets are concatenated while humidity packets are simply forwarded, as shown in Figure 4.3. Consequently, 50 packets of temperature are concatenated in a fashion of 2 packets per frame. On the other side, 50 packets of humidity are simply delivered to the sink without any further processing. It is important to mention that all the received packets at relay node are of same size regardless of the fact that from which data class they belong. Moving further, 50 frames are transmitted into 25 transmission units while 50 non-aggregated packets are transferred into 50 transmissions. In the course of time, number of transmissions, required to transfer 100 packets from relay to sink node, is reduced to 75. Although, number of transmissions from source to relay can also be reduced as if source node is performing aggregation. Nevertheless, we perform aggregation at the relay so transmission count from source till relay remains same for any scenario. Due to this fact, we specifically observe relay node as our reference point for the measurements of energy and transmission count.

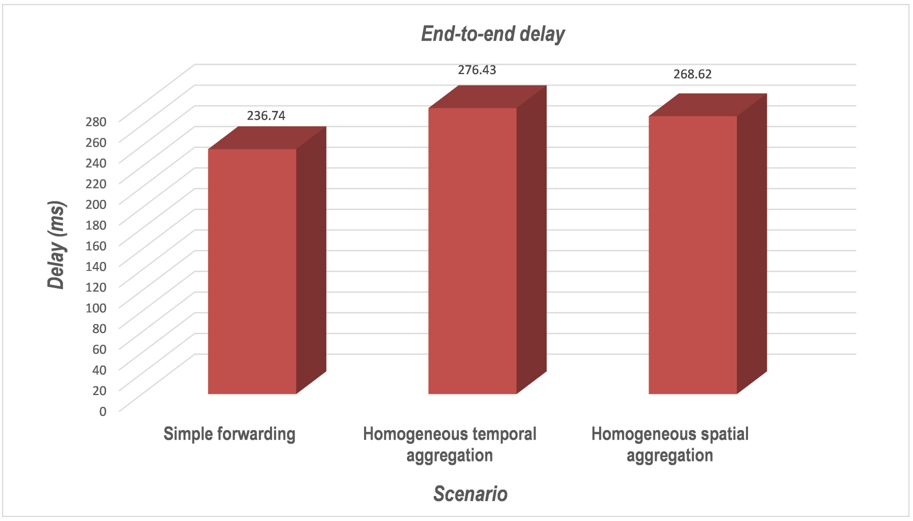


Figure 4.6: A comparison of average end-to-end delay for a single packet with respect to homogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for a single packet.

4.2.2 Results for end-to-end delay

In Figure 4.6, we explain the average end-to-end delay of a packet from source node till destination. This average end-to-end delay is slightly higher in aggregated schemes as compared to non-aggregated one. Our end-to-end delay calculations include transmission delays and processing delays for each node. Starting with the simple forwarding where we get average of 236.74 ms for one packet, this delay includes transmission delay of 1.18 ms and processing delay of 234.37 ms at the source node, total value in mentioned in Table 4.2. Similarly for relay node, we have transmission delay of 1.184 ms but processing delay is almost approaches to zero and for sink node we only have processing delay which approaches to zero. In contrast, homogeneous aggregation schemes involves a lot of processing delay at relay node and sink nodes. This is because after aggregating one frame, relay node has to process another packet without aggregation and forward. Accordingly, relay node spends 7.81 ms and 15.62 ms in terms of processing delay for spatial and temporal aggregation respectively. Additionally, sink has to manage two types of packet reception which is aggregated and non-aggregated. However, in simple forwarding, sink node does not associate any de-aggregation so no processing is involved. Nonetheless, processing delay in homogeneous aggregation is 23.43 ms for both spatial and temporal which is higher as compared to the simple forwarding.

Table 4.2: A comparison of average end-to-end delay between homogeneous aggregation scheme vs simple forwarding for a single packet

Schemes	Delay at source node	Delay at relay node	Delay at sink node
Temporal aggregation	*235.55 ms*	*17.44 ms*	*23.43 ms*
Spatial aggregation	*235.55 ms*	*9.63 ms*	*23.43 ms*
Simple forwarding (without aggregation	*235.55 ms*	*1.18 ms*	*0 ms*

4.2.3 Results for overall delay

This parameter, illustrated in Figure 4.7, explains the overall delay of 100 packets with respect to transmission counts for 100 packets. For instance, we calculated end-to-end delay for a single packet in the

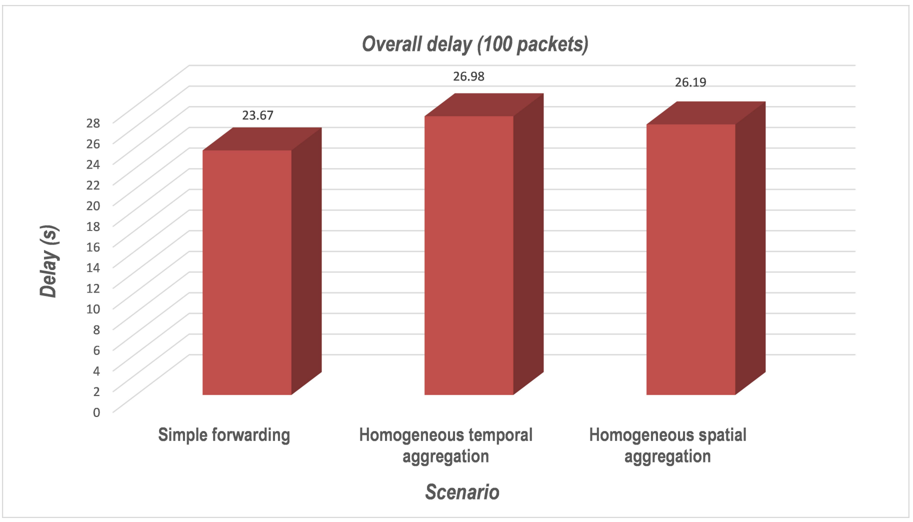


Figure 4.7: A comparison of overall delay for 100 packets with respect to homogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission.

previous section which was 236.74 ms for simple forwarding, 276.43 ms and 268.62 ms for temporal and spatial homogeneous aggregations. In this case, we multiply the transmission delay of relay node with respect to transmission count. As to transmit 100 packets, we have 50 packets transmissions and 25 aggregated frame transmissions in homogeneous aggregation. So relay node's transmission delay for a frame is multiplied by 25 and transmission delay for a simple packet is multiplied with 50. For that reason, we accumulate the overall delay for 100 packets which is 26.98 s and 26.19 s for temporal and spatial mode respectively. While simple forwarding for 100 packets takes 23.67 s which is overall less than homogeneous aggregation.

4.2.4 Results for relay node energy

In case of homogeneous aggregation scheme, as described before, energy consumption is shown in Figure 4.8, where we perform a comparison of relay node's energy with respect to aggregated and non-aggregated scenario. Additionally, energy consumption measurements in all proposed schemes only involve transmis-

Table 4.3: A comparison of relay node's energy consumption between homogeneous aggregation scheme vs simple forwarding for 100 packets

Schemes	Etx (1 packet)	Erx(1 packet)	Total Energy (100 packets)
Temporal aggregation	*0.11 mJ*	*0.083 mJ*	*14.65 mJ*
Spatial aggregation	*0.11 mJ*	*0.083 mJ*	*14.65 mJ*
Simple forwarding (without aggregation)	*0.072 mJ*	*0.083 mJ*	*15.5 mJ*

sion and reception energy for each node. It is clearly mentioned that energy consumption for homogeneous aggregation for both modes is 14.65 mJ while without aggregation is 15.5 mJ. This difference can be observed by a detailed analysis which is given in Table 4.3. As we see that reception energy for relay node remains same for both aggregated and non-aggregated scenarios because all the packets coming from

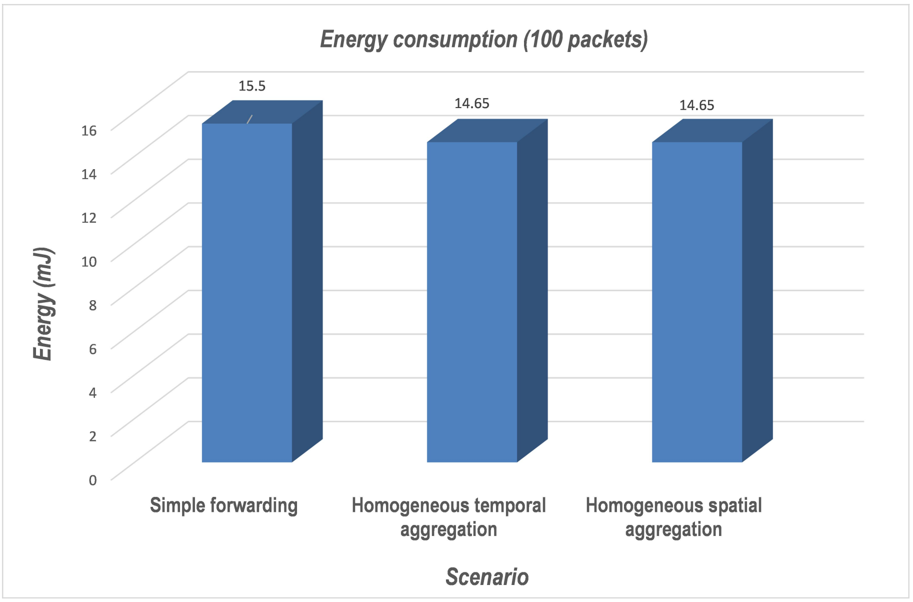


Figure 4.8: A comparison of average energy consumption of the relay node with respect to homogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

source nodes are of same sizes. However, at relay node, when packets from same service type are aggregated while other packets are simply forwarded, transmission energy varies. In terms of total energy of an relay node, it depends how many transmission we have for each case. To be more specific, we received 100 packets at relay node and according to the scheme, 50 packets of temperature packets which should be aggregated in the form of 2 packets per frame while 50 packets of humidity are simply transmitted into 50 transmissions. Therefore, we transmit 75 packets including 50 non-aggregated packets and 25 aggregated frames. These values eventually leads us towards less energy consumption which is 14.65 mJ for 100 packets. These results show that even partial aggregation of 50 packets can decreases the energy consumption. Although, this difference is not nominal, however, energy consumption can be reduced by avoiding partial simple forwarding which consumes 50 % of energy.

4.2.5 Scheme summary

Table 4.4: Overall comparison between homogeneous aggregation scheme vs simple forwarding with respect to energy consumption, average end-to-end delay, overall delay and frame transmission counts

Schemes	Energy consumption (100 packets)	Transmission count (100 packets))	End-to-end delay (1 packets)	Overall delay (100 packets)
Temporal aggregation	*14.65 mJ*	*75*	*276.43 ms*	*26.98 s*
Spatial aggregation	*14.65 mJ*	*75*	*268.62 ms*	*26.19 s*
Simple forwarding (without aggregation)	*15.5 mJ*	*100*	*236.74 ms*	*23.67 s*

To summarize this scheme, we can evaluate homogeneous aggregated forwarding with non- aggregated forwarding with respect to energy consumption, average end-to-end delay, overall delay for 100 packets and transmission count, illustrated in Table 4.4. This table reveals that homogeneous aggregation perform better than simple forwarding as there are slight deviations in energy and transmission count. However, in terms of delay, node consumes slightly higher amount of time in homogeneous aggregation with the difference of approximately 40 ms in temporal and 30 ms in spatial mode. Accordingly, this difference becomes 3.31 s in temporal and 2.52 s in spatial aggregations.

4.3 Experimental Results for S2: Heterogeneous Aggregation

For heterogeneous aggregation, the mechanism for source nodes in both temporal and spatial modes is same. Therefore in these illustrations, we only explain the spatial heterogeneous aggregation scheme from the relay node towards the sink. As shown in Figure 4.9, the relay node 4 has received packets from node 32 and 34 and store in its database. This source identification can be seen in both figures where *a:32* and *a:34* represents the associated source addresses. Two buffer *f* and *g* are assigned to store data from two different sources as temperature data packets are stored in *f* and humidity data packets are stored in *g*. Further, a request is initiated to send packet *f[1]* from temperature storage and *g[2]* from humidity storage to the sink node. As a result, node 4 concatenates these two packets and send towards node 5 which is the sink node. Figure 4.10 reveals the information when an aggregated frame containing one humidity and one temperature packet is received from node 4. After this reception, this frame is de-aggregated into two separate packets and saved until further processing is requested.

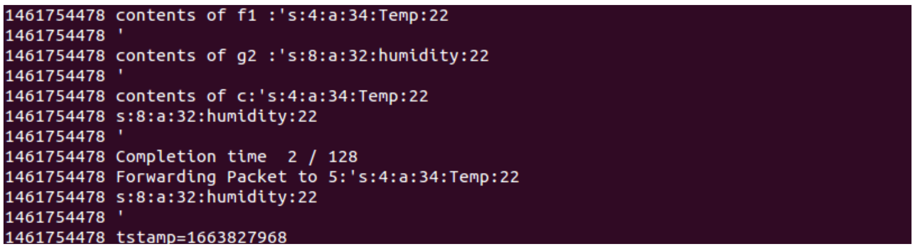
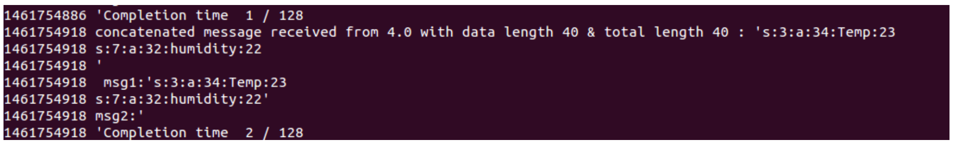

```
1461754478 contents of f1 :'s:4:a:34:Temp:22
1461754478 '
1461754478 contents of g2 :'s:8:a:32:humidity:22
1461754478 '
1461754478 contents of c:'s:4:a:34:Temp:22
1461754478 s:8:a:32:humidity:22
1461754478 '
1461754478 Completion time  2 / 128
1461754478 Forwarding Packet to 5:'s:4:a:34:Temp:22
1461754478 s:8:a:32:humidity:22
1461754478 '
1461754478 tstamp=1663827968
```

Figure 4.9: Packet transmission from the relay node in the heterogeneous scheme for spatial aggregation.



```
1461754886 'Completion time  1 / 128
1461754918 concatenated message received from 4.0 with data length 40 & total length 40 : 's:3:a:34:Temp:23
1461754918 s:7:a:32:humidity:22
1461754918 '
1461754918  msg1:'s:3:a:34:Temp:23
1461754918 s:7:a:32:humidity:22'
1461754918 msg2:'
1461754918 'Completion time  2 / 128
```

Figure 4.10: Packet reception and de-aggregation at the sink node in the heterogeneous scheme for spatial aggregation.

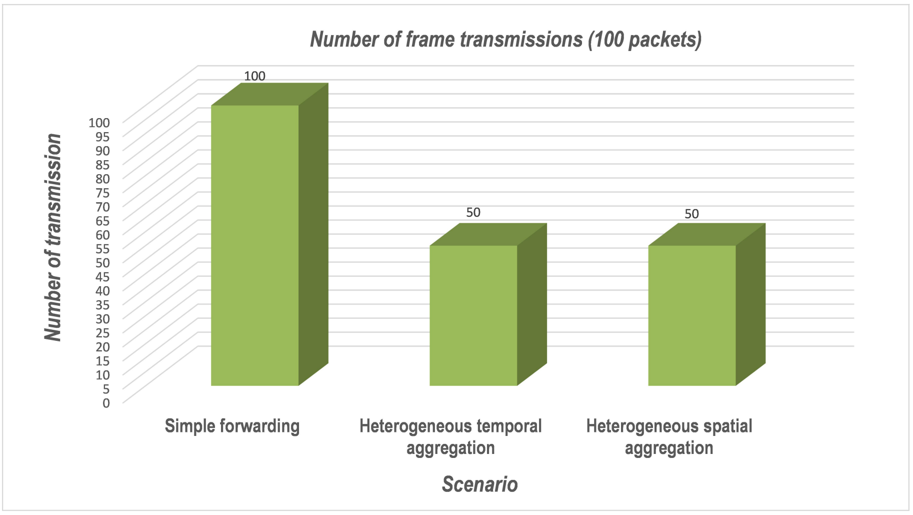


Figure 4.11: A comparison of frame transmission count from relay node between heterogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

4.3.1 Results for number of frame transmissions

From Figure 4.11, a comparison is made for transmissions counts from relay node between heterogeneous aggregation and simple forwarding. According to the working principle, described in Figure 4.9, relay node has received packets and stored in its memory. As we are assessing all the schemes on the basis of 100 packets transmission so we consider 100 packets in node's memory. For that reason, receiver can initiate request to send any available packet from the database, as shown in Figure 4.9 where we send *f[1]* and *g[2]*. Therefore, all 100 packets can be sent towards the sink within the specified selection. According to this principle, 50 frame transmissions, in any order, are required to deliver 100 packets to the sink. Accordingly, number of transmissions are reduced upto 50 %.

4.3.2 Results for end-to-end delay

From Figure 4.12, it can be revealed that average end-to-end delay for both spatial and temporal aggregation mode is same. It is due to the fact that delay associate with each stored packet in the memory does not vary with respect to source configuration. It is noticeable from Table 4.5 that overall end-to-end delay for a packet in heterogeneous aggregation is 260.49 which is lower than the delay in homogeneous. However, it is still higher than simple forwarding scenario as we have a difference of approximately 24 ms which was around 30 ms to 40 ms in case of homogeneous. This slightly larger delay in this scheme exists because there is a lot of processing involved in terms of database management. Again, this 260.49 ms involves 15.62 ms processing time at relay node. In contrast, sink only spends 7.81 ms which is 23.43 ms in the case of homogeneous. This is because sink is only dealing with one type of packet reception in heterogeneous.

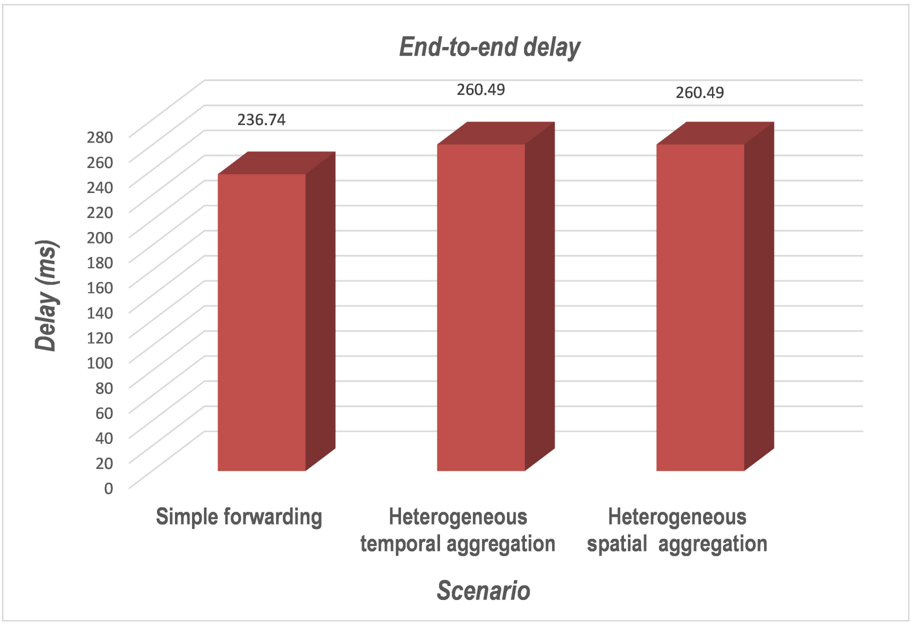


Figure 4.12: A comparison of average end-to-end delay for a single packet with respect to heterogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for a single packet.

Table 4.5: A comparison of average end-to-end delay between heterogeneous aggregation scheme vs simple forwarding for a single packet

Schemes	Delay at source node	Delay at relay node	Delay at sink node
Temporal aggregation	*235.55 ms*	*17.129 ms*	*7.81 ms*
Spatial aggregation	*235.55 ms*	*17.129 ms*	*7.81 ms*
Simple forwarding (without aggregation	*235.55 ms*	*1.184 ms*	*0 ms*

4.3.3 Results for overall delay

In Figure 4.12, end-to-end delay for both temporal and spatial aggregation is given which is 260.49 ms for both. From this observation, overall delay for 100 packets can be calculated by multiplying frame transmission delay with number of frame transmission for 100 packets. This accumulation is shown in Figure 4.13 where overall delay for spatial and temporal is 25.58 s which is 1.91 s higher than the overall delay of simple forwarding. As we described before that delay in homogeneous aggregation is always higher than simple forwarding because aggregation involves some processing delay at relay node which is added in the total delay.

4.3.4 Results for relay node energy

For heterogeneous aggregation scheme, a comparison is shown in Figure 4.14, where we can see that energy consumption in heterogeneous is lower than simple forwarding. However, these energy values are slightly lower than those in homogeneous mode. It is due to the fact that in homogeneous scheme, we have two types of transmissions for aggregated and non-aggregated. Transmission energy for one aggregated

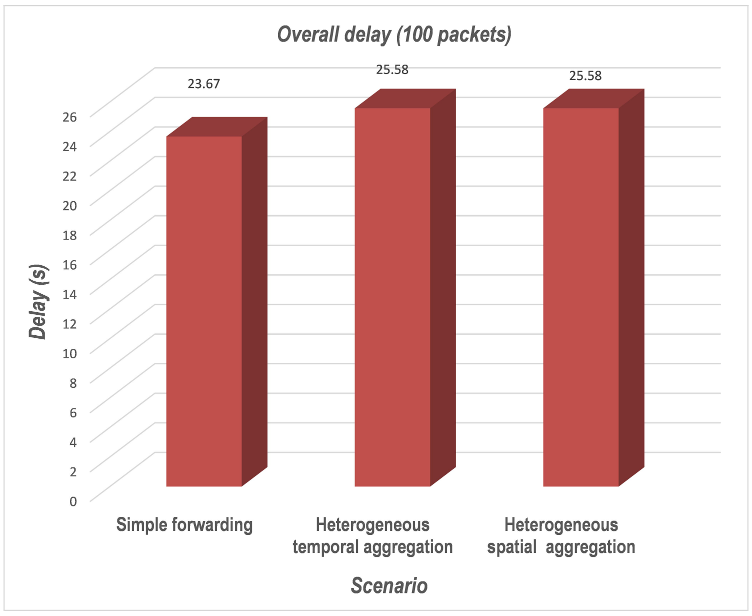


Figure 4.13: A comparison of overall delay for 100 packets with respect to heterogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission.

frame is little higher than a normal packet. However, in this case as we are sending all 100 packets through aggregated frames so overall energy is smaller than homogeneous but much less than simple forwarding. Additionally, database management also consumes slightly higher than simple aggregation. All the specific details can be observed from Table 4.6 where values for transmission and reception energy are given which shows that normal forwarding consumes 15.5 mJ while heterogeneous aggregation consumes 12.85 mJ per 100 packets which saves 2.65 mJ in total.

Table 4.6: A comparison of relay node's energy consumption between heterogeneous aggregation scheme and simple forwarding for 100 packets

Schemes	**Etx (1 frame)**	**Erx(1 frame)**	**Total Energy (100 packets)**
Temporal aggregation	*0.091 mJ*	*0.083 mJ*	*12.85 mJ*
Spatial aggregation	*0.091 mJ*	*0.083 mJ*	*12.85 mJ*
Simple forwarding (without aggregation	*0.072 mJ*	*0.083 mJ*	*15.5 mJ*

4.3.5 Scheme summary

This scheme examine the principle of heterogeneous aggregation with non- aggregated forwarding with respect to energy consumption, average end-to-end delay, overall delay and transmission count. Table 4.7 provides a summary about the overall performance where it shows decreasing transmission counts significantly but also less energy consumption than in simple forwarding. Additionally, this scheme performs better than homogeneous in terms of end-to-end delay and overall delay management.

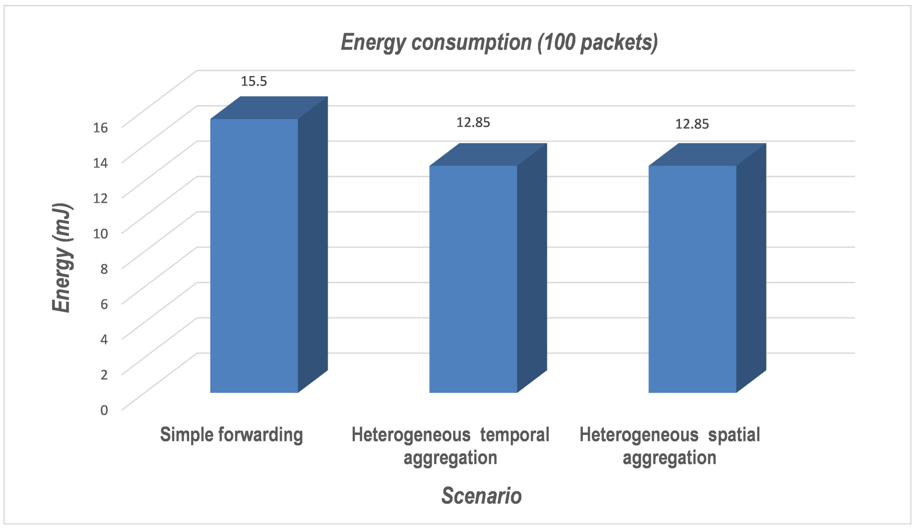


Figure 4.14: A comparison of average energy consumption of the relay node with respect to heterogeneous scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

Table 4.7: Overall comparison between heterogeneous aggregation scheme vs simple forwarding with respect to energy consumption, average end-to-end delay, frame transmission count and overall delay

Schemes	**Energy consumption (100 packets)**	**Transmission count (100 packets))**	**End-to-end delay (1 packet)**	**Overall delay (100 packets)**
Temporal aggregation	*12.85 mJ*	*50*	*260.49 ms*	*25.58 s*
Spatial aggregation	*12.85 mJ*	*50*	*260.49 ms*	*25.58 s*
Simple forwarding (without aggregation	*15.5 mJ*	*100*	*236.74 ms*	*23.67 s*

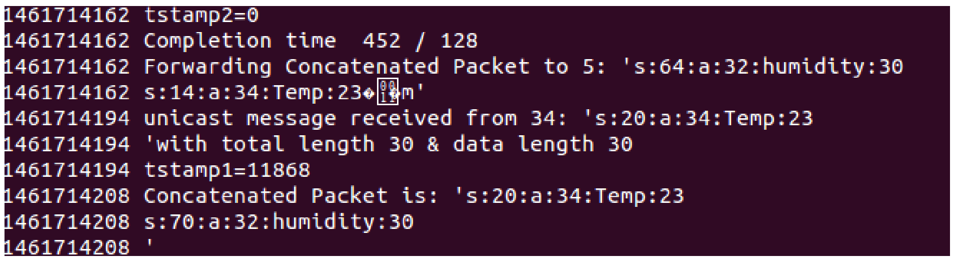
4.4 Experimental Results for S3 & S4: Variable Aggregation

This scheme is divided into two sub-schemes. Firstly, we have fixed frame-length aggregation scheme for spatial and temporal aggregation modes. In Figure 4.15, working of a relay node is shown where fixed frame-length spatial aggregation is implemented where relay node 4 is concatenating two packets, came from node 34 and node 32 and forwards towards node 5. In this figure, it can be clearly seen that only two packets are aggregated in a frame which has sequence number 20 for temperature and 70 for humidity.

Secondly, we have adaptive frame-length aggregation which concatenates upto the maximum size of the frame length. As we have frame size of 127 bytes so we only concatenated upto 3 packets to be on safe side. In Figure 4.16, working of the relay node is shown which is forwarding a frame containing 3 packets, received from node 26 and 32 . Similarly, Figure 4.17 illustrates a scenario where sink node 5 is receiving a frame and de-aggregating 3 packets in the form of 3 tokens. Where one token is still empty.

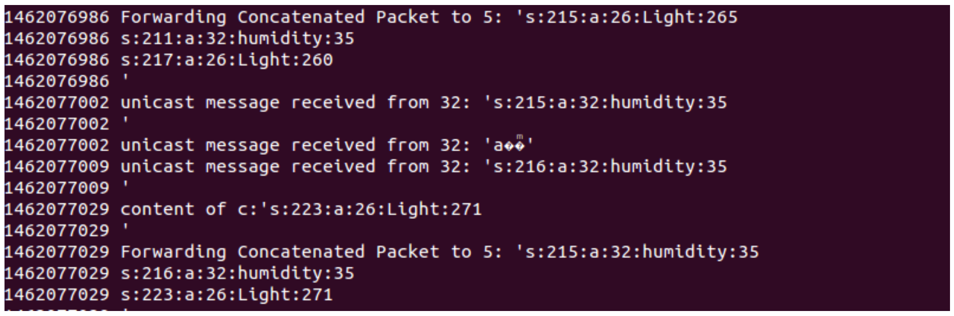
4.4.1 Results for number of frame transmissions

In terms of transmission counts, two schemes are observed. First, we start with fixed frame-length aggregation, shown in Figure 4.18. In this scheme, we fix the frame length size so that it can carry upto two packets which is most commonly used in all discussed schemes. Correspondingly, for 100 packets,



```
1461714162 tstamp2=0
1461714162 Completion time  452 / 128
1461714162 Forwarding Concatenated Packet to 5: 's:64:a:32:humidity:30
1461714162 s:14:a:34:Temp:23◆▯m'
1461714194 unicast message received from 34: 's:20:a:34:Temp:23
1461714194 'with total length 30 & data length 30
1461714194 tstamp1=11868
1461714208 Concatenated Packet is: 's:20:a:34:Temp:23
1461714208 s:70:a:32:humidity:30
1461714208 '
```

Figure 4.15: Packet transmission from the relay node in the fixed frame-length scheme for spatial aggregation.



```
1462076986 Forwarding Concatenated Packet to 5: 's:215:a:26:Light:265
1462076986 s:211:a:32:humidity:35
1462076986 s:217:a:26:Light:260
1462076986 '
1462077002 unicast message received from 32: 's:215:a:32:humidity:35
1462077002 '
1462077002 unicast message received from 32: 'a◆◆'
1462077009 unicast message received from 32: 's:216:a:32:humidity:35
1462077009 '
1462077029 content of c:'s:223:a:26:Light:271
1462077029 '
1462077029 Forwarding Concatenated Packet to 5: 's:215:a:32:humidity:35
1462077029 s:216:a:32:humidity:35
1462077029 s:223:a:26:Light:271
```

Figure 4.16: Packet transmission from the relay node in the adaptive frame-length scheme for spatial aggregation

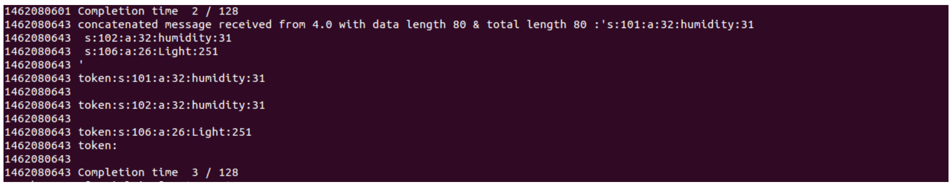
number of transmissions are reduced from 100 to 50. This scheme is following simple aggregation where relay is receiving 2 packets and formulates a frame of two packets. This scheme shows equal outcomes for both spatial and temporal aggregation modes.

In adaptive frame-length scheme, shown in Figure 4.19, we observe this scheme for maximum 3 packets. As in this scheme, relay node performs a periodic aggregation, regardless of the fact that which three packets are received and concatenated in the frame. Thus, average number of transmissions for 100 packets is reduced to 34 as last packet will be delivered in a separate transmission unit. This reduction of transmission count affords significance in the context of aggregation as transmission count are declined upto 66 %.

4.4.2 Results for end-to-end delay

As we have two different graphs of delay for two different schemes, so lets start with fixed frame-length aggregation, shown in Figure 4.20 which explains the end-to-end delay measurements for both spatial and temporal aggregation modes. In both modes, delay for source node remains same which is 235.55 ms. As Table 4.8 depicts the details about node delay where we can see that source nodes in both spatial and temporal have equal amount of delay which is 235.55 ms. Therefore, overall end-to-end delay for this scheme is 252.68 ms. Identically, relay and sink also consumes the same amount of time as spatial and temporal do not have prominent impacts on relay and sink node.

Now, moving towards adaptive frame-length aggregation, shown in Figure 4.21, end-to-end delay for both spatial and temporal aggregation is same. This scheme aggregates upto 3 packets into a frame periodically. However, if we compare this scheme with the fixed frame-length, it takes delay of 269.59 ms which is larger than fix frame-length scenario. This scheme produces higher delay as in case of 3 packets aggregation, relay node takes more time in terms of processing and transmission as packet length increases from 47 bytes to 87 bytes. Although, 3 packets can be accommodated within 70 bytes but we design frame size little bigger as if sequence number increases upto 1000 packets for 3 packets in one frame then 3 packets need an extra 6 bytes in terms of sequence number.



```
1462080601 Completion time  2 / 128
1462080643 concatenated message received from 4.0 with data length 80 & total length 80 :'s:101:a:32:humidity:31
1462080643  s:102:a:32:humidity:31
1462080643  s:106:a:26:Light:251
1462080643 '
1462080643 token:s:101:a:32:humidity:31
1462080643
1462080643 token:s:102:a:32:humidity:31
1462080643
1462080643 token:s:106:a:26:Light:251
1462080643 token:
1462080643
1462080643 Completion time  3 / 128
```

Figure 4.17: Packet reception and de-aggregation at the sink node in the adaptive frame-length scheme for spatial aggregation.

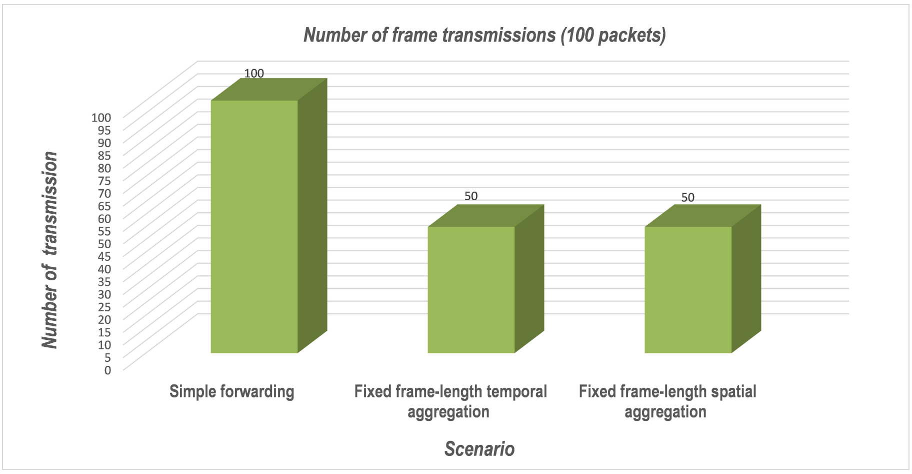


Figure 4.18: A comparison of frame transmission count from the relay node between fixed frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

4.4.3 Results for overall delay

Overall delay measurements are taken by using values of frame transmission from end-to-end delay calculations. In this scenario, we begin with fixed frame-length aggregation, shown in Figure 4.22 where we have overall delay of 24.8 s for both spatial and temporal. This delay is much less then homogeneous aggregation as total number of transmissions in homogeneous aggregation is 75, including 50 packet transmissions and 25 frame transmissions. However, in this case, we have only 50 frame transmissions. This fact significantly changes the overall delay computation. If we compare this scheme with the simple forwarding, we have a difference of 1.12 s for 100 packets.

Now, we observe the behaviour of delay in adaptive packet-length scheme where we have 25.74 s delay for 100 packets. Although, this delay value is also calculated using number of frame transmissions which is only 34 in this scheme for 100 packets. However, this delay value is still 0.94 ms higher than fixed frame-length aggregation due to the fact that frame is carrying 3 packets which affords a lot of transmission and processing as compared to fixed frame where we have 2 packets in a frame. This larger frame size leads to higher delay of 2.07 s when correlated with the simple forwarding.

4.4.4 Results for relay node energy

In order to evaluate relay node's energy consumption for this, we first start with the fixed frame aggregation, shown in Figure 4.24. As mentioned before that source node is consecutively generating packets to meet the period of aggregation at relay. Therefore, faster sensing and processing is required. Therefore, energy at source node is 0.091 mJ for both spatial and temporal. Consequently, overall energy consumption decreases from 15.5 mJ to 12.85 mJ. Accordingly, it consumes 2.65 mJ less energy then simple forwarding for 100 packets.

Figure 4.25 explains the scenarios for adaptive frame-length where relay node is receiving 3 packets from either different sources or the same source. The values for energy consumption depict this fact

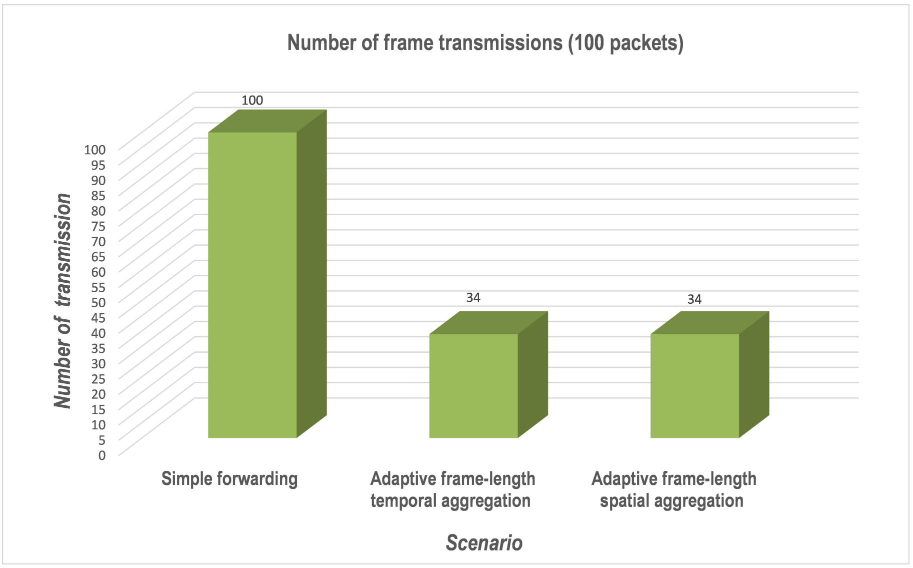


Figure 4.19: A comparison of frame transmission count from the relay node between adaptive frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

that source nodes in adaptive frame-length aggregation are consuming more than those in fixed frame-length situation due to much more sensing and processing in order to deliver 3 packets within a specified cycle. Although, energy for relay node almost remains same as aggregation is happening periodically. Nevertheless, overall energy values for this scheme is higher than fixed frame length scheme as frame transmission with 3 packets consumes more energy than a frame with 2 packets. Therefore, this scheme saves 1.54 mJ per 100 packets. Table 4.9 interprets the energy values for each scheme with specified aggregation mode for better understanding of the reader.

4.4.5 Scheme summary

This set of schemes provides significant results when we talk about number of frame transmissions. As Table 4.10 shows this fact that number of transmissions required to forward 100 packets is decreased to 50 in fixed frame-length aggregation and 34 in adaptive frame-length aggregation. However, delay remains higher due to larger frame size and processing. In addition, adaptive frame-length consumes more energy than fixed frame length.

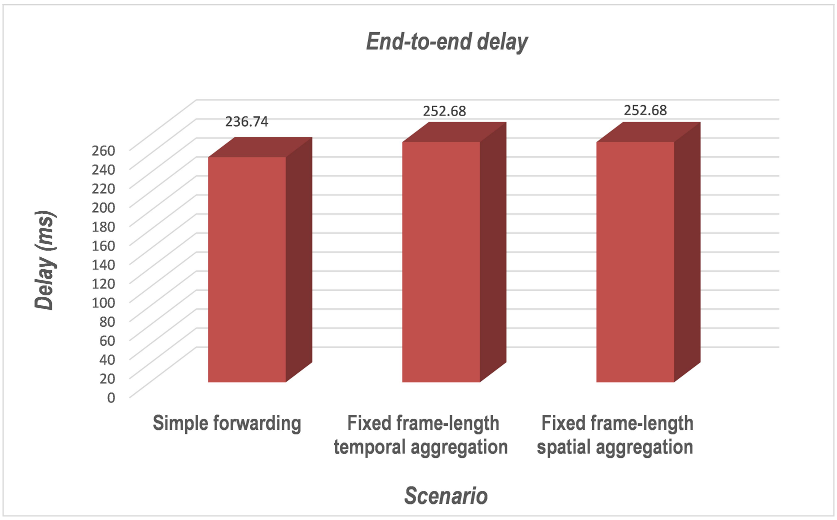


Figure 4.20: A comparison of average end-to-end delay for a single packet with respect to fixed frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for a single packet.

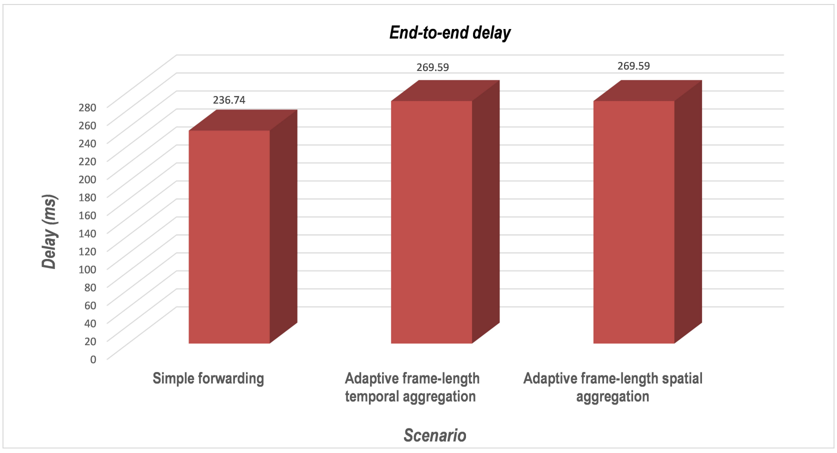


Figure 4.21: A comparison of average end-to-end delay for a single packet with respect to adaptive frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for a single packet.

4.5 Comparison of all Schemes

After presenting the results for all the scenarios, we analyse all the schemes in the form of a short summary to show an overall performance evaluation to our reader. We start with energy analysis, as shown in Figure 4.26. This figure reveals the fact that fixed frame-length and heterogeneous schemes perform better in terms of energy consumption as compared to the other aggregation schemes. Homogeneous scheme consumes much energy because it has to deal with two types of forwarding which requires a lot of processing power. Additionally, adaptive frame-length also consumes more energy than fixed frame-length as this scheme has to deal with the 3 packets aggregation into one frame. However, this scheme has the best efficiency for transmission count as adaptive frame technique accommodate more packets than any other scheme. Moving further, end-to-end delay in heterogeneous scheme is less than rest of the proposed schemes. Furthermore, this scheme provide more optimal number of transmissions than homogeneous aggregation. In the next section, we observe the packet loss ratio for each scheme with respect to varying distance.

Table 4.8: A comparison of average end-to-end delay between variable scheme for spatial and temporal aggregation vs simple forwarding for a single packet

Schemes	(Delay at source node)	(Delay at relay node)	(Delay at sink node)
Temporal aggregation (fixed frame-length)	*235.55 ms*	*9.314 ms*	*7.81 ms*
Spatial aggregation (fixed frame-length)	*235.55 ms*	*9.314 ms*	*7.81 ms*
Temporal aggregation (adaptive frame-length)	*235.55 ms*	*18.40 ms*	*15.62 ms*
Spatial aggregation (adaptive frame-length)	*235.55 ms*	*18.40 ms*	*15.62 ms*
Simple forwarding (without aggregation)	*235.55 ms*	*1.18 ms*	*0 ms*

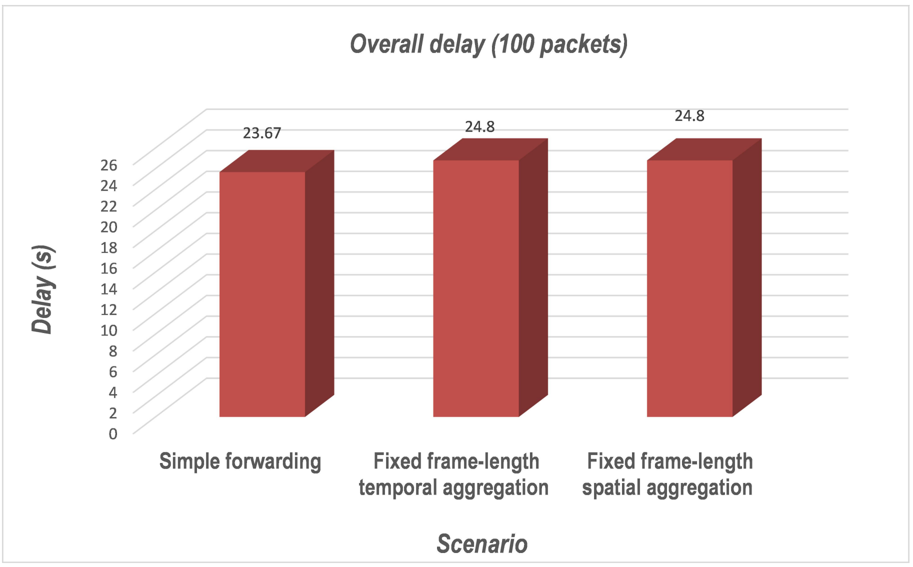


Figure 4.22: A comparison of overall delay for 100 packets with respect to fixed frame-length aggregation scheme for spatial and temporal aggregation vs non-aggregated transmission.

4.5.1 Packet Drop Rate

As we know this fact that packet drop increases with the increase in the distance between the nodes. We observe the packet drop rate by conducting a separate experiment in the outdoor environment. Our objective is to observe the impact of different aggregation schemes for wireless sensors motes by evaluating each scheme on the basis of packet drop rate. In this experiment, each scheme is performed with spatial aggregation approach. Therefore, we used 4 TelosB sensor motes, separated with measured distance. Among these four motes, 2 motes are used as source, one as relay and the other as sink. The distance between source to sink is incremented by 10 m in each iteration. Initially, the distance between each node is kept 10 m from each other in the outdoor environment. During this experiment, each node sends a

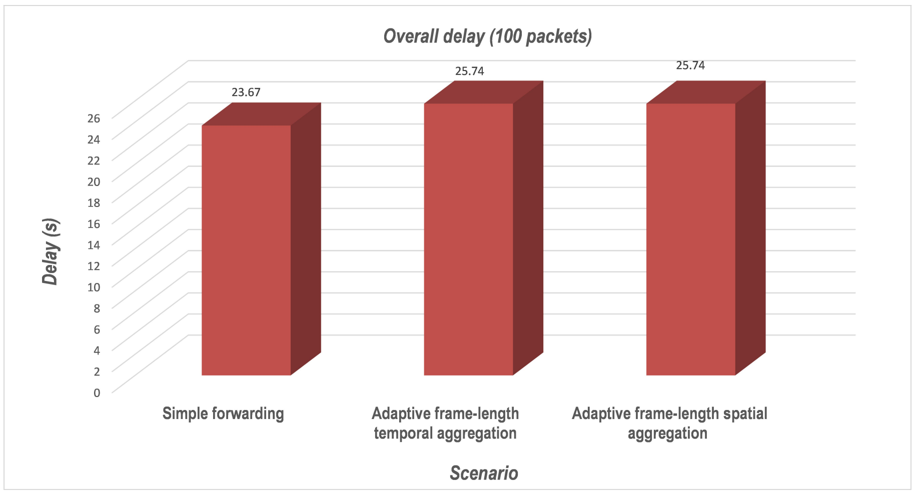


Figure 4.23: A comparison of overall delay for 100 packets with respect to adaptive frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission.

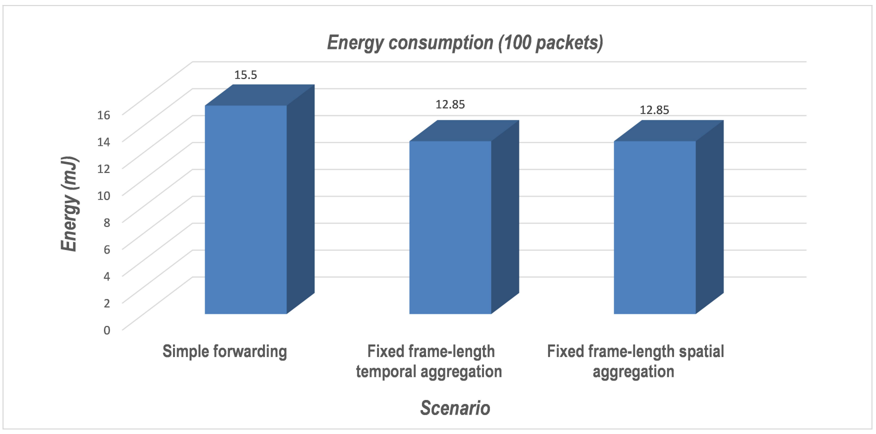


Figure 4.24: A comparison of average energy consumption of the relay node with respect to fixed frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

packet every 5 s and we conducted each iteration for 100 packets to observe the behaviour of the nodes for each scenario. The behaviour of each scheme is described in Figure 4.27 with respect to packet drop vs varying distances. In this figure, we compare the packet drop rate among different schemes with simple forwarding. Starting with the simple forwarding, we can see that at minimum possible distance, packet drop is negligible. However, in homogeneous and heterogeneous aggregation, it changes from 1% to 3%. Similarly, in fixed frame-length and adaptive frame length schemes, it lies between 2% and 3%. Moving further, at 20 m distance, simple forwarding changes a little. Nonetheless, trend changes in all aggregation schemes upto 7% except heterogeneous where it still remain constant at 3 %. At 30 m, there is a slight increment in packet drop for simple forwarding and heterogeneous when it goes to 6 % and 7 % respectively. On the other side, packet drop for homogeneous and adaptive frame length aggregation goes upto 13 %. Surprisingly, fixed frame-length rate is much higher then all aggregation schemes which goes upto 18 %. It is due to the fact, that this scheme is specifically bounded to aggregate two packets only, despite the fact that multiple packets have arrived in the meanwhile. This packet drop rate follows the same trend until we go upto 70 or 80 m. At 80 m, we have maximum packet drop recorded for each scheme. For instance, simple forwarding goes upto 55 % packet loss. Similarly, homogeneous, heterogeneous and adaptive frame-length aggregation have 59 %, 54 % and 53 % respectively. Again, fixed frame-length produces an unexpected rate of 67 % for 100 packets. We observe from these experiments, that aggregation

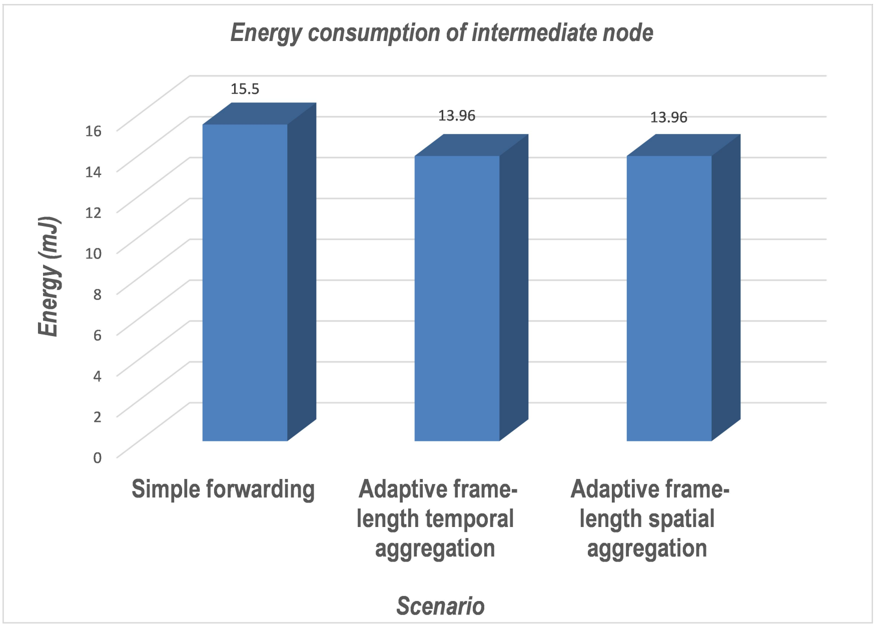


Figure 4.25: A comparison of average energy consumption of the relay node with respect to adaptive frame-length scheme for spatial and temporal aggregation vs non-aggregated transmission for 100 packets.

Table 4.9: A comparison of relay node's energy consumption between variable aggregation for spatial and temporal vs simple forwarding for 100 packets

Schemes	**Etx (1 packet)**	**Erx(1 packet)**	**Total Energy (100 packets)**
Temporal aggregation (fixed frame-length	*0.091 mJ*	*0.083 mJ*	*12.85 mJ*
Spatial aggregation (fixed frame-length	*0.091 mJ*	*0.083 mJ*	*12.85 mJ*
Temporal aggregation (adaptive frame-length)	*0.17 mJ*	*0.083 mJ*	*13.96 mJ*
Spatial aggregation (adaptive frame-length)	*0.17 mJ*	*0.083 mJ*	*13.96 mJ*
Simple forwarding (without aggregation)	*0.072 mJ*	*0.083 mJ*	*15.5 mJ*

Table 4.10: Overall comparison between variable aggregation vs simple forwarding with respect to energy consumption, average end-to-end delay, overall delay and frame transmission count

Schemes	**Energy consumption (100 packets)**	**Transmission count (100 packets))**	**End-to-end delay (1 packets)**	**Overall delay (100 packets)**
Temporal aggregation (fixed frame-length)	*12.85 mJ*	*50*	*252.68 ms*	*24.8 s*
Spatial aggregation (fixed frame-length)	*12.85 mJ*	*50*	*252.68 ms*	*24.8 s*
Temporal aggregation (adaptive frame-length)	*13.96 mJ*	*34*	*269.59 ms*	*25.74 s*
Spatial aggregation (adaptive frame-length)	*13.96 mJ*	*34*	*269.59 ms*	*25.74 s*
Simple forwarding (without aggregation)	*15.5 mJ*	*100*	*236.74 ms*	*23.67 s*

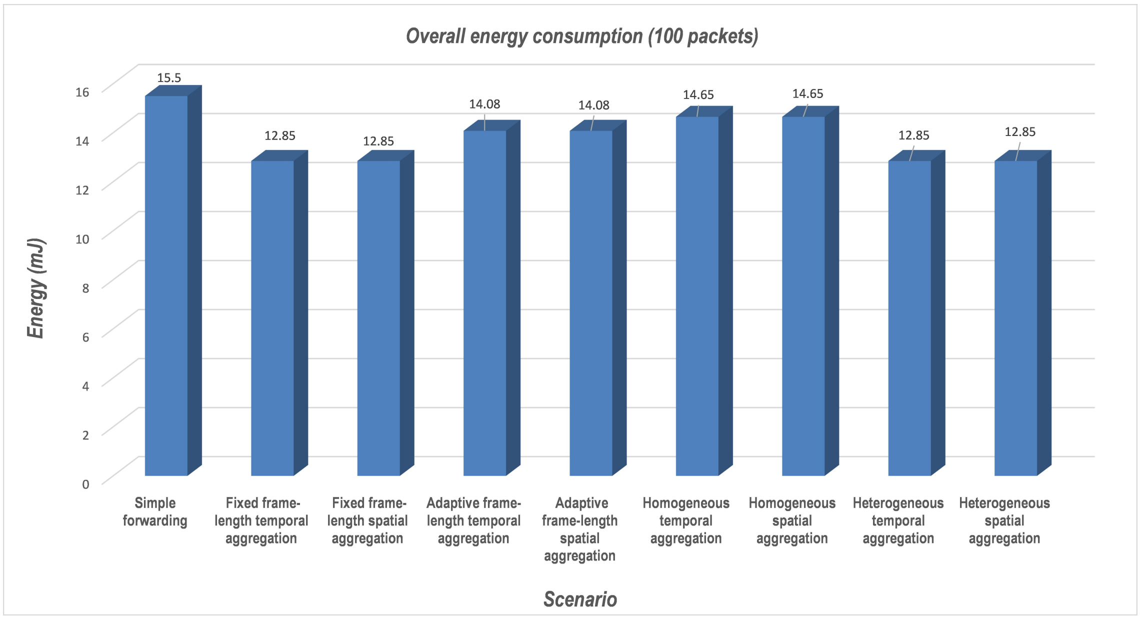


Figure 4.26: An overall comparison of energy consumption between simple forwarding vs all proposed schemes for spatial and temporal aggregation for 100 packets in the relay node. This calculation is done with respect to transmission count required for transmitting 100 packets.

schemes cause more packet drop than traditional forwarding. This is because packet aggregation at the relay nodes is taking more processing time than the simple forwarding. Hence, there are possibility of backlog packets at the relay nodes which increase packet drop probability at the relay nodes. In addition, TelosB sensing range is upto 100 m, but in our scenarios, nodes became unstable after 80 m. Therefore, we conduct these experiments within the range of 80 m to get precision and accuracy in the measurements.

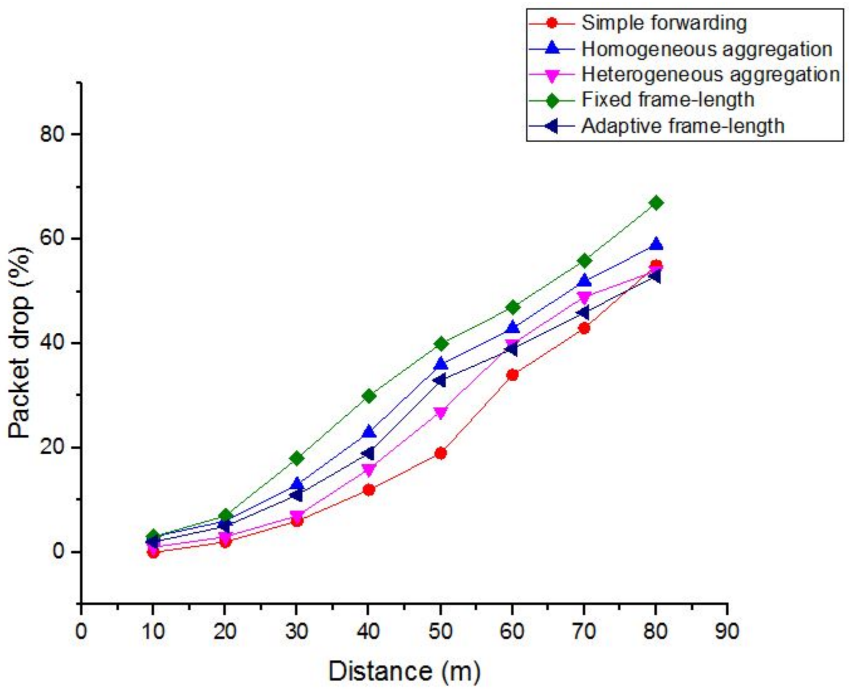


Figure 4.27: A comparison of packet drop rate between simple forwarding and the proposed aggregation schemes for 100 packets.

4.6 Packet Analysis

Packets analysis is performed by capturing a transmission using packet sniffing mechanism. This mechanism involves an interception and traffic logging through a network. Packet sniffers work by intercepting and logging network traffic via wired or wireless network interface. Additionally, the traffic can be analysed in the packet sniffing software which has access to its host computer. In wireless networks, packet sniffer usually captures traffic from one channel at a time and can intercept traffic from that channel accordingly. Once the traffic is logged, the raw data is then analysed by the software to interpret in the human readable form [2].

4.6.1 Sniffing devices and Sniffer 15.4

Sniffer 15.4 [3] is an android application which is used to capture 802.15.4 enabled frames over the air using a cell phone. It allows an integration of the device with other android phones or tablets [4]. Furthermore, packets are captured by an interception over the air and can be seen and exported to Wireshark for further exploration. To set up the sniffing hardware, we programme a TelosB mote with a special sniffing firmware. Once the sensor node is programmed with the firmware it is activated as a sniffing device. Afterwards, the sensor mote has to be connected with a USB OTG cable to view the raw data through the android application, illustrated in Figure 4.28.

4.6.2 Aggregation analysis

In our scenarios, we analyse fixed frame-length spatial aggregation scheme using sniffing tool and Wireshark. During this process, we deploy 4 sensor nodes including two sources, one relay and one sink. Afterwards, we capture some packets from different transmissions and store as raw data for further analysis.

[2]www.en.wikipedia.org/wiki/Packet_analyzer

[3]www.netsecurity.about.com/od/informationresources/a/What-Is-A-Packet-Sniffer.htm

[4]www.rtn.sssup.it/index.php/software/seed-eye-apps/sniffer-154

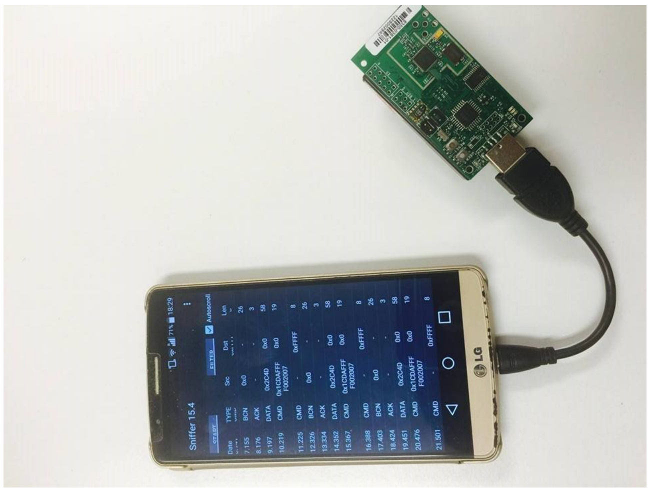
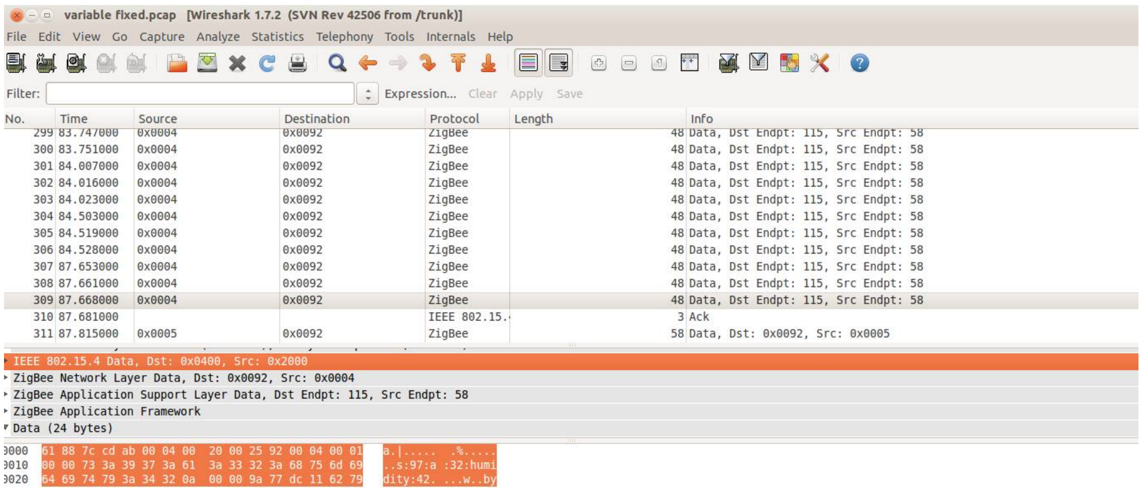


Figure 4.28: TelosB mote connected to android phone using a USB OTG cable.

Later, these packets are analysed using Wireshark to provide a clarification about aggregation. According to Wireshark analysis, packets are validated by checking their amended sequence number and source ID. From Figure 4.29, it can be seen that the captured frame contains humidity packet information with sequence number and node address(shown in red marked on the bottom right). If we look closer on the highlighted red bar, *(IEEE 802.15.4 Data Dst:0x0400,Src:0x2000)*, we can see the destination & source address of the frame.



variable fixed.pcap [Wireshark 1.7.2 (SVN Rev 42506 from /trunk)]

```
No.  Time      Source  Destination Protocol    Length Info
 299 83.747000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 300 83.751000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 301 84.007000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 302 84.016000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 303 84.023000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 304 84.503000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 305 84.519000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 306 84.528000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 307 87.653000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 308 87.661000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 309 87.668000 0x0004  0x0092      ZigBee          48 Data, Dst Endpt: 115, Src Endpt: 58
 310 87.681000                     IEEE 802.15.     3 Ack
 311 87.815000 0x0005  0x0092      ZigBee          58 Data, Dst: 0x0092, Src: 0x0005

▸ IEEE 802.15.4 Data, Dst: 0x0400, Src: 0x2000
▸ ZigBee Network Layer Data, Dst: 0x0092, Src: 0x0004
▸ ZigBee Application Support Layer Data, Dst Endpt: 115, Src Endpt: 58
▸ ZigBee Application Framework
▾ Data (24 bytes)

0000  61 88 7c cd ab 00 04 00  20 00 25 92 00 04 00 01   a.|.....  .%.....
0010  00 00 73 3a 39 37 3a 61  3a 33 32 3a 68 75 6d 69   ..s:97:a :32:humi
0020  64 69 74 79 3a 34 32 0a  00 00 9a 77 dc 11 62 79   dity:42. ...w..by
```

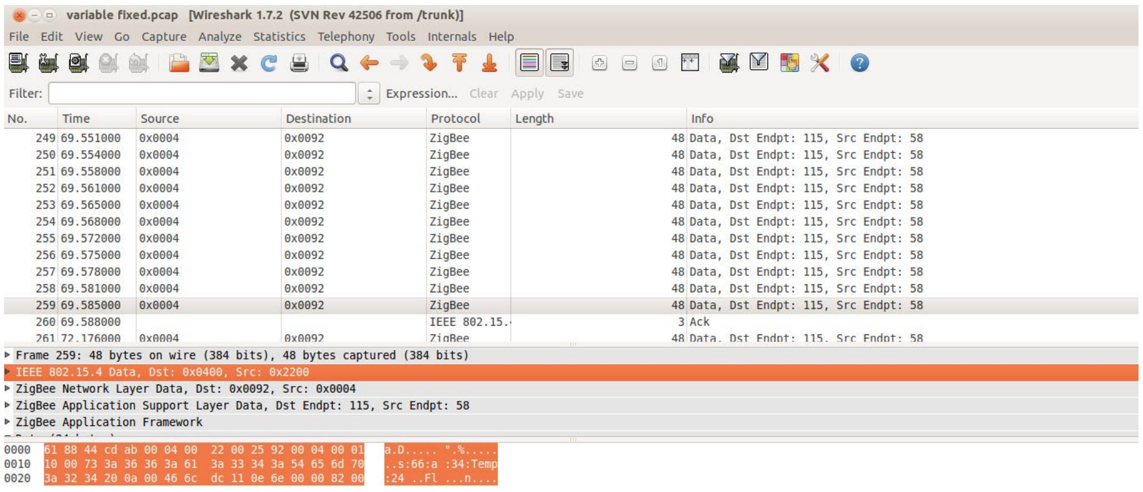
Figure 4.29: A captured packet of humidity during a transmission between source and relay.

These values are in the form of hexadecimal format which can be converted into decimal. Conversion of hexadecimal value to decimal can be performed using Eq 4.10. Accordingly, addresses for source and destination are converted using Eq. 4.10 and Eq. (4.11).

$$Destinationaddress = (04)_{16} = (4)_{10} \tag{4.10}$$

$$Sourceaddress = (20)_{16} = (32)_{10} \tag{4.11}$$

Therefore, we can also verify the source address of the packet from the content of the the captured packet as *a:32* shows that this packet is coming from node 32 towards the intended destination which is node 4.


variable fixed.pcap [Wireshark 1.7.2 (SVN Rev 42506 from /trunk)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time       Source   Destination  Protocol   Length  Info
 249 69.551000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 250 69.554000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 251 69.558000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 252 69.561000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 253 69.565000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 254 69.568000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 255 69.572000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 256 69.575000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 257 69.578000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 258 69.581000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 259 69.585000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 260 69.588000                        IEEE 802.15.    3 Ack
 261 72.176000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
▸ Frame 259: 48 bytes on wire (384 bits), 48 bytes captured (384 bits)
▸ IEEE 802.15.4 Data, Dst: 0x0400, Src: 0x2200
▸ ZigBee Network Layer Data, Dst: 0x0092, Src: 0x0004
▸ ZigBee Application Support Layer Data, Dst Endpt: 115, Src Endpt: 58
▸ ZigBee Application Framework
0000  61 88 44 cd ab 00 04 00  22 00 25 92 00 04 00 01   a.D..... ".%.....
0010  10 00 73 3a 36 36 3a 61  3a 33 34 3a 54 65 6d 70   ..s:66:a :34:Temp
0020  3a 32 34 20 0a 00 46 6c  dc 11 0e 6e 00 00 82 00   :24 ..Fl ...n....
```

Figure 4.30: A captured packet of temperature during a transmission between source and relay.

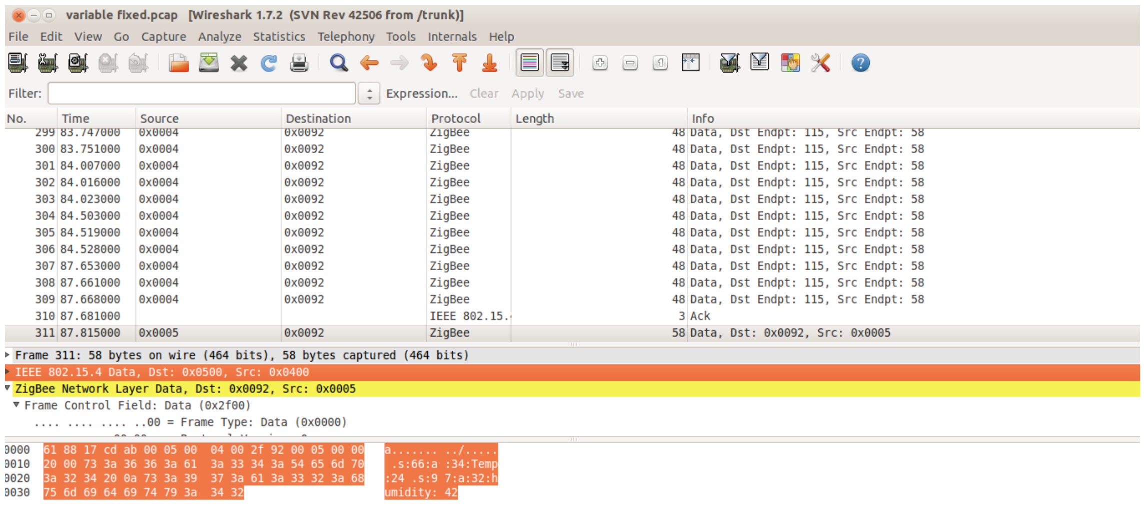
Moving further, from Figure 4.30, it can be revealed that the second captured packet contains temperature packet with some information about sequence number and node address (shown in red marked on the bottom right). Accordingly, if we look closer on the highlighted red bar, *(IEEE 802.15.4 Data Dst:0x0400,Src:0x2200)*, the source and the destination addresses can be easily observed. Similarly, conversion from hexadecimal value to decimal can be performed using Eq. (4.12) and Eq. (4.13).

$$Destinationaddress = (04)_{16} = (4)_{10} \tag{4.12}$$

$$Sourceaddress = (22)_{16} = (34)_{10} \tag{4.13}$$

This conversion clarifies this fact that packet is generated from source node which is 34 and being transferred towards node 4.

From Figure 4.30, it can be seen that the captured packet is actually an aggregated frame which contains information two packets, temperature and humidity, with their sequence number and node addresses, shown in red marked on the bottom right. Therefore, if we look closer on the highlighted red bar *(IEEE 802.15.4 Data Dst:0x0500,Src:0x0400)*, we can see the destination & source address of the frame. These source and destination addresses are converted in Eq. (4.14) and Eq. (4.15).


variable fixed.pcap [Wireshark 1.7.2 (SVN Rev 42506 from /trunk)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time       Source   Destination  Protocol   Length  Info
 299 83.747000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 300 83.751000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 301 84.007000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 302 84.016000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 303 84.023000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 304 84.503000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 305 84.519000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 306 84.528000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 307 87.653000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 308 87.661000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 309 87.668000  0x0004   0x0092       ZigBee         48 Data, Dst Endpt: 115, Src Endpt: 58
 310 87.681000                        IEEE 802.15.    3 Ack
 311 87.815000  0x0005   0x0092       ZigBee         58 Data, Dst: 0x0092, Src: 0x0005
▸ Frame 311: 58 bytes on wire (464 bits), 58 bytes captured (464 bits)
▸ IEEE 802.15.4 Data, Dst: 0x0500, Src: 0x0400
▾ ZigBee Network Layer Data, Dst: 0x0092, Src: 0x0005
  ▾ Frame Control Field: Data (0x2f00)
     .... .... .... ..00 = Frame Type: Data (0x0000)
0000  61 88 17 cd ab 00 05 00  04 00 2f 92 00 05 00 00   a....... ../.....
0010  20 00 73 3a 36 36 3a 61  3a 33 34 3a 54 65 6d 70    .s:66:a :34:Temp
0020  3a 32 34 20 0a 73 3a 39  37 3a 61 3a 33 32 3a 68   :24 .s:9 7:a:32:h
0030  75 6d 69 64 69 74 79 3a  34 32                     umidity: 42
```

Figure 4.31: A captured aggregated frame during transmission between relay and sink. This frame contains packets of humidity and temperature.

$$Destinationaddress = (04)_{16} = (4)_{10} \tag{4.14}$$

$$Sourceaddress = (05)_{16} = (5)_{10} \tag{4.15}$$

Therefore, the frame is generated from node 4 and forwarded towards node 5. Accordingly, if we look closer on the content inside of the frame, shown in red marked on the bottom right, we can see the aggregated frame in which two previously captured packets are aggregated inside a frame which confirms that aggregation has been performed at relay node.

Chapter 5

Discussions

Although aggregation is an engaging technique for multi-hop WSNs, it is also quite challenging to deploy in TelosB sensor motes due to the firm nature of TelosB such as lack of synchronization, resource limitation and longer delays. Specifically, inflexible Contiki platform and its hard-line library functions are quite demanding when it comes to manipulation of codes according the the requirements of the task. However, TelosB mote and Contiki platform are most commonly used in research community for last few years. Regardless of their nature, we adopt this combination to deploy aggregation and to analyse the outcomes.

In this chapter, we try to discuss results obtained from each scheme and compare these results with the questions mentioned in the problem statement in order to explain the contact of each finding with the thesis requirements. Furthermore, we provide some attempts which are partialy implemented during the implementation phase. Lastly, we discuss the adverse and unexpected outcomes and pinpoint the possible limitations and lacks.

5.1 Result Discussions

Our research revolves around the problem questions mentioned in Chapter 1 such as how to design and implement an aggregation scheme which is more suitable for TelosB motes in order to reduce number of transmissions. Further, in terms of evaluation which scheme performs better when compared on the basis of performance parameters like end-to-end delay, energy consumption, packet drop, etc. Here, we evaluate each scheme on the basis of results and conclude that how much we are able to meet our specified requirements of this thesis. Starting with the number of transmissions, described in the first research question, it can be revealed that we are able to find a working solution for this problem in the form of our proposed aggregation schemes. Each scheme reduces number of transmission nominally from 25 % to 65 % and each schemes is deployed using TelosB sensor motes in multi-hop fashion. In this context, design and implementation of these schemes exactly match with the first requirement of this thesis. Moving further, second task is associated with implementation of these schemes at relay node. As we described earlier, relay node is the main focus throughout this process as it has to deal with all the traffic going towards the sink which eventually creates an energy hole. Due to this reason, all aggregation schemes are deployed at the relay node in order to afford a working solution to resolve the energy hole problem. Furthermore, second research question demands an aggregation approach which consumes less energy and time as compared to the non-aggregated forwarding. We observe this fact from the experimental results from Chapter 4 that all proposed schemes are better then simplel forwarding when evaluated on the basis of energy consumption. Nonetheless, delay is slightly higher in aggregation schemes due to presence of large processing and frame transmission time. However, this difference in delay values is minor enough that it can be resolved by better synchronization among all nodes. Third question requires an adaptive mechanism to evaluate aggregation with respect to the performance parameters. In performance parameters, energy consumption is an important factor which decides whether the respective scheme is considerable or not. Therefore, this factor requires a precise measurement technique which can satisfy all the standards. To fulfil this requirement, we adopted a measurement approach which is based on oscilloscope and other physical equipments like resistors, multimeter, etc. Our measured values for transmission and reception currents are verified by [20]. To measure the processing delay at each node, we adopted a very useful technique, described in [24], which is based on system clock. Using this library, a clock is initialized at the start of every process. Therefore, the total processing time is calculated by computing the difference between initial and final value. Considering this fact, our selected measurement techniques are adaptive and provide precise analysis for any kind of scenario in the sensor networks. In terms of packet drop, as we discussed in Section 4.5.1, simple forwarding performs better as compared to each aggregation scheme. This is due to the fact that inter-departure time between each outgoing packet from the source node is kept very less to meet the aggregation requirements. Accordingly, some packets are dropped while reaching at the relay node as it is busy with other receptions. Thus, packet drop ratio can be minimized in aggregation schemes by increasing the inter-departure time between outgoing packets. Moving further, following sections describe the advantages of these aggregation schemes in terms of performance parameters.

5.1.1 Advantages of aggregation schemes

As the results show that aggregation schemes overall perform better then simple forwarding in terms of number of transmissions and energy consumption for relay node which enhances the overall performance of the system. It is due to the fact that MAC principle requires an ACK after each successful transmissions. So, by implementing packet aggregation, number of transmissions are reduced because less number of ACKs are received at relay node from the sink. For that reason, proposed schemes are excellent in terms of reducing traffic overhead.

Advantages of homogeneous packet-selection

Packet aggregation at the relay node does not necessarily mean that every packet should be concatenated. Instead, packets can be selected according to the service type to participate in this execution. Due to this fact, homogeneous selection of packets proved itself as an excellent technique which can be implemented in any scenario which involve any preference of data. In addition, non-aggregated packets are not just discarded but can be simply forwarded without taking part in aggregation. This whole mechanism saves a lot of transmission count as we can see that number of frame transmissions are reduced upto 25 %. Furthermore, delay is not reduced as compared to simple forwarding however this factor is not only associated with homogeneous aggregation but also to the lack of synchronization. Every node requires some start up time to initialize its processing and in case of aggregation, relay has to wait for atleast two packets to build a frame. Energy consumption is also smaller as compared to simple forwarding though the difference is not that prominent. This energy consumption is slightly higher because it involves two types of forwarding, aggregated and non-aggregated which exert a lot of processing when performed in parallel.

Advantages of relay-database

TelosB sensor motes have enough memory of 10 kB to store a certain amount of packets which is used in heterogeneous approach. As numerical results show in Figure 4.11 that this scheme reduces number of transmissions upto 50 %. This prominent decline leads towards less energy consumption then homogeneous scheme. Additionally, in case of packet loss, this mechanism can be used to recover lost packets by performing an identification process based on the sequence numbers. Further, a retransmission algorithm can be used to retrieve the lost packets from the database and redelivered to the sink node. Nevertheless, delay is 25.58 s which is still higher than simple forwarding as storing and retrieving process requires a lot of processing. However if we compare this approach with homogeneous scheme, it performs better in terms of delay management, energy consumption and number of transmissions.

Advantages of adaptive packet length

Typically, all schemes are designed to accommodate two packets in a frame while performing aggregation. Considering this fact, this scheme offers a lot of flexibility in terms of frame size. This interesting technique has a significant impact on transmission count which is decreased upto 65 % of total required transmissions. These transmission count can be reduced further in case of 4 packets per frame. However, packet size is a critical factor here as it should not exceed the size of a frame' payload which is 128 Bytes.

5.2 Other endeavours

In order to provide more flexibility and efficiency in aggregation schemes, there are several factors which can be included inside the aggregation techniques. In this section, we present some of the key efforts which are performed during the employment of packet aggregation to make these schemes more effective and resourceful. However, on account of some convincing aspects, these efforts are not fully successful in terms of implementation.

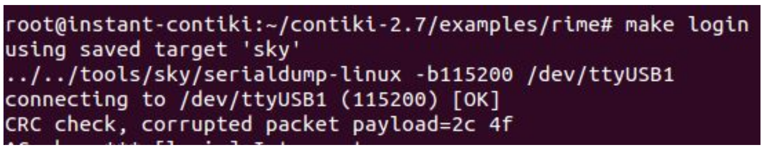
5.2.1 Compression

Compression is a mechanism which is widely used & applied in various communication applications. The main focus of this mechanism is suppressing of data size to minimize memory usage. Although, a lot of compression algorithms exist such as *LZ77* , Burrows Wheeler transform , Huffman coding etc. However, most of these algorithms are not suitable in resource constraint devices such as TelosB mote. We tried to implement *s-lzw* compression algorithm in our aggregation model which is deployed at the source and compresses an outgoing packet. After receiving frame, sink node has to decompress the received frame after de-aggregation in oder to retrieve the original packet. After the deployment of s-lzw, compression is performed successfully, as shown below. This algorithm is performed at the source node as relay is already dealing with a lot of processing due to aggregation. Nevertheless, decompression at the sink node is still uncertain. This is because while source node is sending compressed packets towards relay node, it is really difficult for a relay to distinguish the content of packet in case of homogeneous aggregation as compression algorithms use their own dictionary words instead of sending plain text. Moreover, after aggregation, sink has to de-aggregate first and then decompress the packets separately. Therefore, this mechanism requires more time and research to develop according to our requirements.

```
sprintf(s,"s:%d:a:%d:Temp:%d \n",
send_packets,rimeaddr_node_addr,get_temp());
write_buffer[30]= s;
slzw_compress( 30, 15);
packetbuf_copyfrom(lzw_output_file_buffer,15);
```

5.2.2 CRC

CRC is an approach which is used in communication devices in order to identify error in the received packets. In this approach, some check values are assembled with outgoing data packet from the source node. On the receiver side, CRC error counter inspects the content of received packet, compares the assembled check value and approves the data accordingly. If the counter finds any dissimilarity, it means the received packet is corrupted and further operation is needed [26]. Accordingly, in Zigbee enabled TelosB sensor motes, last two bytes in every packet is FCS which is actually used for CRC calculation. In our aggregation schemes, CRC is enabled in a way that as it can identify the corrupted packet and notify the receiver. CRC in Contiki is implemented by immobilizing the MAC functionality and copying the payload of corrupt packets without moving through the stack. By applying CC2420_CONF_AUTOACK = 0, auto ACK is disabled so there is no more hardware ACK for the sensor. These lines are inserted into Contiki/platform/sky/contiki-conf.h.



```
root@instant-contiki:~/contiki-2.7/examples/rime# make login
using saved target 'sky'
../../tools/sky/serialdump-linux -b115200 /dev/ttyUSB1
connecting to /dev/ttyUSB1 (115200) [OK]
CRC check, corrupted packet payload=2c 4f
```

Figure 5.1: CRC check for corrupted packets at relay node.

```
#define NETSTACK_CONF_MAC nullmac_driver
#define NETSTACK_CONF_RDC nullrdc_noframer_driver
#define NETSTACK_CONF_FRAMER framer_nullmac
#define CC2420_CONF_AUTOACK 0
```

Additionally, following lines are also added into Contiki/core/dev/cc2420.c. By adding these lines, the CRC error counter notifies receiver that the received packet is corrupted, as shown in Figure 5.1. As last two bytes in Zigbee packets are hard coded for CRC. Accordingly printf("%02x ") can print the payload part which contains the check values to pinpoint the actual error.

```
int i;
printf("CRC check, corrupted packet payload=");
for (i=0;i<len;i++) {
printf("%02x ", ((uint8_t *) buf)[i]);
}
printf("\n");
```

From Figure 5.1, it can be seen that the received packet is corrupted and CRC error counter is printing the last two bytes of CRC from the payload of the corrupted packet.

However, there are certain limitations which affect the whole concept of CRC in this platform. While we disable the MAC functionalities, by allowing nullmac_driver to work instead of ContikiMAC, ACK has to be disabled. Just as, the network stack implemented in Contiki is not the same as typical OSI five layers model. It is because instead of having only MAC layer between the PHY and the Network layers, there are 3 different layers: Framer, Radio Duty-Cycle (RDC) and Medium Access Control (MAC). While the network layers are operated through ETSTACK_FRAMER, NETSTACK_RDC and NETSTACK_MAC. By disabling ACK, receiver can not be notified about this corrupted reception of packet so retransmission is not possible by this way. Additionally, if we try to add explicit ACK, the retransmission from the source node is quite challenging in our scenario. It is because assume that source is constantly sensing data packets and sending towards relay while packets are discarded after each transmission. Therefore, if there is any request of retransmission from the relay, source can not send the same sample again as it is not stored in its memory. However, this issue can be resolved by creating another database at source node such as we created in homogeneous aggregation scheme which can store a certain amount samples. Nevertheless, this solution obviously will affect the synchronization and delay of whole network.

5.3 Limitations

Now, we present certain limitations which are very influential while dealing with TelosB motes using Contiki platform. Despite the fact that we managed to complete our task requirements while having these limitations, these factors some how limit the performance and slow down the implementation process.

- Synchronization is a key factor which affects the performance of any aggregation scheme in sensor node. While implementing aggregation schemes using TelosB motes, it is a confronting task to adjust timer for each sensor node according to the requirements of each scheme. While sensing and data processing are performed simultaneously inside the node which insists to provide a concurring organisation to avoid potentially longer delays and packet drops.

- TelosB motes are resource-limited in order to achieve optimization in size, cost and energy consumption. Accordingly, this resource limitation affects the implementation process as most of the compression algorithms requires huge RAM, for instance, when we apply algorithm *LZ77* which can not be implemented inside the mote as it requires atleast 50 kB of memory to process. Therefore, in-node *LZ77* compression in TelosB motes is not an easy task as it requires some specific algorithm with the processing power of less then 10 kB.
- Although, reliability is achieved by sending ACK after every successful transmission, however, in case of packet drop, retransmission is needed to recover the dropped packets . The retransmission mechanism requires another database at source node to keep record of every outgoing packet for certain amount of time which affects the inter-departure time for each packet.
- Contiki is based on nesC which is a subset of C programming language. However, nesC is very inflexible in terms of library support as it does does not support some libraries. For instance, traditional C language provides a variety of libraries for the implementation of retransmission, CRC calculation, timestamps. However, nesC needs some enhancements and improvements for the library support in order to remove the compatibility issues.

Chapter 6

Conclusions and Future Work

This project broadened our knowledge and augmented a lot of experience and competence about packet-aggregation in WSNs. The implementation of this technique in our thesis is a compliment in terms of skills and expertise while working in the environment of WSNs. It allowed us to improve our knowledge and experience about programming in Contiki environment. At this moment, we are very much aware of all the hurdles and complications about in-network resources and how much packet-aggregation is effective while implemented in various topologies. Moreover, why Contiki is favourable over TinyOS while dealing with TelosB sensor motes. Therefore, our attempts and efforts in this context will be quite helpful for us in the eventual. However, more research and exploration is needed in order to improve the quality of this work in near future.

This chapter composed of a brief abbreviation of our work and leads towards a beginning of a new discussion about some novel aspects provided by this report. Eventually, it concludes with some convincing remarks about this report.

6.1 Conclusions

In the course of this research, we executed several experiments and examined several topologies by implementing various aggregation schemes. We evaluated these schemes by sending packets from the source node, performing aggregation at the relay and forwarding in the form the of an aggregated frame towards the sink. This assessment is based on multiple performance parameters such as energy consumption, end-to-end delay, overall delay, number of frame transmissions and packet drop. In Chapter 1, we provided a brief introduction about this topic with specified research questions and the chosen approach to answer these questions. Further, Chapter 2 revolves around the background of this topic and affords an analysis of previously proposed techniques. In addition, this chapter provides information about the tools which we adopted to built the platform to execute different schemes. In Chapter 3, we discussed network design, scheme specifications and protocol employment to present an obvious illustration of our proposed aggregation model. In chapter 4, we contributed a comprehensive scrutiny of the results, acquired by different experiments for each scheme with respect to the performance parameters. This analysis helped us to classify all these aggregation schemes according to these parameters. Chapter 5 contains a self assessment of the acquired results in comparison with the specified research questions in Chapter 1. Furthermore, this chapter explained the execution of some unsuccessful attempts such as implementation of CRC and compression. These efforts were made during the implementation process to facilitate these aggregation schemes. Throughout these investigations, we focused on the relay node as this node has to deal with all the traffic going towards the sink which eventually produces an energy hole due to immature death of relay node. According to our practice throughout this thesis, we noticed that aggregation performs better in terms of energy consumption and transmission count. However, end-to-end delay is not decreased at all even the aggregation is implemented. Traditionally, researchers believe that packet aggregation should decrease the delay due to the less number of transmissions. However in TelosB, sensing and processing at the source node produces an extra delay due the fact that it involved some extra processing while the processing ability of TelosB's CPU is very weak. Additionally, the processing delay in TelosB includes an extra 210 ms sensing delay which is mentioned as a wake up call to activate the sensing process. In homogeneous scheme, energy consumption is higher as it includes 50 % of simple forwarding mechanism which excessively increases the number of transmissions. Heterogeneous scheme performs comparatively better in terms of transmission count as this number is reduced to 50 %. Furthermore, adaptive packet-length scheme limits transmission count to 34 % which clarifies the efficiency of this scheme. In each section, we gave a brief overview of every bit od information which is somehow associated with each experiment. We hope that provided results and analysis give sufficient knowledge and understanding to our reader about aggregation in WSNs.

6.2 Contributions

In terms of our contributions, all the proposed schemes are implemented and tested on the basis of performance parameters However, a detailed summary is given below.

- We started with the aggregation of packets containing raw data coming from a single source to examine the behaviour of packet aggregation in TelosB motes. Later, we aggregated the real data like temperature, humidity and light in a same topology to make these schemes more practical. Additionally, we formulated a mechanism to insert sequence number and source ID inside the payload part to identify each packet and compute the packet loss at the sink node.
- Aggregation is furnished with a design and implementation for spatial and temporal topologies.
- We designed and implemented aggregation according to the service type such homogeneous and heterogeneous schemes. In homogeneous scheme, we managed to perform aggregation according to different service types. Further, in heterogeneous, we systematized a database which stores a certain amount of received packets at relay node's history to provide an option to aggregate any stored packet from the history in case of request initiation Additionally, we designed and implemented variable scheme which contains fixed frame-length and adaptive frame-length aggregation to make this technique more flexible for any kind of scenario. Accordingly, these methods provide working solution for both spatial and temporal aggregation.
- All designed and implemented schemes are tested and evaluated through a set of experiments on the basis of performance parameters. In addition to all these contributions, there are some ineffective attempts which are partially implemented and can be used as a starting point for future investigations.

6.3 Future Topics

After going through all the experiments and research, we found some points which can be an interesting way to explore this topic in more advance manner. First, we examined this protocol with a simple and experimental way, considering only 4 or 5 nodes. However, this topology can be increased to more nodes to observe the efficiency of packet aggregation on higher scale which may lead towards more precise and improved implementation of these schemes.

Second, as these schemes are used for non-confidential packets where no privacy is required. As, in case of confidential and secure transmission, encryption can be implemented in these schemes in the form of concealed packet aggregation to provide privacy and security. Additionally, the standard packet size in case of TelosB is 128 bytes + 48 bytes for payload and header part respectively. Nonetheless, on the assumption of larger packets, standard frame length can be an obstacle while accommodating multiple packets inside an aggregated frame. Thus, compression can be used at the relay node to squeeze the outgoing packets before adding into a frame. This approach seems an acceptable practice in consideration of achieving more beneficial results of aggregation.

IoT indicates the interdependence of every single device which is capable of affording pervasive knowledge and resources in the modern world. This relationship consists of a shared system which includes every single object from computers, smart sensors, actuators, RFID, WSNs, digital media and even the people themselves. Therefore, IoT has appeared as a promising technique which provides a novel solution to enhance the quality of every involved parameter in our life [43]. The fundamental components of IoT includes sensing which is an important chunk of IoT [34]. In this context, sensing is actually a collection and transmission of data from different sources inside the wireless network towards the the sink node. Therefore, sink can process the gathered data according the requirement of the task. The concept of IoT can be used in sensor networks to make the mechanism of sensing and transmission more intelligent and quick. Additionally, communication is also an important feature in IoT, which connects sensing and computation together to acquire desired outcomes. Traditionally, IoT contains typical communication mechanisms which follow conventional packet transmission form source towards the sink. Therefore, the concept of packet aggregation in WSNs can be associated with IoT as it provides certain improvement in terms of energy efficiency and resource conservation. The proposed aggregation methods such as homogeneous, heterogeneous and variable scheme focus on dynamic frame length and variable number of transmission according to the structure and requirement of the protocol. These proposed schemes can be effectively utilized to reduce the expenditures per packet like end-to-end delay, energy and transmission count. Conclusively, we hope that our proposed schemes and contributions will help the future pioneers to obtain an excelling intuition inside the IoT environment for advance exploration of this topic.

Bibliography

[1] Chang JH, Tassiulas L. Energy conserving routing in wireless ad-hoc networks. In INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE 2000 (pp. 22-31). IEEE.

[2] Kalpakis K, Dasgupta K, Namjoshi P. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks. Computer Networks. 2003 Aug 21;42(6):697-716.

[3] Mhatre V, Rosenberg C. Design guidelines for wireless sensor networks: communication, clustering and aggregation. Ad hoc networks. 2004 Jan 31;2(1):45-63.

[4] He W, Nguyen H, Liu X, Nahrstedt K, Abdelzaher T. iPDA: an integrity-protecting private data aggregation scheme for wireless sensor networks. In Military Communications Conference, 2008. MILCOM 2008. IEEE 2008 Nov 16 (pp. 1-7). IEEE.

[5] Dasgupta K, Kalpakis K, Namjoshi P. An efficient clustering-based heuristic for data gathering and aggregation in sensor networks. In Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE 2003 Mar 20 (pp. 1948-1953). IEEE.

[6] Lee M, Wong VW. An energy-aware spanning tree algorithm for data aggregation in wireless sensor networks. In Communications, Computers and signal Processing, 2005. PACRIM. 2005 IEEE Pacific Rim Conference on 2005 Aug 24 (pp. 300-303). IEEE.

[7] Ding M, Cheng X, Xue G. Aggregation tree construction in sensor networks. In Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th 2003 Oct 6 (pp. 2168-2172). IEEE.

[8] Yang G, Xiao M, Zhang S. Data aggregation scheme based on compressed sensing in wireless sensor network. In Information Computing and Applications 2012 Sep 14 (pp. 556-561). Springer Berlin Heidelberg.

[9] WaltenegusDargie CP. Fundamentals of wireless sensor networks. A John Wiley and Sons Ltd., Publication. 2010.

[10] Narmada A, Rao PS. Zigbee based WSN with IP connectivity. In Computational Intelligence, Modelling and Simulation (CIMSiM), 2012 Fourth International Conference on 2012 Sep 25 (pp. 178-181). IEEE.

[11] Bala Krishna M, Vashishta N. Energy efficient data aggregation techniques in wireless sensor networks. In Computational Intelligence and Communication Networks (CICN), 2013 5th International Conference on 2013 Sep 27 (pp. 160-165). IEEE.

[12] Zechinelli M JL, Bucciol P, Vargas-Solar G. Energy aware data aggregation in wireless sensor networks. In Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on 2011 Feb 28 (pp. 1-5). IEEE.

[13] Jose J, Manoj Kumar S. Energy efficient recoverable concealed data aggregation in wireless sensor networks. In Emerging Trends in Computing, Communication and Nanotechnology (ICE-CCN), 2013 International Conference on 2013 Mar 25 (pp. 322-329). IEEE.

[14] Guntupalli L, Martinez-Bauset J, Li F Y, Weitnauer MA. Aggregated packet transmission in duty-cycled WSNs: modeling and performance evaluation. IEEE Transactions on Vehicular Technology doi: 10.1109/TVT.2016.2536686 (pp.1-1)

[15] Ambekar C, Lakhani G, Shah K, Bhanushali K. Energy efficient Data Aggregation in M-SPIN. In Intelligent Systems and Control (ISCO), 2015 IEEE 9th International Conference on 2015 Jan 9 (pp. 1-5). IEEE.

[16] Schiller JH. Mobile communications. Pearson Education, Publication 2003.

[17] Chen Z, Lin C, Wen H, Yin H. An analytical model for evaluating IEEE 802.15. 4 CSMA/CA protocol in low-rate wireless application. In Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on 2007 May 21 (pp. 899-904). IEEE.

[18] Gajjar S, Choksi N, Sarkar M, Dasgupta K. Comparative analysis of wireless sensor network motes. In Signal Processing and Integrated Networks (SPIN), 2014 International Conference on 2014 Feb 20 (pp. 426-431). IEEE.

[19] Chipcon AS. CC2420 datasheet-2.4 GHz IEEE 802.15. 4/ZigBee-Ready RF Transceiver (Rev. B). Chipcon AS. 2007.

[20] Datasheet, TelosB. "Crossbow Inc." 2013.

[21] Reusing T. Comparison of operating systems tinyos and contiki. Sens. Nodes-Operation, Netw. Appli.(SN). 2012 Aug;7.

[22] Dunkels A. The contikimac radio duty cycling protocol.

[23] Dunkels A, Österlind F, He Z. An adaptive communication architecture for wireless sensor networks. In Proceedings of the 5th international conference on Embedded networked sensor systems 2007 Nov 6 (pp. 335-349). ACM.

[24] Hassan R, Qamar T. Asymmetric-key cryptography for contiki.

[25] Mahmood A, Imran S. Energy consumption in wireless sensor networks: real life and simulation based measurements, IKT-508 on 2015. UIA

[26] Support.motioneng.com, CRC Error Counters, 2016. [Online]. Available: `http://support.motioneng.com/technology/synqnet/crc\_err\_ctr.htm`. [Accessed: 05- May-2016].

[27] Osterlind AD, Dunkels A. Contiki programming course: Hands-on session notes. Swedish Institute of Computer Science, Siena. 2009 Jul.

[28] He W, Liu X, Nguyen H, Nahrstedt K, Abdelzaher T. Pda: Privacy-preserving data aggregation in wireless sensor networks. In INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE 2007 May 6 (pp. 2045-2053). IEEE.

[29] Tsitsipis D, Dima SM, Kritikakou A, Panagiotou C, Gialelis J, Michail H, Koubias S. Priority handling aggregation technique (PHAT) for wireless sensor networks. InEmerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on 2012 Sep 17 (pp. 1-8). IEEE.

[30] Jain A, Gruteser M, Neufeld M, Grunwald D. Benefits of packet aggregation in ad-hoc wireless network (Doctoral dissertation, University of Colorado).

[31] Krishnamachari B, Estrin D, Wicker S. The impact of data aggregation in wireless sensor networks. In Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on 2002 (pp. 575-578). IEEE.

[32] Contiki.sourceforge.net, Contiki 2.6: File List, 2016. [Online]. Available: `http://contiki.sourceforge.net/docs/2.6/files.html`, [Accessed: 05- May- 2016].

[33] Patil NS, Patil PR. Data aggregation in wireless sensor network. In IEEE International Conference on Computational Intelligence and Computing Research 2010 Dec 28 (pp. 1-6).

[34] Al-Fuqaha A, Guizani M, Mohammadi M, Aledhari M, Ayyash M. Internet of things: A survey on enabling technologies, protocols, and applications. Communications Surveys & Tutorials, IEEE. 2015 Nov 18;17(4):2347-76.

[35] Cui J, Valois F. Data aggregation in wireless sensor networks: compressing or forecasting?. In Wireless Communications and Networking Conference (WCNC), 2014 IEEE 2014 Apr 6 (pp. 2892-2897). IEEE.

[36] Lindsey S, Raghavendra C, Sivalingam KM. Data gathering algorithms in sensor networks using energy metrics. Parallel and Distributed Systems, IEEE Transactions on. 2002 Sep;13(9):924-35.

[37] Xiang L, Luo J, Vasilakos A. Compressed data aggregation for energy efficient wireless sensor networks. In Sensor, mesh and ad hoc communications and networks (SECON), 2011 8th annual IEEE communications society conference on 2011 Jun 27 (pp. 46-54). IEEE.

[38] Dunkels A, Grönvall B, Voigt T. Contiki-a lightweight and flexible operating system for tiny networked sensors. In Local Computer Networks, 2004. 29th Annual IEEE International Conference on 2004 Nov 16 (pp. 455-462). IEEE.

[39] Benini L, Farella E, Guiducci C. Wireless sensor networks: Enabling technology for ambient intelligence. Microelectronics journal. 2006 Dec 31;37(12):1639-49.

[40] Polastre J, Szewczyk R, Culler D. Telos: enabling ultra-low power wireless research. In Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on 2005 Apr 15 (pp. 364-369). IEEE.

[41] Xiang L, Luo J, Rosenberg C. Compressed data aggregation: Energy-efficient and high-fidelity data collection. IEEE/ACM Transactions on Networking (TON). 2013 Dec 1;21(6):1722-35.

[42] Koubaa A, Alves M, Tovar E. A comprehensive simulation study of slotted CSMA/CA for IEEE 802.15. 4 wireless sensor networks. In 5th IEEE International Workshop on Factory Communication Systems 2006 Jun 28 (pp. 183-192). IEEE.

[43] Xia F, Yang LT, Wang L, Vinel A. Internet of things. International Journal of Communication Systems. 2012 Sep 1;25(9):1101.

Appendixes

In this chapter, we explain the technical configuration by providing a brief user manual for better understanding of the reader. In second section, we provide our programming codes for each scheme which are described in Chapter 4.

Appendix A

User manual

Before starting this manual, we expect that reader is already familiar with the Linux environment and has basic knowledge about Contiki and TelosB sensor motes. This knowledge will be helpful to get familiar with the commands and the environment. This manual is divided into two sections.

1. Instant Contiki installation and set-up.
2. Configuration and compiling a Contiki application.

Instant Contiki installation and set-up

This process contains downloading and installation of Instant Contiki and VMware. Many sources are available to download Instant Contiki, however, below mentioned source [1] is the one we used. So download the latest version of Instant Contiki from:

```
http://sourceforge.net/projects/contiki/files/Instant%20Contiki/
```

This downloaded file is an Ubuntu supported virtual image which can be booted into VMware. To download VMware, follow the link below:

```
https://www.vmware.com/products/player/playerpro-evaluation.html
```

When installation is finished, follow these steps:

1. Go to Home and select " Create New Virtual Machine".
2. Select "Install disc image file (iso).
3. Browse to the directory where you saved the Instant Contiki image file.
4. Run the installation and wait until the process is finished.
5. Run the virtual machine and password for the root user is "user".

Before running Contiki on hardware, two basic steps are required.

[1] www.contiki-os.org/start.html

- The required version for MSP430 toolchain is 4.7 so make sure that the version is upgraded. However, in case you are not sure, run the command *“InstallMSP430-4.7toolchain”* in the terminal from Home folder. This installation required internet connection.
- After this, you need to install *“gcc-msp430”* which is available at *“Ubuntu Software Center”*. Just type *“MSP430”* and select *“gcc-msp430”* from the given list.

After the successful execution of all these steps, now you can connect the TelosB motes with the Contiki. Examples required to understand the simple unicast and broadcast communication are given in the following directory:

“$HOME/contiki - 2.7/examples/rime$”.

Configuration and compiling a Contiki application

After performing all the steps mentioned in first section of this manual, you are able to install the Instant Contiki and your platform is ready to compile any Contiki application. Connect the TelosB mote to the USB port of the PC. Make sure that device is properly connected by checking the green icon on the top right corner showing *”memsic future devices”*. After making the connection, follow the steps below [2]:

1. Open the terminal and go to the above mentioned directory which contains all the examples. This can be done by typing following command.

   cd contiki/examples/rime

2. Before selecting any example, make sure that you are compiling this as a root user. Type the following command and password.

   sudo -s
   Password: user

3. Choose any example which you want to compile at this stage. For instance, if you want to implement simple unicast, simply type the example using following command.

   make TARGET=sky example-unicast.upload

4. To see the compiled output, type the *” make login”* command when the loading process is finished.
5. To see the compiled output with timestamps, type the *” make serialveiw”* command when the loading process is finished.

make TARGET=sky savetarget is another command which is used to build a Makefile with respect to the new platform. This Makefile.target is situated in the directory where your example.c is located.

Example codes can be run on Ubuntu without even installing Instant Contiki. Nevertheless, in most cases, it gives an error which can be associate with the installation of UISP tool kit [3]. The UISP tool kit can be installed by typing *apt-get install usip* in the terminal. This solution works for most of the sensor motes. However, in case of MIB510 programmer boards which are used for MicaZ, it may not work. so try to adopt the following procedure in case of MicaZ.

[2] www.anrg.usc.edu/contiki/index.php/Contiki_build_system
[3] www.wsnmagazine.com/step-by-step-method-of-writing-contiki-programs/

1. First, download the UISP file from

   `http://azadeha.at.ifi.uio.no/uisp.tar.gz`

2. Access the downloaded files from the Contiki shell and apply the successive commands.
   # tar -xvzf uisp.tar.gz # cd uisp # ./bootstrap # ./configure # make # make # sudo make install

3. Follow the directory where you saved the example file by typing
   contiki-2.4/examples/rime

4. Afterwards, *example.c* can be compiled in Cooja simulator by typing
   make TARGET=cooja example-multihop.cooja

Consequently, we hope that this user manual will help the reader to understand the procedure of initial installation and configuration.

Appendix B

In this section, we are providing codes for source node, relay node and sink node with respect to homogeneous scheme.

Homogeneous spatial and temporal aggregation

Source node

```

#include "contiki.h"
#include "net/rime.h"
#include "net/rime/timesynch.h"
#include "dev/button-sensor.h"
#include "dev/leds.h"
#include "net/queuebuf.h"
#include <stdio.h>
#include "net/packetbuf.h"
#include <string.h>
#include "dev/sht11-sensor.h"
#include "dev/light-sensor.h"
#include "net/rime/timesynch.h"
#include "sys/clock.h"

/*---------------------------------------------------------------------------*/

PROCESS(example_unicast_process, "Example unicast");
AUTOSTART_PROCESSES(&example_unicast_process);
 static struct unicast_conn uc;
 static rimeaddr_t addr;
 static int send_packets;
 static clock_time_t start_time;
 static float frac;

   static void
 recv_uc(struct unicast_conn *c, const rimeaddr_t *from)

{
   printf("ACK received from %d: \n",
        from->u8[0]);
   leds_toggle(LEDS_BLUE);
```

```
}

/*-------------------------------------------------------------------------------*/
static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;
static int send_packets;
*-------------------------------------------------------------------------------*/
                              SENSING
*-------------------------------------------------------------------------------*/
  static int
get_temp(void)
{
  return ((sht11_sensor.value(SHT11_SENSOR_TEMP) / 10) - 396) / 10 ;
}

  static int
get_light(void)

{
  return 10 * light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC) / 7;
}

static int get_humidity(void)
{
  return   (((0.0405*sht11_sensor.value(SHT11_SENSOR_HUMIDITY)) - 4) + ((-2.8 *
    0.000001)*(pow(sht11_sensor.value(SHT11_SENSOR_HUMIDITY),2))));
}
PROCESS_THREAD(example_unicast_process, ev, data)
{
  PROCESS_EXITHANDLER(unicast_close(&uc);)
    PROCESS_BEGIN();
  unicast_open(&uc, 146, &unicast_callbacks);
  SENSORS_ACTIVATE(sht11_sensor);
  SENSORS_ACTIVATE(light_sensor);
  while(1) {
    static struct etimer et;
    rimeaddr_t addr;
    etimer_set(&et, CLOCK_SECOND*8);
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    /* Delay Calculation */
    clock_init();
    start_time = clock_time();
    leds_on(LEDS_BLUE);
    leds_off(LEDS_RED);

*-------------------------------------------------------------------------------*/
                    PACKET FORMATION
*-------------------------------------------------------------------------------*/

    send_packets=send_packets+1;
    char t[30];
    sprintf(t,"Humidity:%d%:s:%d:a:%d:  \n",get_humidity(),send_packets,
    rimeaddr_node_addr);
    packetbuf_copyfrom(t, 30);
    addr.u8[0] = 4;
    addr.u8[1] = 0;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
      unicast_send(&uc, &addr);
      leds_off(LEDS_BLUE);
      leds_on(LEDS_RED);
      printf("unicast message sent to %d.%d: '%s'\n",
          addr.u8[0], addr.u8[1],(char *)packetbuf_dataptr());
      packetbuf_clear();
      printf(" with data length %d & total length %d with header length %d \n",
  packetbuf_datalen(),packetbuf_totlen(),packetbuf_hdrlen());
      char s[30];
      sprintf(s,"Temp:%d:s:%d:a:%d \n",get_temp(),send_packets,rimeaddr_node_addr);
      packetbuf_copyfrom(s, 30);
      addr.u8[0] = 4;
      addr.u8[1] = 0;
      if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
        unicast_send(&uc, &addr);
        packetbuf_clear();
        printf("unicast message sent to %d:  '%s' \n",
            addr.u8[0],(char*)packetbuf_dataptr());
        printf(" with data length %d & total length %d with header length %d \n",
  packetbuf_datalen(),packetbuf_totlen(),packetbuf_hdrlen());
        packetbuf_clear();
      }
      etimer_set(&et, CLOCK_SECOND*10);

      PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

      send_packets=send_packets+1;
      char q[30];
      sprintf(q,"Light:%d:s:%d:a:%d \n",get_light(),send_packets,rimeaddr_node_addr);
      packetbuf_copyfrom(q, 30);
      addr.u8[0] = 4;
      addr.u8[1] = 0;
      if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
        unicast_send(&uc, &addr);
        leds_off(LEDS_BLUE);
        leds_on(LEDS_RED);
        printf("unicast message sent to %d:  '%s' \n",
            addr.u8[0],(char*)packetbuf_dataptr());
        printf(" with data length %d & total length %d with header length %d \n",
  packetbuf_datalen(),packetbuf_totlen(),packetbuf_hdrlen());
        packetbuf_clear();
        printf("Completion time  %lu / %u \n", (unsigned long)clock_time() -
  start_time, CLOCK_SECOND);

        packetbuf_clear();
      }
    }
  }
  PROCESS_END();
}

/*---------------------------------------------------------------------------*/
                    FOR SPATIAL (2ND SOURCE)
/*---------------------------------------------------------------------------*/

PROCESS(example_unicast_process, "Example unicast");
AUTOSTART_PROCESSES(&example_unicast_process);

//static char a[1000], b[1000], c[1000];
static struct unicast_conn uc;
static rimeaddr_t addr;
static int send_packets;
//static int z[1000];

/*---------------------------------------------------------------------------*/
  static void
recv_uc(struct unicast_conn *c, const rimeaddr_t *from)
{

  printf("ACK received from %d: \n",
      from->u8[0]);
  leds_toggle(LEDS_BLUE);
}

/*---------------------------------------------------------------------------*/
static const struct unicast_callbacks unicast_callbacks = {recv_uc};
static struct unicast_conn uc;
static int send_packets;
  static int
get_temp(void)
{
  return ((sht11_sensor.value(SHT11_SENSOR_TEMP) / 10) - 396) / 10 ;

}

  static int
get_light(void)
{
  return 10 * light_sensor.value(LIGHT_SENSOR_PHOTOSYNTHETIC) / 7;
}

PROCESS_THREAD(example_unicast_process, ev, data)
{

  PROCESS_EXITHANDLER(unicast_close(&uc);)
    PROCESS_BEGIN();
  unicast_open(&uc, 146, &unicast_callbacks);
  SENSORS_ACTIVATE(sht11_sensor);
  SENSORS_ACTIVATE(light_sensor);
  while(1) {
    static struct etimer et;
    rimeaddr_t addr;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
      unicast_send(&uc, &addr);
      packetbuf_clear();
      printf("unicast message sent to %d:with total %d length '%s' \n",
          addr.u8[0],packetbuf_totlen(),(char*)packetbuf_dataptr());
    }

    etimer_set(&et, CLOCK_SECOND*6);

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    send_packets=send_packets+1;
    char t[30];
    sprintf(t,"s:%d:a:%d:Light:%d \n",send_packets,rimeaddr_node_addr,get_light());
    packetbuf_copyfrom(t, 30);
    addr.u8[0] = 4;
    addr.u8[1] = 0;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
      unicast_send(&uc, &addr);
      packetbuf_clear();
      printf("unicast message sent to %d:'%s'\n",
          addr.u8[0],(char*)packetbuf_dataptr());
    }
  }
  PROCESS_END();
}
```

Listing 6.1: Source node (spatial and temporal)

Relay node

```
/*------------------------------------------------------------------------
```

```
                              RECEIVING  AND  AGGREGATION
*-------------------------------------------------------------------------------------------------------*/
  static  void
recv_uc(struct  unicast_conn  *c,  const  rimeaddr_t  *from)

{
  if  (f[0]  ==  '\0')
  {
    x  =  0;
    clock_init();
    start_time  =  clock_time();
    leds_on(LEDS_BLUE);
    leds_off(LEDS_RED);
    printf("unicast  message  received  from  %d:  '%s'with  total  length  %d  \n",
        from->u8[0],(char  *)packetbuf_dataptr(),packetbuf_totlen());
    (char  *)packetbuf_copyto(f);
    if   (from->u8[0]==  32){
      if  (p[0]  ==  '\0'){
        strcpy(p,  f);
        else{

          strcpy(q,  f);

          packetbuf_clear();
        }
      }
      if   (from->u8[0]==  34){

        (char  *)packetbuf_copyto(s);

        strcpy(r,  s);
      }
    }
    strcpy(f,  "")  ;
    if  (p[0]  !=  '\0'  &&  q[0]  !=  '\0'  )  {
      c=strcat(p,q);
      strcpy(z,  c);
      printf("Concatenated  Packet  is:  '%s'\n",
          (char  *)z);
      packetbuf_clear();
      x  =  1;
      strcpy(p,  "");
      strcpy(q,  "");
      strcpy(c,  "");
    }

/*-------------------------------------------------------------------------------------------------------*/
                           FORWARDING  PART
*-------------------------------------------------------------------------------------------------------*/
    if  (z[0]  !=  '\0'  )  {
      static  struct  etimer  et5;
      etimer_set(&et5,  CLOCK_SECOND*9);
      addr.u8[0]  =  5;
      packetbuf_copyfrom((char  *)z,50);

      if(!rimeaddr_cmp(&addr,  &rimeaddr_node_addr))
      {
        unicast_send(&uc,  &addr);
        printf("Forwarding  Concatenated  humidity  Packet  to  %d:  '%s'  with  total  length
     %d\n"  ,addr.u8[0],(char  *)packetbuf_dataptr(),packetbuf_totlen());
        strcpy(z,  "");
        packetbuf_clear();
      }
    }

    if (r[0] != '\0' ) {

      static struct etimer et7;
      etimer_set(&et7, CLOCK_SECOND*7);
      addr.u8[0] = 5;
      packetbuf_copyfrom((char *)r,30);
      if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr))
      {
        unicast_send(&uc, &addr);
        printf("Forwarding Temperature Without Aggregation to %d: '%s' with total
    length %d\n" ,addr.u8[0],(char *)packetbuf_dataptr(),packetbuf_totlen());
        printf("Completion time %lu / %u \n", (unsigned long)clock_time() -
    start_time , CLOCK_SECOND);
        packetbuf_clear();
        strcpy(r, "");
      }
    }
    packetbuf_clear();
  }
  addr.u8[0] = 4;
  if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr)) {
    unicast_send(&uc, &addr);
    leds_toggle(LEDS_RED);
    printf("Sending ACk to: %d: \n",addr.u8[0]);
    packetbuf_clear();
    x = 0;
  }
}
```

Listing 6.2: Relay node

Sink node

```


/*-----------------------------------------------------------------------*/
              RECEIVING AND DEAGGREGATION
/*---------------------------------------------------------------------------*/
{
    if (d[0] == '\0')
    {
      clock_init();
      start_time = clock_time();
      printf("Unicast Message Received from %d.%d with id : '%s'",
          from->u8[0],from->u8[1],(char *)packetbuf_dataptr());
      y = 0;
      packetbuf_copyto(d);
      if (d[0]== 'H') {
        printf(" Humidity Packet received from %d.%d with id : '%s' \n",
            from->u8[0],from->u8[1],(char *)packetbuf_dataptr());
        strtok_r (d," ", &ptr);
        printf ("Splitted Packet 1 :'%s'\n",d);
        printf ("Splitted Packet 2 :'%s'\n",ptr);
        printf("Completion time %lu / %u \n", (unsigned long)clock_time() -
    start_time , CLOCK_SECOND);
        packetbuf_clear();
      }
      else if (d[0] == 'T')  {
        (char *)packetbuf_copyto(s);
        strcpy(r, s);
        printf("Saved Temprature Packet is: '%s'\n",
            (char *)r);
      }
      strcpy(d, "");
      strcpy(r, "");
      strcpy(s, "");
      packetbuf_clear();
    }
}
```

Listing 6.3: Sink node

Heterogeneous spatial and temporal aggregation

As the procedure is same for source node and sink node. Therefore, we only providing codes for the relay node.

Relay node

```

{
  if (from->u8[0]== 34) {
    clock_init();
    start_time = clock_time();
    leds_on(LEDS_BLUE);
    leds_off(LEDS_RED);
    i=i+1;   i<=20;
    packetbuf_copyto(f[i]);
    printf("unicast message received from %d:'%s' with length %d\n",
        (char*)packetbuf_dataptr(),packetbuf_datalen());
  }
  else if (from->u8[0]== 32){
    k=k+1; l<=20;
    packetbuf_copyto(g[k]);  }
  x = 0;
  if (  f[1]!='\0' && g[2]!='\0' ) {
    strcpy (m,f[1]);
    strcpy (n,g[2]);
    c=strcat(m,n);
    printf ("contents of f1 :'%s'\n" ,f[1]);
    printf ("contents of g2 :'%s'\n" ,g[2]);
    printf("contents of c:'%s'\n",c);
    strcpy (e,c);
    /*----------------------------------------------------------------------*/
                          FORWARDING PART
    /*----------------------------------------------------------------------*/
    packetbuf_copyfrom((char *)e,40);
    addr.u8[0] = 5;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr))
    {
      unicast_send(&uc, &addr);
      leds_off(LEDS_BLUE);
      leds_on(LEDS_RED);
      double diff=clock_time() - start_time;
      double num_seconds=(double)diff/CLOCK_SECOND;
      printf("Completion time  %lu / %u \n", (unsigned long)clock_time() - start_time
    , CLOCK_SECOND);
      printf("Forwarding Packet to %d:'%s'\n" ,addr.u8[0],(char *)packetbuf_dataptr()
    );
    }
  }
  packetbuf_clear();
  strcpy(m, "") ;
  strcpy(n, "") ;
}
```

Listing 6.4: Relay node

Variable fixed frame-length spatial and temporal aggregation

Relay node

```

{
  if (f[0] == '\0')
  {
    x = 0;
    addr.u8[0] = 26;
    printf("unicast message received from %d: '%s'\n",
        from->u8[0],(char *)packetbuf_dataptr());
    clock_init();
    start_time = clock_time();
    leds_on(LEDS_BLUE);
    leds_off(LEDS_RED);

    (char *)packetbuf_copyto(f);
    packetbuf_clear();
  }
  if (f[0] != '\0')
  {
    addr.u8[0] = 4;
    printf("unicast message received from %d: '%s'\n",
        from->u8[0],(char *)packetbuf_dataptr());
    (char *)packetbuf_copyto(g);
    packetbuf_clear();
    x = 1;
  }
  if (g[0] != '\0'  ){
    c=strcat(f,g);
    printf("Concatenated Packet is: '%s'\n",
        (char *)c);
    packetbuf_clear();

/*------------------------------------------------------------------------*/
                              FORWARDING PART
/*------------------------------------------------------------------------*/
    packetbuf_copyfrom((char *)c,40);
    addr.u8[0] = 5;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr))
    {
      unicast_send(&uc, &addr);
      leds_off(LEDS_BLUE);
      leds_on(LEDS_RED);
      printf("Completion time  %lu / %u \n", (unsigned long)clock_time() - start_time
    , CLOCK_SECOND);
      packetbuf_clear();
      printf("Forwarding Concatenated Packet to %d: '%s'\n" ,addr.u8[0],(char *)c);
    }
    packetbuf_clear();
    strcpy(f, "") ;
    strcpy(g, "") ;
    strcpy(c, "");
  }
}
```

Listing 6.5: Relay node

Variable adaptive frame-length spatial and temporal aggregation

Relay node

```

{

  if (f[0] == '\0')
  {
    x = 0;
    addr.u8[0] = 26;
    printf("unicast message received from %d: '%s'\n",
        from->u8[0],(char *)packetbuf_dataptr());
    clock_init();
    start_time = clock_time();
    leds_on(LEDS_BLUE);
    leds_off(LEDS_RED);
    (char *)packetbuf_copyto(f);
    packetbuf_clear();
  }

  if  (from->u8[0]== 32)

  {
    addr.u8[0] = 4;
    printf("unicast message received from %d: '%s'\n",
        from->u8[0],(char *)packetbuf_dataptr());
    (char *)packetbuf_copyto(g);
    packetbuf_clear();
  }

  if  (from->u8[0]== 26)                    {
    (char *)packetbuf_copyto(h);
    printf("content of c:'%s'\n",h);
    packetbuf_clear();       }
  if (h[0] != '\0')
  {
      etimer_set(&et, CLOCK_SECOND*5);
    snprintf(m, 80, "%s%s%s", f, g, h);
    packetbuf_copyfrom((char *)m,80);
    addr.u8[0] = 5;
    if(!rimeaddr_cmp(&addr, &rimeaddr_node_addr))
    {
      unicast_send(&uc, &addr);
      printf("Forwarding Concatenated Packet to %d: '%s'\n" ,addr.u8[0],(char *)
    packetbuf_dataptr());
    }
    strcpy(f, "") ;
    strcpy(g, "") ;
    strcpy(c, "");
    strcpy(h, "");
    strcpy(m, "");
  }
}
```

Listing 6.6: Relay node