ROBUST WIRELESS REPROGRAMMING METHOD USING DIFFERENTIAL APPROACH FOR WIRELESS SENSOR NETWORKS

Ms. Ujwala Bhanarkar, Dr. M U Kharat
G.H.Raisoni COEM, Pune.

ABSTRACT

The concept of software reprogramming in wireless sensor networks allows end users to improve the overall functionality of sensor network. The main aim of software reprogramming in WSN is to improve the energy utilization performance with low cost. In this paper initially we are discussing the overall research categories of software reprogramming and then discussing the different protocols presented for software. In this paper two problems of software reprogramming in WSN are investigated such as energy efficiency improvement as well as improvement in reprogramming delay. The new method introduced here with aim of improving the energy efficiency and delay of reprogramming. This new method is based on differential reprogramming mechanism which that mitigates the effects of program layout modifications and retains maximum similarity between old and new software using a clone detection mechanism. Moreover, this proposed method organizes the global variables in a novel way that eliminates the effect of variable shifting. We studied the previous method called Stream which we implemented here and compared this with our proposed approach. In this paper we have presented our initial simulation results which are carried using the NS2 tool.


I. INTRODUCTION

Wireless sensor networks (WSNs) have been studied for a wide range of applications, such as ecological surveillance, habitat monitoring, infrastructure protection, etc. WSN applications often need to be changed after deployment for a variety of reasons—upgrading node software, correcting software bugs, and patching security holes. Many large-scale WSNs, however, are deployed in environments where physically collecting previously deployed nodes is either very difficult or
Enabling sensor nodes to be reprogrammable over the air is a crucial technique to address such challenges [1]. There are several key factors that affect the reprogramming efficiency and reprogramming lifetime of a WSN, including the transferred code size, the loading cost, and the reprogramming voltage requirement.

An important feature of wireless sensor network is reprogramming, i.e. the capability to change software functionality of nodes within the network at run time. Changes come in the form of updates, consisting of new applications, bug fixes or modified parameters. Reprogramming is important both during development, for fast prototyping and debugging, and after deployment, for adapting functionality. We can categorize reprogramming according to the type of change that is required in the network and on the nodes themselves. In general, we call these modifications updates. We can distinguish:

- an update of the operating system;
- an update of an application;
- an addition of a new application;
- a modification of parameters in an existing application.

Wireless sensor nodes are usually reprogrammed in two ways: either by flashing the node with a complete firmware image, or by loading a partial executable binary. The second approach is more flexible and allows easier extension of applications, without the need to reboot the operating system. Despite its flexibility, it has limited support on existing sensor network platforms. However, the Contiki operating system [6], which is used in this report, is specifically designed for wireless sensor networks and has dynamic linking as core functionality.

In literature [2] [3] [4] [5] [6] [7] [8] [9], we have studied many protocols those are presented for energy efficiency as well as improvement in reprogramming delay in wireless sensor networks. However each method is having its own limitations. In this paper we are presenting the novel method which is based on distributed network reprogramming approach. In the below section we are first presenting the review of different methods presented for software reprogramming. In section next we will present the proposed method and its model. In below section II presenting the literature survey over the existing methods. Later in section III present the proposed algorithms, its block diagram, and mathematical representing is presented. Further in section IV, the practical analysis described in which the current state of results presented. The different datasets those are used for practical study are also listed there. Finally in section V conclusion is made based on all discussions.

II. LITERATURE SURVEY

Basically research in reprogramming of WSN are basically studied under three different categories such as reprogramming protocols, reducing the size of transmitted updates, and execution environment of sensor node. In this paper our main working is over the WSN reprogramming protocols. These protocols are used for disseminating the updates. Below figure 1 is showing the overall architecture for WSN reprogramming research categories. The main aim of the different researches done in these categories is to improve the performances of energy efficiency, network performance, network security etc.
During this section we will present the different protocols those are presented for WSN reprogramming. The protocols those are build for disseminating the software reprogramming in WSN. The main difference between the data collection protocols and reprogramming protocols in WSN are:

- Software updating data flow is from the gateway to the nodes,
- Transport must be reliable.

Following figure 2 is showing the basic architecture for disseminating protocols in WSN.

Figure 2: Dissemination Protocols

MOAP [2]: MOAP is a multi-hop, over-the-air code distribution mechanism specifically targeted at MICA2 motes running TinyOS. It uses store-and-forward, providing a ‘ripple’ pattern of updates; lost segments are identified by the receiver using a sliding window, and are re-requested using a unicast message to prevent duplication; a keep alive timer is used to recover from unanswered unicast retransmission requests – when it expires a broadcast request is sent. The base station broadcasts publish messages advertising the version number of the new code. Receiving nodes check this against their own version number, and can request the update with subscribe messages. A link-statistics mechanism is used to try to avoid unreliable links. After waiting a period to receive all subscriptions, the sender then starts the data transfer. Missing segments are requested directly from the sender, which prioritizes these over further data transmissions. Once a node has received an entire image, it becomes a sender in turn. If a sender receives no subscribe messages, it transfers the new image to program memory from EPROM, and reboots with the new code. Sliding window
acknowledgements reduce power consumption (reduced EEPROM reads) at the cost of reduced out-of-order message tolerance. There is no support for rate control, or suppressing multiple senders (apart from link statistics).

*MNP [3]:* MNP is targeted at MICA2 motes running TinyOS and uses the XNP boot loader along with a dedicated network protocol to provide multi-hop, in-network programming. The MNP protocol operates in 4 phases:

1. Advertisement/Request, where sources advertise the new version of the code, and all interested nodes make requests. Nodes listen to both advertisements and requests, and decide whether to start forwarding code or not (this acts as a suppression scheme to avoid network overload);
2. Forward/Download, where a source broadcasts a *Start Download* message to prepare the receivers, and then sends the program code a packet at a time (in packet-sized *segments*) to the receivers to be stored in external memory (EEPROM) – there is no ack, the receiver keeps a linked-list of missing segments in EEPROM to save RAM space;
3. Query/Update, where the source broadcasts a *Query* to all its receivers, which respond by unicast by asking for the missing packets (segments) – these are then rebroadcast by the source node, and then another *Query* is broadcast until there are no requests for missing packets. The receivers, having received the full image, now become source nodes and start advertising the new program;
4. Reboot, entered when a source received no *requests* in response to an *advertisement*, where the new program image is transferred to program memory, and the node reboots with the new code.

A node sends a *download request* to all senders, this assists in sender selection, and also allows the hidden terminal effect to be reduced (as other potential senders can overhead this request). The sender selection algorithm attempts to allow only one active sender in a particular neighborhood. Flow control is rate based, determined by the EEPROM write speed (of the MICA2 mote).

*Trickle [4]:* Trickle runs under TinyOS/Mate – it acts as a service to continuously propagate code updates throughout the network. Periodically (*gossiping interval* $\tau$) using the *maintenance algorithm* every node broadcasts a code summary (‘metadata’) if it has not overheard a certain number of neighbours transmit the same information. If a recipient detects the need for an update (either in the sender or in the receiver) then it brings everyone nearby up to date by broadcasting the needed code. Trickle dynamically regulates the per-node, Trickle-related traffic to a particular rate (rx+tx), thus adjusting automatically to the local network density. This scales well, even with packet loss taken into account. A listen-only period is used to minimize the short-listen problem (where desynchronized nodes may cause redundant transmissions due to a shift in their timer phases). The CSMA hidden-terminal problem does not lead to excessive misbehavior by Trickle, as long as the traffic rate is kept low. By dynamically changing the gossip interval, Trickle can propagate changes rapidly, while using less network bandwidth when there are no known changes. Programs fit into a single TinyOS packet.

*Deluge [5]:* Deluge is a data dissemination protocol and algorithm for propagating large amounts of data throughout a WSN using incremental upgrades for enhanced performance. It is particularly aimed at disseminating software image updates, identified by incremental version numbers, for network reprogramming. There is no support for heterogeneity: the same image is disseminated to all nodes in the network. The program image is split into fixed size pages that can be ‘reasonably’ buffered in RAM, and each page is split into fixed size packets so that a packet can be sent without fragmentation by the TinyOS network stack. A bit vector of pages received can be sent in a single TinyOS network packet. Nodes broadcast advertisements containing a version number and a bit
vector of the associated pages received, using a variable period based on updating activity. If a node determines that it needs to upgrade part of its image to match a newer version, then, after listening to further advertisements for a time, it sends a request to the selected neighbour for the lowest page number required, and the packets required within that page. After listening for further requests, the sender selects a page, and broadcasts every requested packet in that page. When a node receives the last packet required to complete a page, it broadcasts an advertisement before requesting further pages – this enhances parallelization ("spatial multiplexing") of the update within the network (as the node can now issue further requests in parallel with responding to requests from other nodes). The protocol keeps the state data to a fixed size, independent of the number of neighbours.

There are no ACK's or NACK's – requesters either request new pages, or re-request missing packets from a previous page. There is no global co-ordination to select senders; heuristics are used to try and elect relatively remote senders in order to minimize radio network contention. Incremental updating is supported through the use of Complete Advertisements which indicate which pages in an image have changed since the previous version; requesters can then request just the changed pages. Future versions of Deluge are expected to address the following issues: control message suppression, running updates concurrently with applications, explicitly reducing energy consumption, and support for multiple types and versions of images.

**Deployment Support Network (DSN)** [6]: An alternative to either accessing the nodes individually, or accessing them over the sensor network, is to provide a parallel, maintenance network – a Deployment Support Network (DSN). Accessing the nodes individually for a software update is normally impractical for two reasons: the large number of nodes, and the inaccessibility of the nodes. Accessing them over the wireless sensor network itself has several disadvantages: it relies on the network being operational, it has an impact on the performance of the sensor network, and it depletes the nodes' energy. A DSN of small, mobile, temporarily attachable nodes can provide a solution to some of these problems. Providing virtual connections from a host PC to the individual nodes allows normal host tools to be used in updating the software.

**Impala/ZebraNet** [7]: Impala is the middleware layer of the ZebraNet wireless sensor network, which uses wildlife tracking as a target application in the development of a mobile wireless sensor network. Impala provides an event-based middleware layer, which is specially designed to allow applications to be updated and adapted dynamically through the use of application adapters. Events are processed by the Event Filter and then dispatched through these Application Adapters to the appropriate application. Zebrenet nodes are intended to be situated in large numbers in places inaccessible to system administrators, and to support this, Zebrenet supports high node mobility, constrained network bandwidth, and a wide range of updates (from bug fixes, through updates, to adding and deleting entire applications). Applications consist of multiple, shareable modules, organized in 2K blocks. The Application Updater allows applications to continue running during updates, and can process multiple contemporaneous updates; version numbering is used to ensure compatibility of updates with existing modules. It also handles incomplete updates, and provides a set of simple sanity checks before linking in a new module. Software updates are performed in a 3-step process: firstly the nodes exchange in index of application modules, then they make unicast requests (using node ID as a tie-breaker) for updated modules, and finally they respond to requests from other nodes, by transmitting the first packet for the first requested modules. The backoff timer increases exponentially if all neighbours have the same software versions – this significantly reduces management traffic, but can delay updates when two originally separated groups of nodes (at different version numbers) become connected. If memory space is exhausted, then older incomplete version are deleted. When software reception is complete, then after performing simple sanity checks, the old version application is terminated, the modules in the new version are linked in, and the new application is initialized prior to use.
Infuse [8]: Infuse is a TDMA-based protocol for disseminating bulk data in location-aware sensor networks. Nodes periodically select predecessors and successors; energy is reduced by a selective listening policy, turning off the receiver for other TDMA slots; the selective use of predecessors and successors also prevents broadcast storms. A base station broadcasts a start download message containing a version ID and specifying the number of capsules in the new data sequence. It then sends the capsules in subsequent TDMA slots. Receivers forward the start download message; when they receive data modules they save them to flash and forward them. When the last capsule is received, it signals the application that the download is complete. Two different loss recovery mechanisms are discussed in their paper: Go-Back-N, based on implicit acknowledgements; and Selective-Retransmission, based on explicit, piggybacked acknowledgements. By selecting preferred predecessors, energy use can be reduced further. The authors claim that Infuse has more effective pipelining than Deluge, as all the sensors receive the data capsules at approximately the same time, avoiding the CSMA contention at the middle of the network seen in Deluge.

XNP [9]: XNP provides a single-hop, in-network programming facility for TinyOS. A special Boot Loader must be resident in a reserved section of program memory, and the xnp protocol module must be wired into an application (to allow for subsequent XNP updates). A host PC application xnp loads the image, via a base station mote running TOSBase (this acts as a serialto- wireless bridge) to one (mote-id specific) or many (group-id specific) nodes within direct radio range of the base. The image is sent in capsules, one per packet; there is a fixed time delay between packet transmissions. In unicast mode, xnp checks delivery for each capsule; in broadcast mode, missing packets are handled, after the full image download has completed, using a follow-up query request (nodes respond with a list of missing capsules). The program is loaded into external (nonprogram) memory. Applications are halted during the program download. When a reboot command is issued (via the xnp host program), then the boot loader is called: this copies the program from external to program memory, and then jumps to the start of the new program.

III. PROPOSED APPROACH FRAMEWORK AND DESIGN

3.1 Proposed System Architecture
This paper is focusing to present the differential-based reprogramming scheme, which utilizes clone detection to determine code changes efficiently. Furthermore, this method handles branches, global variables, indirect addresses and relative branches by amending the ELF format in a manner which is compatible with standard ELF. These effective amendments dramatically reduce the internal ash memory usage and require near zero external ash memory. Furthermore, the wireless sensor motes require no restart after applying this method generated patch. RISC architecture does not allow to direct access to the memory location. Memory locations are accessed or modified using registers. Indirect addressing provides fast access to large data structures, e.g. arrays, linked lists, union, etc. However, any changes in the global variable layout will also change the corresponding indirect instructions and must be taken care of in order to produce correct patch files.

Design Goals:
The main objectives of the differential-based reprogramming scheme are as follows:

- Minimizing difference size, as small difference means less energy and time to reprogram the network.
- Minimizing ash memory writing, as ash memory writing is energy hungry and reduction of ash memory writing means better energy and time efficiency.
- Minimizing external ash requirement, as reducing external ash usage means reducing cost, as well as reducing requirement for board real estate.
Supporting heterogeneity as many different hardware platforms and operating systems are available for wireless sensor networks, with diverse specifications.

Eliminating the need for mote reboot after applying the patch on mote software.

**Differential-based Reprogramming Scheme Using Clone Detection:** A differential-based reprogramming scheme consists of several major steps. These are, receive old and new files from code database, calculate the patch using an algorithm, compress and encode patch, store (old, new, patch) tuple in the database (for future usages), transfer the encoded patch over the network, sensor mote receives patch and stores it in memory, bootloader reads, parses, and applies that patch on existing firmware and finally, re-start the application.

Figure 4.1 illustrates the steps of differential reprogramming.

![Figure 3: major steps of differential-based reprogramming scheme](image)

To calculate the patch file, this method considers both executable files, i.e. the ELF file, as well as the high-level source code. Making use of high-level programming languages provides significant advantage over other schemes, which only use low-level, because it’s grammatical and syntax rules provide better flexibility to determine code similarities. The code similarities are determined using clone detection tools. The reprogramming steps of this method are shown in Fig. 3. We will explain each and every step of this process in our future works with practical analysis of the same. The methods of reprogramming steps are presented in detail in next figure 4:

1. Proposed Method calculates clones between the old and the new file using clone detection tool. The clone detection tool takes in the C file of the old and the modified program and calculates a mapping between functions and variables.
We assume that the clone detection mechanism has perfect precision. The C files, in case of TinyOS programming, are generated by the compiler as it converts the nesC into C before it creates the final ELF file. The mapping is a list of function and variable clones between the old and the new file. The clone detection algorithm can detect clones under various circumstances, e.g. change in file layout, rename of functions and variables, addition or removal of one or more lines of code, etc. For this thesis, we emphasize ‘Type 3’ clone detection as it can detect addition and removal of new code, functions and variables.

2. Proposed method disassembles the ELF files (both old and modified), using the core dump utilities, to determine different sections, i.e. code, data and bss. In this step, mappings between functions and global variables, computed from the clone detection step are used for the function and the variable reordering. This step is further subdivided as follows:

(a) Reorder functions and global variables to enhance the similarity between old and new programs. Place new functions, initialized and uninitialized global variables at the end of the code, data and bss sections respectively.

(b) Change all references to the re-ordered functions and variables.
3.2 Algorithm and Mathematical Formulation

3.2.1 Problem Definition

Input Set

Consider a set $S$ of $N$ sensor nodes given to monitor events in a data-intensive WSN application. Assume that they are deployed by some generic deployment strategy feasible locations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_h$</td>
<td>High frequency content of input signals</td>
</tr>
<tr>
<td>$R_l$</td>
<td>Sampling at a low rate</td>
</tr>
<tr>
<td>$R_h$</td>
<td>Sampling at a high rate</td>
</tr>
<tr>
<td>$R_m$</td>
<td>Minimum required sampling rate</td>
</tr>
<tr>
<td>$R_c$</td>
<td>Current sampling rate (i.e., continue active at this rate)</td>
</tr>
<tr>
<td>$d$</td>
<td>Time interval index, $d=1,2,\ldots$</td>
</tr>
<tr>
<td>$D_h$</td>
<td>The duration of each burst of high rate sampling</td>
</tr>
<tr>
<td>$D_l$</td>
<td>The duration (short interval) of low rate or adjust sampling</td>
</tr>
<tr>
<td>$I$</td>
<td>Give a whole monitoring round or time interval</td>
</tr>
</tbody>
</table>

NP-Complete Problem

$L=[l_1;l_2;\ldots;l_N]$, where sensor $i$ is placed at location $li$, and $l_0$ is a suitable location of the sink. Let $X$ be the communication range, where the maximum and minimum communication ranges of a sensor are $X_{max}$ and $X_{min}$, respectively. $X_{min}$ is used to maintain local topology, where every pair of sensors within $X_{min}$ are allowed to share and compare their decision with their neighbors. Sensor $i$ corresponds to a node, and any two nodes are connected if their corresponding nodes in $X_{min}$ can communicate directly.

The intention of adopting adjustable $X$ is to reduce energy costs for frequent long distance transmission. Advanced sensor platforms, such as Imote2, support discrete power levels.

NP-Hard Problem

A. Energy Cost Model ($E_i$): One objective is to minimize network energy cost, hence to maximize the network lifetime. We achieve it by reducing the total energy cost, denoted by $E_i$, on a sensor $i$ in different aspects, including sampling, analog to digital conversion (ADC), computation, and communication. At first, we briefly describe how energy is consumed by a sensor communication component in packet transmission/reception. The maximum energy cost of a sensor depends on a routing protocol used by the data collection application. This falls into the domain of power aware routing.

Consider a routing algorithm: we define $q[i]$ as the $i$th hop sensor on the path $q$, and $q$ as the amount of traffic flowing along path $q$ within each round of monitoring data collection. Then $q[i]q[i+1]$ is the distance between any two sensors $q[i]$ and $q[i+1]$. $q[i]q[i+1] \leq R_{min}$ is used for data delivery to the neighbors and $q[i]q[i+1] \leq R_{max}$ is used for data delivery to the sink. Let $e_s$ and $e_r$ be the energy cost for receiving and transmitting data, respectively. Thus, $E_i$ is decomposed into the following parts:

$$E_i = e_t + e_{comp} + e_{ADC}$$

(1)
(i) $e_t$ is the total energy cost per bit for transmission over a link between a transmitter and a receiver. Hence,

$$e_t = \sum_{\forall q, \exists i, q[i]=l_i} \gamma_q \cdot e_s(q[i]q[i+1]) + \sum_{\forall q, \exists i, q[i]=l_i} e_r(q[i])$$

We do not consider the distance between two nodes when calculating energy cost for receiving data.

(ii) $e_{comp}$ is the energy consumed by the computation that is mainly due to the onboard processor, such as a microcontroller, DSP chip, or FPGA. These devices consume energy proportional to the number of processing cycles, as well as the maximum processor frequency $f$, switching capacitance $\mu$, and hardware specific constants $k$ and $\beta$, respectively.

Figure 5: Illustration of one dimensional signal indicating, sampling interval, sampling points, and how to sample it

The number of cycles required to perform a task on the amount of samples ($m$) are estimated according to the computational complexity $O(m)$, which describes how many basic operations, i.e., averages, additions, multiplications, etc., must be performed in executing the task. The computational energy to complete a task can be calculated according to:

$$e_{comp} = O(m) \cdot \mu \left( \frac{J}{k} + \beta \right)$$

(iii) $e_{ADC}$ is the energy consumed by the ADC. In the sampling, two of the modules are the most important, namely the ADC and the sensor itself, when they need energy. As in most cases, if the samples come at fixed time intervals, the average energy can be related to the energy per sample and the number of samples acquired. However, in the case of event sensitive adaptive sampling in e-Sampling, the energy cost can vary due to $m$ and $Rc$ (for symbol description, refer to TABLE 1). Thus, $e_{ADC}$ is proportional to $m$ and the sampling rate used. We calculate the remaining energy ($E_{rem i}$) reserved for sampling on sensor $i$ at the beginning of a given monitoring round $I$. Let $E_{req i}$ be the maximum energy required on $i$ (which is equivalent to $E_i$) for a set of $a$ actions in $I$. We define the system lifetime $T$ to be the total number of rounds of monitoring data collection before any battery runs out of energy:

$$T = E_i \frac{r_{em}}{E_{req}} / E_i \frac{r_{eq}}{E_{req}}$$

3.2.2: Algorithm Designed
We design method as follows. In the beginning of an interval, sensors start short and recurrent bursts of sampling at a high-rate ($Rh$), and examine these samples to analyze $Fh$. The
sensor probes the entire bandwidth that is opposite of checking only the bandwidth visible at some sampling point when sampling at another time at a low/adjusted rate ($R_l$). $D_h$ is the duration of each burst of sampling at $Rh$ that is followed by $D_l$, where the sampling rate used is calculated based on findings in $F_h$.

**Pseudo Code of Algorithm**

DecentralisedControl {  //1st stage data reduction
  While (True) {
    S.RateComp in $D_h$=True {
      /*start Sampling Rate Computation at the beginning of the system or at a certain interval */
      Run the Algorithm 1; // Sampling rate and interval adaptation
      Compute new $R_e$ ; // Set a new sampling rate
      Compute $D_t$;}} // set the duration for the new rate
    ComputeEventIndication{  //2nd stage data reduction
      Run the Event Indication Algorithm
      If indication.Strength>=40%
      Transmit the indication;
      Else transmit an acknowledgment;
    }

**IV. WORK DONE**

In this section we are discussing the practical environment, scenarios, performance metrics used etc.

**5.1 Simulation Platform**

For the simulation of this work we have to need the following setups requirement for the same
1) Cygwin: for the windows XP
2) Ns-allinone-2.32.

**5.2 Network Scenarios**

Inputs: Following are parameters which are varied for these simulations:

- Number of Sensor Nodes
- Mobility Speed
- Number of Base Stations
- Size network area

**Mac protocol**

1) 802.11

**Scenarios**

1) 10 nodes
2) 20 nodes
3) 30 nodes
4) 40 nodes
5) 50 nodes
Routing Protocols
1) DELUGE
2) DSR
3) EDiff (Proposed Routing Protocol)

Table 2: Network Scenario

<table>
<thead>
<tr>
<th>Number of Nodes</th>
<th>10/20/30/40/50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traffic Patterns</td>
<td>CBR (Constant Bit Rate)</td>
</tr>
<tr>
<td>Network Size</td>
<td>500 x 500 (X x Y)/670 x 670/1000 x 1000</td>
</tr>
<tr>
<td>Max Speed</td>
<td>2 m/s</td>
</tr>
<tr>
<td>Simulation Time</td>
<td>100sec</td>
</tr>
<tr>
<td>Transmission Packet Rate Time</td>
<td>10 m/s</td>
</tr>
<tr>
<td>Pause Time</td>
<td>2.0s</td>
</tr>
<tr>
<td>Routing Protocol</td>
<td>DELUGE/DSR/EDIFF</td>
</tr>
</tbody>
</table>

5.3 Performance Metrics

Energy Consumption: The metric is measured as the percent of energy consumed by a node with respect to its initial energy. The initial energy and the final energy left in the node, at the end of the simulation run are measured. The percent energy consumed by a node is calculated as the energy consumed to the initial energy. And finally the percent energy consumed by all the nodes in a scenario is calculated as the average of their individual energy consumption of the nodes.

\[
\text{Average\_Energy\_Consumed} = \frac{\text{Sum\_of\_Percent\_Energy\_Consumed\_by\_All\_Nodes}}{\text{Number\_of\_Nodes}}
\]

Throughput: The Maximum bandwidth can be calculated as follows:

\[
\text{Throughput} \leq \frac{\text{RWIN}}{\text{RTT}}
\]

where RWIN is the TCP Receive Window and RTT is the round-trip time for the path. The Max TCP Window size in the absence of TCP window scale option is 65,535 bytes. Example: Max Bandwidth = 65535 bytes / 0.220 s = 297886.36 bytes/s * 8 = 2.383 Mbit/s.

End To End Delay: End-to-end delay refers to the time taken for a packet to be transmitted across a network from source to destination.

\[
\text{d}_{\text{end-end}} = \text{N} [\text{d}_{\text{trans}} + \text{d}_{\text{prop}} + \text{d}_{\text{proc}}]
\]

where
\[
\text{d}_{\text{end-end}} = \text{end-to-end delay}
\]
\[
\text{d}_{\text{trans}} = \text{transmission delay}
\]
\[
\text{d}_{\text{prop}} = \text{propagation delay}
\]
\[
\text{d}_{\text{proc}} = \text{processing delay}
\]
\[
\text{N} = \text{number of links (Number of routers - 1)}
\]

5.4 Results Obtained

Following figures 6, 7 and 8 are showing the achieved so far for the performance metrics such as throughput, energy consumption and packet delivery ratio (PDR) respectively. In all this
results, it is clearly showing that proposed EDIFF protocol showing better performance as compared to existing protocols.

![Avg. Throughput vs. WSN Scenarios](image1)

**Figure 6:** Avg. Throughput vs. WSN Scenarios

![Average Energy Consumption vs. WSN Scenarios](image2)

**Figure 7:** Average Energy Consumption vs. WSN Scenarios

![PDR vs. WSN Scenarios](image3)

**Figure 8:** PDR vs. WSN Scenarios

V. CONCLUSION AND FUTURE WORK

In this paper we have presented the differential software reprogramming method for wireless sensor networks. The existing methods for wireless reprogramming are having many limitations as they are transmitting the large amounts of information over the network, rewrite the whole internal ash memory and require large amount of external ash memory. Radio transmission and ash memory writing are two of the most energy and time-intensive operations in wireless sensor networks. In this
paper we presented, a new reprogramming scheme, which reduces the amount of information to be transmitted, reduces internal flash memory writing and eliminates necessity of external memory without modifying the compiler. In this paper we have added three results which are showing the proposed differential method outperforming existing method in terms of energy consumption by varying number sensor nodes as well as other network performance metrics. In future work, the investigation over other routing performances such as end to end delay, throughput, jitter etc.

REFERENCES