SCHOOL OF COMPUTER SCIENCE
CARLETON UNIVERSITY

Performance Management of Hydroacoustic Surveillance Networks

Wenqian Wang
Supervised by:
Dr. Michel Barbeau
May 4, 2016

1 INTRODUCTION

A Hydroacoustic Surveillance Network (HASN) is a type of Underwater Acoustic Sensor Network (UASN). Such a network consists of devices equipped with acoustic communication capabilities that are deployed underwater at different depths to perform collaborative monitoring tasks [1]. The major goal of this project is to design and implement a performance management system for the HASN. As an example, we use the Location-free Link State Routing (LLSR) [2] protocol. Data packet delivery is upward direction forwarding, using LLSR. The sink node is responsible for collecting the information produced by sensors. For implementing this performance management system, four problems need to be solved: management packet delivery, management packet processing, management packet acknowledgement and management packet authentication. For management packet delivery and management packet authentication, we provide a solution mechanism that downward relay broadcast the management packets. All of the packets contain one authentication byte in this performance management system. For management packet processing, we designed a network management agent with an embedded Management Information Base (MIB). For management packet acknowledgement, we provide a solution that uses LLSR [2] as the packet delivery protocol. In this report, we first present the background (see section 2), then we address the problem (see section 3) and give the detailed solution and our design (see section 4), and conclude with a summary of the project and future work discussion (see section 5).

2 BACKGROUND

In this section, some basic concepts about the network management are briefly introduced before we give a formal introduction of the cornerstone in this project. Then some related works on wireless sensor network management are presented. Finally, a brief introduction to LLSR [2] and GNU Radio [3], on which the project is built upon and developed with, is given.

2.1 INTRODUCTION TO NETWORK MANAGEMENT

As one part of the network design and implementation, network management is the essential key of a functional network. Management refers to the ability to configure, control, operate, and diagnose equipment [4]. In network management, there is a variety of tasks we want to achieve. These tasks could be categorized into five major functional areas. They are defined in the functional model of the Open Systems Interconnection (OSI) management architecture: fault management, configuration management, accounting management, performance management and security management.

In this HASN project, our task is focused on implementing performance management. The goal is the guarantee of quality of service. Related to the design of a management protocol that focuses on performance management, we would like to present some works on management frameworks and protocols in wireless sensor networks.

2.2 RELATED WORK

MANNA AND MNMP

Management Architecture for Wireless Sensor Network (MANNA) is a policy-based management sytem [5]. The system collects dynamic management information, maps into Wireless Sensor Network (WSN) models, and executes management functions and services based on the WSN models [6]. The MANNA management policy defines several management functions that are executed when certain network conditions are met. The WSN models maintain information about the state of the network and store this information in a MIB. The management of the dynamic WSN behaviours is done by analyzing and updating the configurations in the MIB through the MANNA network management protocol (MNMP). This protocol is very light. The sensor nodes in the network are organized in clusters. Each node sends its status to its cluster head. The head of the cluster is responsible for executing local management functions. It aggregates management data received from sensor nodes [6]. Then the head forwards the management data to the base station. Several cluster heads can also work together to achieve global network management. In a hierarchical network architecture deployed with MANNA and MNMP, the goal is to increase the accuracy of the decisions, while the energy efficiency depends on the size of the cluster.

SNMS

The Sensor Network Management System (SNMS) [7] is an interactive tool for monitoring the health of sensor networks. This system has two main management functions: query-based network health data collection and event logging [5]. The user could use the query function to retrieve and oversee the parameters of each node. The event-driven logging function allows the user to set the parameters of the events and nodes. With this system, the nodes in the network report the data according to the values of dynamically configured thresholds. SNMS supports collection and dissemination of traffic patterns [8]. The collection is used to collect heath data from the nodes using a data gathering tree. SNMS has a tree construction protocol. Every node maintains a single best parent node based on the signal strength. The goal is minimising memory usage and traffic. The dissemination of the messages is achieved by applying the Drip protocol. This protocol is application independent. It is able to provide management functions even when the other applications fail. However, the Drip protocol faces a challenge when a component requests several independent variables. There is a trade-off between channel usage and node caching [7]. The SNMS major advantage is the minimal impact on both local node caching and whole network traffic due to the fact that a management query is manually done by a human. While the disadvantage of SNMS is the limit on the proactive management ability.

sNMP

The Sensor Network Management Protocol (sNMP) [6] defines sensor models to represent the current state of the network. It also has various management functions. In sNMP, the sensor models consist of a MIB for storing the status and parameters in the network. The TopDisc protocol used by sNMP has three functions: network state retrieval, data dissemination and aggregation and duty cycle assignment. The main advantage of sNMP is the inter-cluster management information routing. The network administrator can specify the rules for forwarding. The load can be distributed among the nodes in one cluster. It aims at a fair duty cycle assignment [6]. However, cluster-head election and maintenance are expensive in terms of latency and energy [8].

SNMP

Simple Network Management Protocol (SNMP) [9]an internet-standard protocol for managing devices on IP networks. A SNMP agent embedded in each node is responsible for collecting and maintaining data in a MIB. The core part of SNMP is a simple set of operations (and the information these operations gather) that give administrators the ability to change the state of nodes in the network. Also, the SNMP protocol allows for periodic querying variables through an event-driven mechanism. SNMP is widely used in many systems and devices. This project is inspired by SNMP and other sensor network management approaches. In this project the administrator could get and set the parameters of each node by sending in-band management packets within the network. The problems we face with the use of SNMP in sensor networks

are four points: increase of the network overhead, increase of latency, limits of the processing capacity of the manager and limits of the local manager's local segment [8]. Beside these four points, this project focuses on designing a management system for the nodes of a underwater acoustic network which require minimizing load of traffic. Due to the fact that it has been designed for managing IP networks, SNMP introduces several layers encapsulation and long headers. For the sink nodes of the HASN which work on the surface of water, the use of SNMP is still an ideal way for remotely managing the entire network system through conventional IP networks. A recent project which integrates the SNMP functionality into GNU Radio allows existing client side tools on various platforms to access the state of GNU Radio module [10]. With the help of this module, a HASN can be managed. Global network management in remote environments can be achieved.

2.3 Location-free Link State Routing

This project is built into the implementation of the LLSR [2] protocol developed with the GNU Radio toolkit. Unlike location based routing protocols, which rely on an embedded GPS to acquire the geographical position, location-free link state routing is much more suitable for the underwater acoustic sensor networks in which embedded GPS do not work. By applying a greedy hop-by-hop approach, this protocol uses both pressure and beacon signals for ranking and selecting the next one hop neighbour node for forwarding packets. The details of this protocol are illustrated in the problem statement section (Section 3).

2.4 GNU Radio

As an open-source software development toolkit, GNU Radio [3] provides a variety of pre-installed signal processing blocks for implementing software defined radio. It also supports developers to programme unique out of tree blocks for wireless communication and construction of sensor networks. Due to these perks offered by GNU Radio, it is a good tool for our hydroacoustic surveillance network project, which contains three aspects: signal processing, network routing and network management.

3 Problem Statement

When a network management system is designed for wireless sensor networks, there are unique properties that need to be taken into consideration. The project focuses on implementing network management into a HASN with LLSR [2]. We aim to meet design criteria which are used for evaluating sensor network management systems [6]. These five criteria are lightweight operation, robustness and fault tolerance, adaptability and responsiveness, minimal data storage and scalability.

Before presenting of the problems encountered when implementing a performance management system, further explanation about LLSR [2] is presented in the next section.

3.1 Essential LLSR

The core part of LLSR, for HASN is packet forwarding in the direction of the sink [2]. Each node selects a next hop based on a link-state metrics through a beacon mechanism. The sink node always chooses itself as its next hop.

The normal process of establishing routing is demonstrated as follows: Initially, the sink node located on the surface of the water generates a beacon packet that contains its own address, hop count (equal to zero) and path quality. The nodes underwater near the surface first receive this beacon message and update their own hop count, path quality, and table of neighbour nodes. After this, they choose their exclusive next hop node for forwarding the data packet. They produce a new beacon packet based on the one they received and their own status. Then other nodes receive the new beacon packets sent by neighbouring nodes and repeat the process mentioned above. Each node periodically broadcasts a beacon packet.

The data packets in this network are forwarded hop-by-hop until they reach the sink. Each node maintains a table that contains its neighbour nodes, but it only chooses one node as its next hop. If a data packet is being delivered with acknowledgement control sets in its header structure, each node forwards this packet and waits for an acknowledgement packet from its next hop node. This ensures that the data packet is delivered from one node to another. Until a node receives an acknowledgement packet, it remains in the busy status and keeps rejecting other incoming data packets. The source node of the incoming data tries sending the packet several times until it drops the packet.

3.2 Network Management Design

The sink node (with fixed address 0) on the surface of the water is the stable cluster head of the HASN. As the cluster head, it has the burden of executing multiple tasks which are collecting the normal transmitted data, forwarding the management data down to nodes deployed underwater and collecting and processing the management data and operation results. To avoid conflicts between the upward direction packets and downward direction management packets, the ARQ protocol is used. This protocol is also an error-control method for reliable packet delivery. Each node has a SNMP-inspired agent to respond to the management requests sent from the top. In this agent, there is a MIB for mapping the management entities in our

module and their current value. By changing the values mapped to the entities, we can manually monitor and configure a HASN to ensure network performance.

3.3 PROBLEMS

As it is mentioned above, the data packets are being forwarded from deeper nodes to the ones located near the surface, hop-by-hop. Each node does not acknowledge which node chooses itself as its next hop until it receives one data packet marked with itself as a temporary destination on the way to the sink. The final destination of all the data packets is the sink node. The packets are always upward sent by the sensor nodes. There are four problems encountered when implementing a network management system:

- First, the management packets are generated and sent from the sink node and need to reach all the nodes in the network.
- Second, the delivery of management packets reaching the bottom nodes of the network need to be confirmed.
- Third, the management packets need to be properly handled and processed in the destination nodes. Upward management packet delivery needs to be established for sending feedback of management operations and performance monitoring data to the sink node.
- Fourth, authentication validation and integrity checking for management packets need to be implemented.

4 Solution

High-Level Design

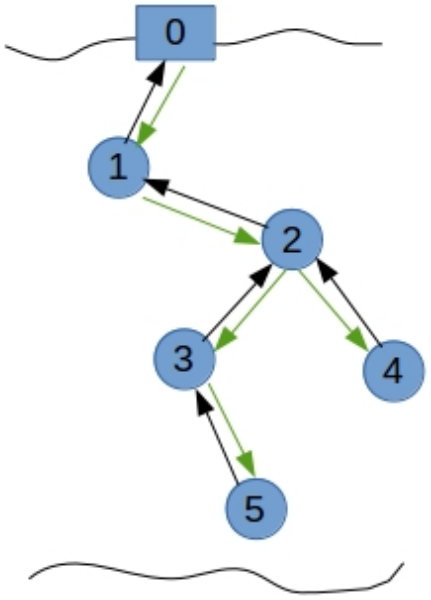

Figure 4.1: Data packet routing.

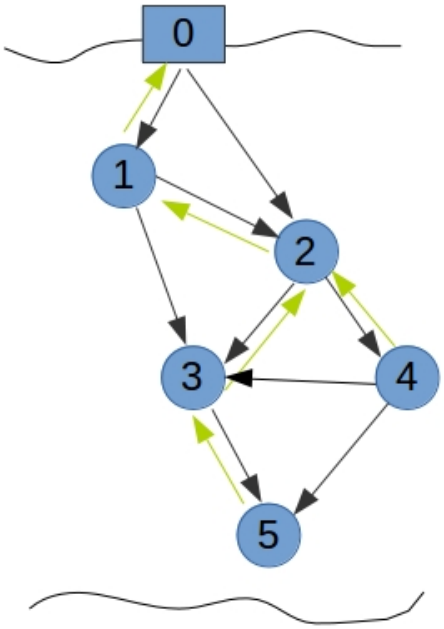

Figure 4.2: Management packet routing.

Figure 4.1 illustrates the flow of data traffic while Figure 4.2 shows the flow of management traffic. Black arrows in Figure 4.1 indicate the data packets transmitted from node-to-node. In Figure 4.2, the black arrows indicate management packets sent out from the node-to-node. While the green arrows in both Figures 4.1 and 4.2 show the transmissions of the acknowledgement packets. Firstly, for data traffic, the data packets collected by each node are forwarded to the sink node (node 0). Each node can only get the acknowledgement packets from their unique next hop node. The ARQ protocol is used for delivering data packets. The design consists of broadcasting and forwarding the management packets from the sink node down to the nodes in the network, like demonstrated in Figure 4.2. Each node repeats once a management packet it receives and rejects the packets if it has received before. As one can see in Figure 4.2, nodes 0 and 1 both deliver one management packet to node 2. Node 2 takes one and rejects one packet. Same for node 3. It rejects all packets except one. This guarantees that a management packet is being forwarded down to the bottom of the network, assuming it does not reach the destination before. While the acknowledgement packet is only sent to each node's next hop destination. This design guarantees the delivery of the packets.

As for the management packet processing, in each node, there is an agent for handling a message retrieved in a management packet. The agent has a MIB (Management Information Base) for mapping managed object identities to parameters of the network. After the node processes the management packet, an acknowledgement mechanism, using LLSR [2], upward forwards management data and operation result to the sink node. For the authentication purposes, the sink node has a table that stores unique secret keys shared with each node in the network. When the sink node receives a management command from the embedded management application, it generates a hash value based on the information contained in the command and destination secret key. After adding the hash value and management protocol header to the management command, the management packet is ready and then pushed into the channel.

DETAILED DESIGN

At first, we would like to introduce our management packet structure, see Figure 4.3.

Figure 4.3: Management packet structure.

Each field is one byte in Figure 4.3, PROTO ID is the id for management protocol. PKT SRC is the source of this management packet as this packet is forwarded hop-by-hop. MGMT TRACK is the tracking number of the management packet sent from the sink node. DEST is the address of the destination node. The default of VALUE is zero. It is used to store the value accompanying a set operation. OPT indicates the management operation 1 (set value) or 0 (get value). OID is the id of the managed object on which the operation is performed. HASH VALUE is an integer for authentication. An example management packet is:

3 1 0 6 4 1 1 21

The management protocol id is 3. This packet is sent from node 1. The packet number is 0. The destination address is 4. The command is changing the value of managed object 1 to 6. The authentication value is 21.

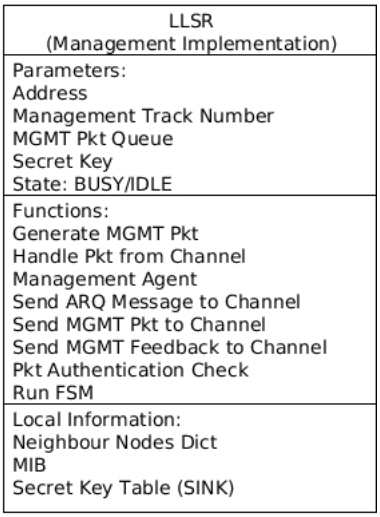

Figure 4.4: The class diagram.

Figure 4.4 is a class diagram that shows the parameters, functions and local information that each node keeps. They are directly related to our management implementation design.

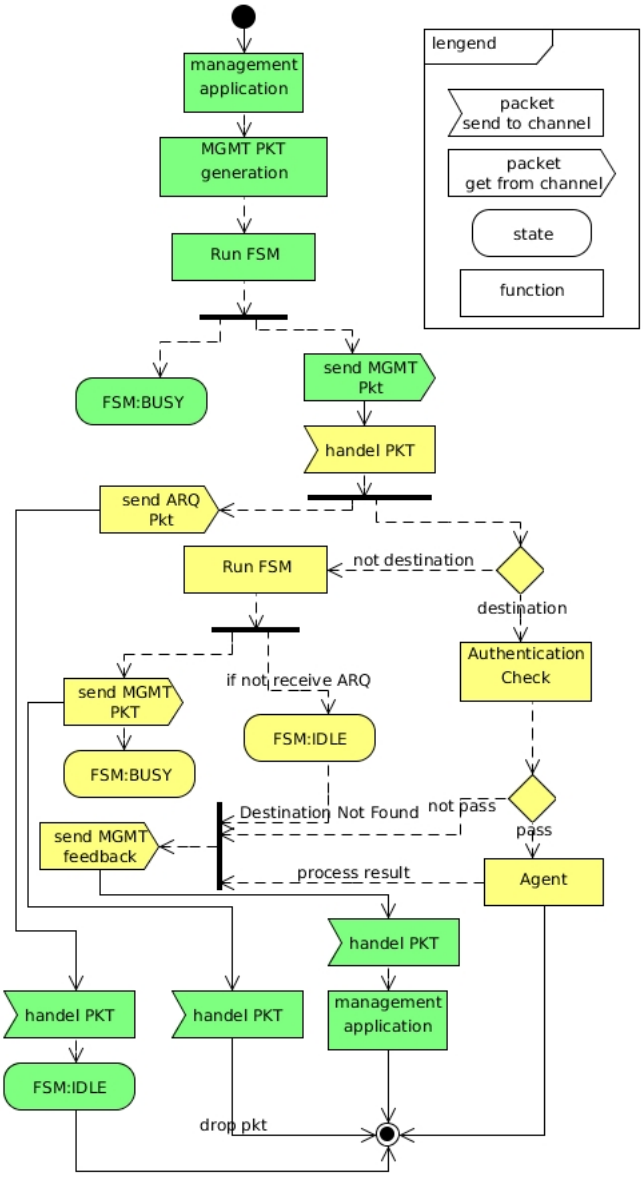

Figure 4.5: The activity diagram.

Figure 4.5 is an activity diagram that shows the activities related to the management protocol between two nodes, node 0 (green) and node 1 (yellow). A rectangle represents a function that has no interaction with a channel port while a wedge represents a function that is directly connected with a channel port in the LLSR class. The dotted lines represent the messages transmitted within the functions in the LLSR class while the solid lines show the packet transmitted in a communication channel except the ones linked to the start and end point of the diagram. Also, labels are added to several lines to specify the messages handled by the functions they link.

A management application sends a Protocol Data Unit (PDU [1]) to the application port of the network module of the sink node. This PDU has four values, as shown in Figure 4.3, which are VALUE, DEST ADD, OPT and OID. Our solution is similar to the send-acknowledge mechanism used for the data routing. Now, we discuss the detailed design of the network management packet delivery.

At the beginning, the sink node receives the management packet assembled as a PDU coming from the application port. To generate the management packet the sink node adds the management protocol id, self-address id, unique auto increasing management packet track number and authentication byte to the data retrieved from this PDU. It pushes this packet into a queue waiting for send out. The sink node does not send the packet out until it receives beacons from other nodes to ensure that the management packet could be received by its downward nodes.

After sending out the packet, the embedded finite state machine transfers from the IDLE state to the BUSY state. Other incoming packets that require handling by this node are rejected to avoid conflicts. When the neighbour nodes of the sender receive this management packet, they send one acknowledgement packet that contains the address of their unique next hop node. By doing this, the node could be transfer from the BUSY state to the IDLE state only when getting the acknowledgement packet from the nodes which select it as their next hop node. This solves the second problem mentioned in section 3.3. Assuming the management packet reaches a bottom node and this is not the destination, then after several sending attempts the node can not get any acknowledgement packet from other nodes. Because of this, it sends a management acknowledgement packet with a failure message using the management acknowledgement protocol back to the top.

Now back to the message handling part of each node. The details of this part are in the next section. If the packet reaches its destination, it gets processed otherwise this packet is resent until it reaches the destination or a bottom node of the network. The first and second problems are both solved in section 3.3.

[1] In GNU Radio, PDU is an envelope for transporting various types of data between different modules

```
# ------------------------------
# Handle a message from the radio
# ------------------------------
def radio_rx(self, msg):
    # message structureis a meta data-data?
    try:
        meta = pmt.car(msg)
        data =  pmt.cdr(msg)
    except:
        if self.debug_stderr:
            #log the error
            sys.stderr.write("in radio_rx(): message is not a PDU\n")
        return
    # data is a vector of unsigned chars?
    if pmt.is_u8vector(data):
        data = pmt.u8vector_elements(data)
    else:
        if self.debug_stderr:
            #log the error
            sys.stderr.write("in radio_rx(): data is not a u8vector\n")
        return
    # convert meta data dictionary from PMT to Python type
    meta_dict = pmt.to_python(meta)
    if not (type(meta_dict) is dict):
        meta_dict = {}
    # Get exclusive access
    with self.lock:
        self._radio_rx(data, meta_dict)
```

Figure 4.6: Packet unpacking function.

In this section, the design of the management data processing is presented for solving part of the third problem introduced in section 3.3. In Figure 4.6, a function unpacks one PDU from the radio (channel) port. As one can see from the code, the PDU get unpacked into data and meta_dict and is passed to the message handle function "_radio_rx". Further data processing is done in this function.

Figure 4.7, shows how a management packet is processed. When the packet protocol id is management protocol, the function checks if this packet comes from a known neighbour node. As the Figure 4.4 shows, in the class there is a dictionary that keeps information about the nearby nodes. Each neighbour is represented as an entity that has multiple properties. One of the properties of each entity in the dictionary is the number of downward management packets received (management track number also the TRACK NUM in Figure 4.3). If the packet is from a unknown node, the function ignores and drops this packet. If the packet is from a known node, the function updates the management track number of the correspond node in its dictionary. A management packet is flooded in the network. Each node receives the management packet with the same track number from all the nodes in its neighbour node dictionary.

```
# ----------------------
# mgmt packet processing
# ----------------------
if data[PKT_PROT_ID]==MGMT_PROTO:
    new_packet=False
    message=[]
    # check this packet from neighbour
    if self.nodes[data[PKT_SRC]]:
            # last mgmt packet number and new mgmt packet number different?
            new_packet=self.nodes[data[PKT_SRC]].lmn!=data[MGMT_TRACK]
            self.nodes[data[PKT_SRC]].setLmn(data[MGMT_TRACK])
    # check if the packet is a old packet
    if self.mgmt_track-1==data[MGMT_TRACK]:
       if self.debug_stderr:
             sys.stderr.write("%d: Receive former mgmt packet, drop" % self.addr)
       return
    else:
        self.send_ack(self.next_hop, data[MGMT_TRACK], data[PKT_PROT_ID])
    if new_packet:
        # this node is the destination
        if self.addr==data[MGMT_DEST]:
            # yes! processing
            # check hash
            if self.checkhash(data[3:MGMT_PKT_LENGTH-1]+[self.localkey], data[MGMT_HASH])==False:
                if self.debug_stderr:
                   sys.stderr.write("%d: MGMT TRACK: %d, Hash Wrong and Original Hash: %d,Hash Get: %d \n" % \
                   (self.addr,data[MGMT_TRACK],data[MGMT_HASH],self.addhash(data[3:MGMT_PKT_LENGTH-1]+[self.localkey])))
                return
            message=self.agent(data[MGMT_OPT],data[MGMT_OID],data[MGMT_VAL])
            sys.stderr.write("mgmt message: %d\n" % message)
            self._mgmt_ack_rx(self.mgmt_ack_pdu(message[0], data[MGMT_TRACk], message[1]))
        # else, if the packet is new, then broadcast it out
        elif self.mgmt_track==data[MGMT_TRACK]:
             self._mgmt_rx(self.pdupacker(data[MGMT_MIN:MGMT_PKT_LENGTH-1]))
    return
```

Figure 4.7: Management packet processing script.

After checking the packet source, the function then verifies if this packet is a former one. If the incoming packet is not new then it is a packet from the neighbour nodes, the packet is dropped. While if the packet is a new packet, the node sends an acknowledgement packet to its next hop towards the direction of the sink node. After that the destination of this packet is checked, if it is not equal to the local address, the node re-sends this management packet. While if it is equal to the node local address, this management packet is sent from the sink node to this node. The function calls the agent function with three parameters: OPT, OID and VALUE to do the final processing. The code of the agent function is in Figure 4.8

```
# ---------------------------------------
# Network management Agent
# ---------------------------------------
def agent(self, opt, oid, value):
    self.message=[]
    # simple management information base
    self.mib={1:self.max_attempts}
    # check oid valid
    if oid in self.mib:
       if opt == 0:
          self.message=[0,self.mib[oid]]
       elif opt == 1:
          self.mib[oid]=value
          self.message=[1,200]
    else:
    # wrong id
       self.message=[1,204]
    # return result
    return self.message
```

Figure 4.8: Management agent function.

As one can see, the agent is very simple, it contains a dictionary called "mib". In this sample function, this dictionary currently contains one entity for testing. If the OID is matched, the management operation is retrieved (OPT=0) or changed (OPT=1) according to OPT and VALUE. The result is returned to the function for generating a management acknowledgement packet. In Figure 4.7, the management processing result is handled by another set of functions which detailed design is presented in the next section.

Management Acknowledgement

In this section, the management acknowledgement procedure is presented for solving the third problem in section 3.3. To perform performance management, the function that handles the management information and management operation result forwarding to the sink node is a crucial part of this project. Figure 4.9 shows the management acknowledgement packet structure.

Figure 4.9: Management acknowledgement packet structure.

The forwarding of a management acknowledgement packet is the same as for a data packet. PKT SRC and PKT DEST are the temporary source and destination. PKT CNT shares the counter value of the acknowledged data packet. This is used by the ARQ protocol when the packet is forwarded to the sink node. We put our focus on explaining the fields MGMT FLAG, MGMT SRC, MGMT TRACK and MGMT VALUE. The HASH VALUE, as one can see, is grey due to the fact that it is generated using the same secret key (sink node secret key) over the fields MGMT FLAG, MGMT SRC, MGMT TRACK and MGMT VALUE. MGMT FLAG is an identifier used by the sink to distinguish the various types of the management acknowledgement packets. The default MGMT FLAG value is 1, marking the packet a response to the management operation

GET (retrieving the value associated to an OID). The MGMT FLAG value 2 indicates that the packet has a network error message or successfully returns information from the source node. The value 0 is for automatically generated management data packets sent to the sink node for monitoring network performance. The MGMT SRC is the indicator of the packet original sender (source node). The MGMT TRACK shows that this packet is the response of a management packet with the same track number. The MGMT VALUE stores a value or status code.

When the sink node gets one management acknowledgement packet, it checks the type of the response, and generates a response PDU to the management application which is responsible for running high-level management tasks. The last part of the third problem in section 3.3 is solved.

AUTHENTICATION

This section explains the solution of the fourth problem in our design. Since the delivery of management packet is a procedure of broadcasting, a simple and light set of functions that provide each node the ability for checking the authenticity and integrity of the management packets are implemented. To achieve that, a table is added to the sink node which stores the unique secret keys shared with all the nodes in the network. Every time a management packet is sent, one additional byte is added at the end of it. This byte is generated by combining the destination key and 4 bytes of information (VALUE, DEST ADD, OPT and OID in Figure 4.3) sent from the management application with the SHA256 function in the Python Hash Library. When the packet reaches the destination, the DEST node uses its unique key combining with the 4 bytes of information it received to generate a one-byte hash value to compare with the one it received. By applying this procedure, the packet authenticity and integrity are confirmed. The function for generating this hash value is implemented in all the nodes and called when the nodes are generating the management packets and management acknowledgement packets.

5 Conclusion and Future work

This work focuses on the implement a network management system into a HASN project that uses LLSR [2] for link establishment and data packet upward delivery. The design of this management system has achieved four goals: management packet downward delivery, bottom node detection, feed back of the management operations and monitoring data and the authenticity and integrity checking for the packets involved in the network management. A downward broadcast mechanism is designed for management packet delivery. By using the next hop of each node in the network for sending the acknowledgement packet, the detection of the bottom node is achieved since there are no more nodes that select itself as their next hop. The upward management acknowledgement packets are sent, as the data packets, using LLSR [2]. The authentication of the management packets is done by adding an extra hash value byte that is generated with a unique secret key associated with each node in the network. The Appendix of this report gives a simple simulation example of this project. In the appendix, a flow graph for simulating this project is provided and debug messages of the simulation are also presented. In the end, a picture of the device which used is planned for field testing is given.

The behaviour of this network needs to be improved such that the management commands can be generated by the management application in the sink node based on the performance monitoring data collected from network nodes. We also need to connect our system, which is implemented in the head of the network cluster, with the SNMP GNU Radio module [10] for providing accessibility from a SNMP client that operates on a platform in a remote environment. For the final step, the whole system is going to be implemented using small devices and tested in the field. An evaluation of the design will be conducted based on the criteria mentioned in section 3.

REFERENCES

[1] Dario Pompili, Tommaso Melodia, and Ian F Akyildiz. Distributed routing algorithms for underwater acoustic sensor networks. *Wireless Communications, IEEE Transactions on*, 9(9):2934–2944, 2010.

[2] Michel Barbeau, Stephane Blouin, Gimer Cervera, Joaquin Garcia-Alfaro, and Evangelos Kranakis. Location-free link state routing for underwater acoustic sensor networks. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 1544–1549. IEEE, 2015.

[3] Gnu radio. The GNU Radio Foundation, 2016. `http://gnuradio.org/`.

[4] S. Abeck. *Network management.* Amsterdam: Morgan Kaufmann/Elsevier, 2009.

[5] Linnyer Beatrys Ruiz, José Marcos Nogueira, and Antonio AF Loureiro. Manna: A management architecture for wireless sensor networks. *Communications Magazine, IEEE*, 41(2):116–125, 2003.

[6] Winnie Louis Lee, Amitava Datta, and Rachel Cardell-Oliver. Network management in wireless sensor networks. *Handbook of Mobile Ad Hoc and Pervasive Communications: American Scientific Publishers*, 2006.

[7] G. Tolle and D. Culler. Design of an application-cooperative management system for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pages 121–132, Jan 2005.

[8] Bin Zhang and Guohui Li. Survey of network management protocols in wireless sensor network. In *E-Business and Information System Security, 2009. EBISS'09. International Conference on*, pages 1–5. IEEE, 2009.

[9] D. Harrington, R. Presuhn, and B. Wijnen. An architecture for describing simple network management protocol (SNMP) management frameworks. United States, 2002. RFC Editor.

[10] Zach Renaud. Network management for software defined radio applications. Honours project report, School of Computer Science, Carleton University, 2016.

APPENDIX

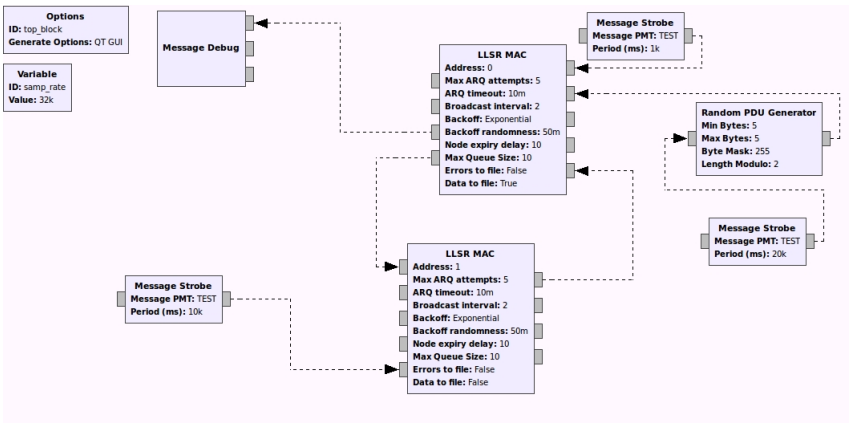

Figure 5.1: GNU Radio flow graph.

Figure 5.1 is a GNU Radio flow graph that is built up for simulating the project. As one can see, the LLSR MAC is the module which our project is implemented into. We use the block Random PDU Generator to produce a packet for simulating one command sent from the management application. The Message Strobe is a block for test purposes which controls the time period for calling the connected block. It is obvious that there are only two LLSR MAC blocks in this flow graph which represents two nodes in our network. A message Debug block is linked to the LLSR MAC block for printing the message received from the output port.

```
9:in send_beacon_pkt(): sending beacon packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): receiving packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): node 0 is alive
in SelectNextHop(): addr: 1, hc: 1, pq: 1, next hop: 0
1:in send_beacon_pkt(): sending beacon packet:
PROT ID: 2 SRC: 1 HC: 1 PQ: 1
9:in run_fsm(): sending mgmt packet, pkt track No: 0
9: send_mgmt_pkt_radio(): sending mgmt packet:
PROT ID: 3 PKT FROM: 0 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
pkttype:1
1:in _radio_rx(): receiving packet:
9:in _radio_rx(): receiving packet:
PROT ID: 2 SRC: 1 HC: 1 PQ: 1
9:in _radio_rx(): node 1 is alive
in SelectNextHop(): addr: 0, hc: 0, pq: 255, next hop: 0
PROT ID: 3 PKT FROM: 0 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
1:in send_ack(): sending ack packet for protocol 3:
PROT ID: 0 SRC: 1 DEST: 0 CNT: 0 ACK for PROT: 3
1:in run_fsm(): sending mgmt packet, pkt track No: 0
1: send_mgmt_pkt_radio(): sending mgmt packet:
PROT ID: 3 PKT FROM: 1 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
pkttype:1
9:in _radio_rx(): receiving packet:
PROT ID: 0 SRC: 1 DEST: 0 CNT: 0 ACK for PROT: 3
9:in_radio_rx(): got mgmt ack 0 and recover to IDLE
9:in _radio_rx(): receiving packet:
PROT ID: 3 PKT FROM: 1 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
9: Receive former mgmt packet, drop0:in send_beacon_pkt(): sending beacon packet
:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
```

Figure 5.2: Trace of messages 1.

As one can see from Figure 5.2, at first, the two nodes establish communication by sending and receiving beacon packets. Then, the management application port of node 0 (sink node) receives one PDU from the Random PDU Generator which has 4 bytes. After that, node 0

generates one management packet with an authentication byte and pushes this packet to the channel port. When node 1 receives this packet, it checks the destination, and sends this packet back to the channel. Node 0 receives this packet and finds out it is not a new packet and drops it.

```
CHANNEL-BUSY: 1 in run_fsm(): retransmission after 1 retries
current retransmission pkttype: 1
1: send_mgmt_pkt_radio(): sending mgmt packet:
PROT ID: 3 PKT FROM: 1 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
0:in _radio_rx(): receiving packet:
PROT ID: 2 SRC: 1 HC: 1 PQ: 1
0:in _radio_rx(): node 1 is alive
in SelectNextHop(): addr: 0, hc: 0, pq: 255, next hop: 0
0:in _radio_rx(): receiving packet:
PROT ID: 3 PKT FROM: 1 TRACK :0 VALUE :32 DEST: 248 OPT: 233 OID: 56 HASH: 18
0: Receive former mgmt packet, drop0:in send_beacon_pkt(): sending beacon packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): receiving packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): node 0 is alive
in SelectNextHop(): addr: 1, hc: 1, pq: 1, next hop: 0
0:in send_beacon_pkt(): sending beacon packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): receiving packet:
PROT ID: 2 SRC: 0 HC: 0 PQ: 255
1:in _radio_rx(): node 0 is alive
in SelectNextHop(): addr: 1, hc: 1, pq: 1, next hop: 0
```

Figure 5.3: Trace of messages 2.

In Figure 5.3, node 1 is waiting for the ARQ packet supposed to be sent from other nodes that selects it as their next hop. The node 1 does not get the ARQ packet, and start retransmitting this packet. Node 0 keeps dropping this old packet sent from the node 1.

```
1:in send_mgmt_ack_radio(): sending packet:
PROT ID: 4 MGMT ACK PKT FROM: 1 MGMT ACK PKT TO :0 MGMT ACK PKT CNT :0 MGMT ACK
FLAG: 1 MGMT ACK SRC: 1 MGMT ACK TRACK: 0 MGMT ACK VALUE: 2 MGMT ACK HASH: 250
pkttype:2
1: No arq pkt receive, this node is the bottom and mgmt_pkt not reach the dest
1: sending error message to sink node
PROT ID: 2 SRC: 1 HC: 1 PQ: 1
0:in _radio_rx(): node 1 is alive
in SelectNextHop(): addr: 0, hc: 0, pq: 255, next hop: 0
0:in _radio_rx(): receiving packet:
PROT ID: 4 MGMT ACK PKT FROM: 1 MGMT ACK PKT TO :0 MGMT ACK PKT CNT :0 MGMT ACK
FLAG: 1 MGMT ACK SRC: 1 MGMT ACK TRACK: 0 MGMT ACK VALUE: 2 MGMT ACK HASH: 250
```

Figure 5.4: Trace message 3

Finally node 1 reaches its predefined max attempt threshold. It automatically sends an error message to the sink node. (Management Acknowledgement ID, protocol id is 4 and error message code, VALUE, is 2)

Figure 5.5: Device on which our project will be implemented.

In Figure 5.5, it shows one device on which our project is going to be implemented in and tested with.