CHAPTER 4
SECURE TINYRPL ROUTING PROTOCOL

4.1 INTRODUCTION

In order to meet the routing requirements and challenges in Low power and Lossy Networks (LLNs), the Internet Engineering Task Force (IETF) and Routing over Low power and Lossy Networks (ROLL) working group have designed a new standardized IPv6 routing protocol called RPL. The prime objective of RPL is to reduce the energy consumption and provide routing paths to support all types of traffics such as P2P, MP2P and P2MP. The IP for Smart Objects (IPSO) alliance is a leading organization that defines Internet of Things (IoT) and IP for small devices. It is needed to enhance the ability of resource constrained devices to communicate with the IP world without any application gateway. To achieve this, the IETF has specified a standard called IPv6 for Low power Wireless Personal Area Networks (6LoWPAN) which supports web services. The main objective of this research work is to utilize the open source routing protocol RPL and to propose a novel Secure TinyRPL protocol. Though adding security features is a burden to the RPL stack, secure routing in 6LoWPAN environment is inevitable.

4.2 DESIGN OF RPL – OVERVIEW

RPL is a Proactive Distance Vector IPv6 routing protocol for LLNs which specifies the method of building a Destination Oriented Directed Acyclic Graph (DODAG) with Objective Function (OF) and several
metrics/constraints. The RPL defines application specific routing requirements for the sensor networks. The RPL separates packet processing and forwarding as a separate task from the routing optimization. It provides a mechanism to disseminate information over the dynamically-formed network topology. The objective function usually operates on the combination of these metrics and constraints. Finally it computes the ‘best’ path. The deployments vary with different objectives which can operate on a single node and the mesh network formed may need to carry traffic accordingly with different requirements of path quality. A DODAG may use the OF based on proper metric/constraint requirement such as

i) A best path with Minimum Expected Transmissions Count (ETX) with prevention of non-encrypted edges.

ii) A best path with minimum delay while preventing nodes with battery life alone.

The DODAG formation is carried out with certain rules apart from the metrics/constraints used in OF (load-balancing, count of parents and parents as backup).

RPL builds a DODAG over any real time network on satisfying specific criteria and hence creates a logical routing topology. The construction of multiple active routing topologies which can carry information for different applications is decided by the network administrator. A node can join one or more DODAGs (RPL Instances) and can support traffic with necessary metric/constraints defined as the characteristic of that DODAG. Thus it facilitates the movement of the traffic up and down the links of that specific DODAG.
4.2.1 Topology Formation

The construction process of the topology is initiated at the root or LBRN (LowPAN Border Router Node), which is configured with a unique IPv6 address by the system administrator. There could be multiple roots configured in the system for multiple instances. The DODAG related information is exchanged within the topology with the help of four ICMPv6 control messages identified by a single byte code field.

The ICMPv6 Control messages used in a DODAG formation are

1. DIS (DODAG Information Solicitation)
2. DIO (DODAG Information Object)
3. DAO (DODAG Destination Advertisement Object)
4. DAOK (DODAG Destination Advertisement Object Acknowledge)

The DIS message is used for soliciting the DODAG information proactively. It helps in triggering DIO message transmissions. The information about the DODAG structure is advertised using DIO messages. These messages are received and processed by the neighbouring nodes in the communication range of the root node. The decision of joining a DODAG tree is done by a node upon receiving multiple DIOs from potential nodes. The decision is based on the characteristic of DAG, Objective Function, path cost and adopted local policy. A node upon joining the network has a route towards DODAG root enabling ‘upward’ traffic with the help of DIO’s. As a counter message, DAO messages are used for advertising reach-ability towards destination nodes with prefix information supporting ‘downward’ traffic. In order to establish secure communication of routing messages,
secure variants of the above mentioned RPL messages such as SDIS (Secure DODAG Information Solicitation), SDIO (Secure DODAG Information Object), SDAO (Secure DODAG Destination Advertisement Object) and SDAOK (Secure DODAG Destination Advertisement Object Acknowledge) can be used. This option is still an open ended task for grabbing the attention of researchers.

4.2.2 RPL Instance ID

Multiple independent RPL instances may be available within a given LLN. When an RPL node belongs to multiple instances, it may behave as a router in one instance and a leaf node in the other. There are two types of RPL Instances: Local and Global. RPL divides the RPL Instance ID space between Global and Local instances to allow for both coordinated and unilateral allocation of RPL Instance IDs.

Global RPL Instances are unique to the entire LLN that are long-lived and coordinated by having one or more DODAGs. The RPL Instance ID field is a single byte field present in every control message having 128 instances. Individual DODAGID is assigned to a root node named as Local RPL Instance which is allocated in a unilateral manner. Local RPL Instances can be used for constructing DODAGs in supporting a future on-demand routing solution. There is an RPL instance ID field (1 Byte) in every RPL control message with the D flag designated for specifying destination address (if set) or the source address (if cleared).

4.2.3 DODAG Construction and Routing Procedure

The DODAG root is called as a ‘parent’ node which advertises itself to its neighbour nodes with DIO messages and computes its ‘rank’. If the coordinated node is configured to act as a router, it starts advertising the
graph information down the network with new information to its neighbouring peers and forms a sub-DODAG. If it is a “leaf node”, it has to simply join the DODAG and is not supposed to send any DIO message. Parent selection, information advertisement and addition of route are carried out by the neighbouring peers on repeating the same process with DIO messages. A DODAG is built with the linking edges from the root node to the leaf nodes by this rippling effect. RPL supports efficient routing paths for three different traffic patterns: Multipoint to point (MP2P), Point to multipoint (P2MP) and Point to point (P2P) in an LLN. In an MP2P (Multipoint-to-point) forwarding model, the leaf node of the DODAG has reach-ability towards the root node by simply forwarding the data packets to its preferred parent using UPWARD routing. Each node computes its rank depending on its relative position with respect to the root node in the topology which also helps in loop avoidance.

Figure 4.1 DODAG Construction Process
The DODAG formation process is as shown in Figure 4.1. A root node advertises itself with the smallest rank value in that DODAG by setting its DAG preference to be the least and parent set to be empty. Upward route discovery process enables a node to join a DODAG by discovering neighbours of the DODAG and identifying a set of parents.

RPL has provision to operate with multiple DODAG versions. Each DODAG version is uniquely identified by RPL Instance ID, DODAG ID and DODAG Version Number. A node is considered as the member of the DODAG if all its parent sets belong to that version or if it is the root node and will not send any DIO that is not the member of that version. The root node has the preference to increment its version number and become a part of the new DODAG Version. Other nodes in the DODAG cannot advertise a DODAG Version Number greater than that it has heard before.

The construction process of DODAG follows a Neighbour Discovery procedure in which the two major steps are

1. DIO messages are broadcast by DODAG root to build downward routes from root to client nodes.

2. Transmission of Unicast DAO messages from the client nodes to build routes in upward direction.

In the process of constructing a new DODAG, the root broadcasts a DIO to announce its Rank and its DODAGID information. This activity facilitates the nodes to learn their positions in the DODAG. The objective function used is identified from the Objective Code Point (OCP) field of the options present in the DIO configuration. This message is received by a client node. It may either be a node willing to join the DODAG or an already
existing node. When a node wants to join the DODAG upon receiving the DIO message,

- It adds the sender address to its list of parents
- Compute its rank using the Objective Function, such that the node’s rank is higher than that of its parents
- Forward the DIO with updated rank.

The client node takes one node as the preferred parent among its parents’ list as the default node for forwarding inward traffic. Upon receiving a second DIO message a node in a DODAG decides to follow any one of the following three steps:

- The DIO message is discarded
- The message is processed to maintain its position
- Improve its position by getting a lesser rank.

A node changing the rank will have to discard its entire previous parent list and create a new list to avoid looping. The flowchart shown in Figure 4.2 depicts the DODAG router operation. After the construction, all client nodes will have a default upward route which can transmit inward traffic. A default route is formed by each node with its preferred parent. If the Mode of Operation flag in the DIO is not zero, further downward routes are supported and maintained from the root to the client nodes. In this case, all client nodes send a unicast DAO message to determine the upward route information. A complete route is established between the root node and the client nodes in a DODAG during this reverse path. The address of the visited nodes is updated in DAO message along the upward route.
4.2.4 Storing and Non-Storing Modes

Each node in the DODAG helps in building the routing state so that DOWNWARD information can flow in the LLN. The prefix information contained in the DAO messages that are received from Sub-DAG nodes is stored in each node. Memory and scalability issues of the nodes in the network make them handicapped to translate and store all the prefix entries in a routing table. Hence nodes are classified into two types as storing and non-storing modes.
In non-storing mode, DAO message is unicast to the DODAG Root reporting about its parents. The DODAG Root is capable of storing the entire routing information from the DAO message for further communication. In order to deliver an information packet from the root to the destination node, DODAG root incorporates the consolidated routing information into the routing header of the source. This packet is forwarded to the next hop sibling node. Every intermediate node in the path towards the destination forwards the information to its child node after examining the routing header information. The above process is repeated until the information packet is received by the destination node.

But in the storing mode, the DAO message is unicast from the child node to its preferred parent node. Each node stores the routing information in the routing table and forwards the packet to its ancestor. This routing table is examined for every next hop while delivering the information from the DODAG root to the destination. The flow chart in Figure 4.3 illustrates the steps involved in DAO process of DODAG formation of RPL in storing and non-storing mode.

Usually a trade-off for deciding about storing and non-storing mode of operation in an LLN is made depending on the computing resources such as power, bandwidth, memory consumption etc. For example, storing mode occupies memory space to store routing tables while non-storing mode has increased the size packets exhausting the power and bandwidth during communication even though each node does not have a routing table entry.
4.2.5 Node Movement between DODAGs

In order to provide enhanced scalability, the routing topology can be separated among multiple DODAGs within an instance. A node can only be associated with a single DODAG within a specific instance but can join multiple instances. However, the routing protocol permits the movement of a node from one DODAG to a different one within the framework. It is depending on specific fundamental rules which satisfy that instance. As the node moves to another DODAG it has to discard its current parent set, re-compute the new rank based on its current position and perform parent selection again.
4.2.6 Dynamic Routing Metrics and Constraints

Routing metrics used in routing protocols help in computing the shortest path whereas constraint based routing are used in pruning both links and nodes which are not meeting the required constraints in an LLN. Node level metrics can be residual energy of the node, hop count of the node etc. ETX, latency and reliability are some of the Link Quality level metrics. The protocol adapts dynamically to metric and constraint changes in the network that can be recorded or accumulated.

4.2.7 Loop Avoidance and Loop Detection

Traditional networks have a major issue of temporary loops formed by changes in routing topology and synchronization among nodes. It ends up in loss of packets because of congestion in the links and expiry of TTL field set in the packets. Light of research is thrown towards various optimization techniques in avoiding these loops. Loop detection and avoidance with minimum data rate is an important aspect of RPL in establishing a network of smart objects. But there is no guarantee for the absence of loops rather RPL tries to detect loops using data path validation and avoids them. The rules of loop avoidance rely on the ranking property of the nodes. In addition to these rules, loop detection is mandatory as loop avoidance is not guaranteed. This can be achieved by setting bits in the routing header for validating the data path. As the packet moves up and down due to temporary loops, these bits help in the detection of loops. Let a parent node send a packet with the down bit set to its child that forwards it to the next hop node. After receiving, the receiving node may identify an inconsistency in the packet if it is destined in the up direction in the routing table. This packet on identification has to be discarded due to looping or inconsistency.
4.2.8 Global Repair and Local Repair

In any routing protocol, the capacity to overcome failures in the routing topology is a key feature named as Repair. RPL supports two complementary repairing mechanisms for both link and node failures viz. Global repair and Local repair. When network failure is identified either due to the unavailability or a missing router to forward the packet to destination, the local repair mechanism is triggered of path a key feature for any routing protocol and refers to the ability to repair by finding an alternate path/parent within the DODAG version. This doesn’t have any global inference on the entire DODAG. When the local repair results in the divergence of optimum routing shape, a need for Global repair is created insisting the necessity of rebuilding the DODAG. A Global repair increments the DODAG version number and initiates a new DODAG version triggered at the root node resulting in additional control traffic flow in the network. An additional cost is incurred to manage this traffic.

4.2.9 Trickle Timer Management

Trickle timer management is another aspect in which RPL showcases its uniqueness from other routing protocols that operate in less-constrained environments. In LLNs, particularly when the network has to be constructed with energy efficient devices, it’s essential to limit the control plane traffic (RPL) in the network. Mostly routing protocols use periodic keep-alives to establish routing contiguity and to keep routing tables up-to-date. In a resource constraint environment, the above process could be expensive. Trickle Timer mechanism is a simple, adaptive, energy efficient and scalable algorithm. It is used in pruning redundant control traffic and monitoring the consistency or stability of the network.
The configuration parameters of the algorithm are the $I_{\text{min}}$, $I_{\text{max}}$ and $\gamma$.

$I_{\text{min}}$ → Minimum Interval Size

$I_{\text{max}}$ → Maximum Interval Size

$\gamma$ → Redundancy Constant

DODAG formation is considered as a consistency issue in RPL which utilizes the trickle timers for deciding the requirement of multicast DIO messages. Issue of DIO messages are governed by the dynamic timers of trickle algorithm using DIO interval minimum and DIO doubling rate. Some of the events are treated particularly as inconsistencies in the network such as disagreement of data among the neighbours. This Trickle timer mechanism has a facility of resetting the timer during inconsistency of the network. For example, once a node detects a loop within the network, or, once a node joins the network or moves within the network, it is considered as an inconsistency in the network.

The stability of the network is affected during following instances which cause the reset of trickle timers:

i) When a node joins a DODAG.

ii) When a node moves within a DODAG

iii) When a node receives a new DIO message from a DODAG parent.

iv) When looping occurs

v) When an inconsistency in a metric communicated in DIO message is identified
When the network is stabilized the interval of the trickle timer is increased resulting in fewer DIO messages being sent in the network. As inconsistencies are detected, the trickle timers are reset in the nodes to increase the frequency of DIOs. With the help of this mechanism, the RPL has control over the stability of the network and the frequency of DIO messages is increased in the vicinity where the inconsistency is detected. In other words, as the network becomes stable, the number of RPL messages decreases. When an inconsistency is detected the timers are reset so that the issue can be fixed swiftly. The main advantage of the trickle timer implementation is that its simplicity in coding and its effortless implementation.

4.3 TINYRPL

![Figure 4.4 Architecture of TinyRPL](image-url)
TinyRPL is implemented in TinyOS2.x as a sample prototype routing protocol that provides robust multi-hop routing over unreliable wireless links and defines many application specific requirements for Sensor Networks as shown in Figure 4.4. TinyOS is an event driven OS for embedded devices developed by U.C. Berkeley. This OS is implemented with component oriented programming abstractions that provides code modularity and facilitates component reuse at the cost of steep learning curve and code complexity (Levis P et al 2005). The main features are non-preemptive scheduler and multiple abstractions including communication interfaces and hardware timer virtualization. The research community has made laudable efforts to implement and provide open source standards for the development of wireless sensing systems for real time implementations.

The application layer supports all real time IP based applications using HTTP, CoAP protocols. The transport layer supports TCP, UDP protocols as communication protocols. For the wireless communication, UDPecho component is used in the transport layer of TinyOS. Blip 2.0 and TinyRPL are open source standardized implementations (available at http://code.google.com/p/tinyos-main/) using TinyOS for WSN environment in the network layer. Blip is the Berkeley Low power IP stack standard designed for real world applications and it provides space for developing customized solution for new IP based protocols. It has the latest header-compression standard, dhcpv6 for address assignment and RPL routing.

TinyRPL released by IETF routing standard does not implement the entire RPL. It separates packet processing and forwarding from the route optimization objective and provides a mechanism to disseminate information over dynamically formed network topology. The routing layer is responsible for relaying across multiple hops separating source and destination nodes. It is divided into a forwarding engine, which utilizes the routing table for
neighbourhood delivery of packets and a routing protocol, which populates the routing table in storing or non-storing mode as detailed earlier. It operates on the top of IEEE 802.15.4 MAC layer.

The architecture provides multiple MAC layer implementations such as preamble sampling, Low power listening MAC etc. It is obvious that the architecture supports the standardization of two different organizations; in the Network layer as per IETF specifications and in the MAC layer as per IEEE specifications. The Physical layer is responsible for the physical connectivity of the network topology formed with the hardware motes. It consists of radio transceiver chip, flash memory for storage, serial port for communication, LEDs with interfaces and sensors for monitoring.

4.4 SECURE TINYRPL

As LLNs are the part of several critical infrastructures such defense and military applications in real world, security is an important design consideration. The security implementation into TinyRPL is left open as an optional feature to the researchers and it is considered to be a burden to the RPL stack as it consumes the resources of the motes. As discussed earlier about the importance and necessity of the security framework for Wireless Sensor Network environment, an effort has been made to implement the proposed Secure TinyRPL in the place of TinyRPL. The proposed architecture is depicted clearly in Figure 4.5. The protocol consists of cryptographic algorithms for establishing confidentiality and Message Authentication code for establishing authenticity and integrity. It also consists of a Key Management Module (KMM) and Random Key Generator submodule to support the key management procedure.

In a resource constrained environment, providing sophisticated security mechanisms in RPL is not possible economically. Implementation of
security in RPL can be done by operating the motes in any one of the three optional security modes as discussed in,

i) Unsecured mode

ii) Pre-installed mode

iii) Authenticated mode.

The unsecured mode is the mode with link layer security establishment which is not sufficient to guarantee the integrity of routing messages.

Pre-installed mode specifies the generation and transfer of secure control messages such as SDIS, SDIO, SDAO and SDAO-ACK messages. Suitable cryptographic mode of operation may be implemented with the pre-installed keys stored in the nodes to establish message authentication along with confidentiality.

Third mode is the Authenticated mode in which a node can join an RPL instance only as a leaf node with its pre-installed key. But to join as a router, it requires authentication form an authority. This mode is not discussed elaborately and kept open in the IETF draft (Winter et al 2010).

Secure Tiny RPL is an effort made for the implementation of secure communication of routing messages in the pre-installed mode with the analysis made on appropriate cryptographic algorithm. The symmetric key algorithms appropriate for the resource constrained environment are analyzed with RC5, SKIPJACK, Blow fish, IDEA, Xeta. Finally the implementations are compared with RC5 and SKIPJACK algorithms.
The emerging Low power networks have fast varying requirements and characteristics. The objective function does not specify the metrics/constraints but operate on them to form the DODAG. RPL does not define any specific routing metrics, path cost, control packet overhead, forwarding policies or loss of secure connectivity. The selection of these metrics to meet the requirements is still open. In this work, TinyOS is used to evaluate the performance of secure connectivity in the newly proposed standards and compare with the existing routing protocol. Hence efficient key management protocol for establishing security is taken under consideration.

This work describes the Secured TinyRPL with suitable key management solution as shown in Figure 4.5. RPL is the routing protocol for LLN and does not provide secure communication of messages. The proposed
implementation uses symmetric encryption algorithms such as SKIPJACK, RC5 for providing confidentiality and Message Authentication Code (MAC) for verifying the authenticity and integrity of messages.

4.4.1 Key Management Procedure

As Secured TinyRPL is operated in the pre-installed mode, the nodes are loaded with secret key before deployment and are unaltered till next deployment. Key Management Module is built into DODAG root and is responsible for the generation, storage and distribution of keys. KMM based on symmetric key management is centralized at the root node, which controls the generation, distribution and regeneration of authentication and encryption keys. Key generation is performed by the Random Key Generator (RKG) sub-module. The renewal of keys is performed periodically and the period of renewal is set at the compilation time. The distribution of newly generated keys is broadcast after encryption with the pre-distributed keys. Only legitimate nodes with correct pre-installed keys can decrypt the new key packets.

4.4.2 Random Key Generation (RKG) Sub-Module

The randomness of the sensor traffic flowing from the root node to the gateway through serial communication is used as the random value for the Pseudo random key generation. Limitation lies on the memory constraints of allocating memory space for the Random Key Generator. It is designed as a bitwise recursive XOR operation of the seed value of 16 bit with the random value depending on the root-gateway communication denoted by the following equation.

\[
seed_i = seed_{i-1} \oplus (w \gg 16 \ (i \mod p)),
\]

\[\text{(4.1)}\]
\[ seed_0 = 0 \] 

where

- \( i \rightarrow \) current value
- \( 0 \rightarrow \) Initial value
- \( w \rightarrow \) Random value of the root-gateway communication (16 bit)
- \( \gg_M \rightarrow \) M-bit right circular shift operation
- \( p \rightarrow \) Largest 16 bit prime number

The equation (4.1) is computed as bit wise XOR of the random value and seed value along with the current seed value is updated. In TinyOS, pseudo random generator RandomMlcgp component is seeded with the seed value and its complement is used during the generation of encryption and authentication keys periodically.

### 4.4.3 Initialization Vector

The Encryption scheme used in Secured TinyRPL utilizes Initialization Vector (IV) of 8 bytes length. Therefore the vector repeats after only \(2^8+1\) bytes. IV is having the following structure: \( dst||AM||l||src||ctr\) where \( dst\) is the address of the destination node (transmitter), \( AM\) is the type of the active message handler, \( l\) is the data payload length, \( src\) is the address of the source (receiver), and \( ctr\) is a 16-bit counter (Corin RD et al 2011). However the overhead due to IV is limited by borrowing 4 bytes (destination address, AM type, length) from header. So IV occupies only 4 bytes.

Since \( ctr\) value starts at 0 and incremented upon message transmission by the sender, which is used as a start value for IV. Otherwise it will repeat the same IV for all nodes usually. Also generating IV’s randomly
gives collision after $2^{16}$ packets has been sent. In this work, the $src||ctr$ format strives to maximize the number of packets sent before IV reuse occurs. For a network of $n$ nodes all transmitting at the same rate, it is expected that $n*2^{16}$ packets have to be sent before IV reuse occurs.

Cipher Block Chaining (CBC) mode of encryption provides robustness to information leakage when IV’s repeat. CBC mode of encryption for 8-byte block cipher results in message expansion which increases the overhead. Therefore Cipher Text Stealing technique is used which keeps the cipher text length equal to the plaintext [NIST].

$$C_0 = E_K[IV] \quad (4.3)$$

$$C_i = E_K[C_{(i-1)} \oplus P_i] \quad 0 < i < n-2 \quad (4.4)$$

$$C_{(n-1)} = E_K[C_{(n-2)} \oplus P_{(n-1)}]$$

$$= C^*_{(n-1)} || C^{**}_{(n-1)} \quad (4.5)$$

$$C_n = E_K[C_{(n-1)} \oplus P_n]$$

$$= E_K[C^*_{(n-1)} \oplus P^*_n || C^{**}_{(n-1)}] \quad (4.6)$$

where $P$ is the plaintext (message) which is divided into “n” blocks, $b$ is the block size of the underlying block cipher, $d = 1-(n-1)$ $b$, $P_n=P_n^*||0$, $P_n$ is the last block padded with (b-d) zeroes and $*$ refers to the d bytes of a block and $**$ refers to the last (b-d) bytes of a block. SKIPJACK (Brickell) and RC5 algorithm are used separately for encryption. Although SKIPJACK provides reasonable security than RC5, the latter is flexible in terms of Block size, Key size and number of rounds. It also requires less memory. The choice of the
cryptographic algorithms can be changed depending on the application environment.

4.4.4 Message Authentication Code

Authentication and Integrity is provided by means of MAC (Bellare et al 2000). It uses CBC mode of underlying block cipher (Karlof et al]. MAC produces 4 bytes of output. The hash function for MAC is given by the formula

\[ H_i = E_{K_i} \left( H_{i-1} \oplus P \right), \quad 1 \leq i \leq n \]  

and \( H_0 = E_{K_1} (IV) \) is the cipher text, \( P \) is the message, MAC is the message authentication code. The resulting code from final block is again encrypted with a second key. Since there is a higher possibility of node compromising if the same keys are used for both CBC encryption and authentication, it can be avoided by using two keys for calculating the MAC value. An adversary has to try \( 2^{31} \) attempts to forge the packets. The constrained nature of the sensor nodes makes this not feasible. Further, the security of CBC MAC depends on the underlying block cipher.

4.4.5 Replay Attack Protection

For protection against replay attacks, the counter field provided in the security field of RPL is used. When set, the counter field acts as a delay protection. When reset, it acts as a counter. When the receiving node notices the counter field is reset, it checks for the increase in counter value for each control message it receives. Counter value is increased after each DIO or
DAO control message is sent. When the value is not increased, it rejects the RPL packet. Thus it provides protection against replay attack.

4.5 IMPLEMENTATION

The research work is implemented in TelosB motes with one node acting as a base station and five nodes acting as leaf nodes utilizing UDPSocket application. TelosB motes have 48KB of ROM and 10 KB of RAM. PppRouter application is used for the base station and UDPecho application is used for the leaf nodes. There are two interfaces in the PppRouter application: one is a serial interface between the mote and the base station and the other interface is its 6LoWPAN interface communicating over IEEE 802.15.4. UDPecho is a simple application that uses BLIP 2.0 protocol.

As discussed already BLIP is a TinyOS implementation for a number of IP based protocols using which one can form multi-hop IP networks consisting of various motes communicating over shared protocols. SKIPJACK algorithm uses 32 rounds and 80 bit key. RC5 algorithm uses 12 rounds. Since our work is implemented in TinyOS, the programming consists of modules and components. SKIPJACK encryption algorithm and RC5 algorithm are used as a separate module with Keys module for generating and updating the keys. Security module acts as a core interface between the RPLRouting modules and other components.

4.6 RESULTS AND DISCUSSION

Figure 4.6 shows the screenshot of the memory occupied while building the entire architecture stack for leaf node. During compilation of SKIPJACK Secure TinyRPL into the target device it is found that the 36 KB of ROM and 8 KB of RAM are consumed in the TelosB motes to be deployed as leaf nodes of the network.
Figure 4.6 Screen shot of Memory Occupation of Leaf Node

The screen shot of the memory occupied while building the entire architecture stack for the root node is depicted clearly in Figure 4.7. During compilation of SKIPJACK Secure TinyRPL into the target device it is found that 48 KB of ROM and 9 KB of RAM are consumed in the TelosB motes to be deployed as leaf nodes of the network. The main drawback identified is that the total capacity of TelosB is consumed and further implementation of application over these nodes is not made possible. Comparatively RC5 Secure TinyRPL occupies 0.9KB less compared to SKIPJACK Secure TinyRPL.
The screen shot of Packet transmission between the nodes in the established network with RC5 Secure TinyRPL is shown in Figure 4.8. The ping statistics show that all the transmitted packets are received without any loss of information as the motes lie in the communication range. It also displays the time needed for the data packets originated at the leaf nodes to reach the root node. The root node is connected via the USB port to the base station where the data packets are aggregated and stored in a file. Also the process is checked with a sniffer whether the transmitted control packets are secure. The SDIS, SDIO and SDAO messages are found to be secure and only the control messages without security can be revealed to visualize the routing information.
Figure 4.8 Screen shot of Packet transfer between motes of Network

No. of Packets Transmitted vs Time

<table>
<thead>
<tr>
<th>Time (ms)</th>
<th>RCS Secure TinyRPL</th>
<th>SkipJack Secure TinyRPL</th>
<th>RPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>155.65</td>
<td>167.35</td>
<td>162.11</td>
</tr>
<tr>
<td>50</td>
<td>825.11</td>
<td>823.73</td>
<td>821.33</td>
</tr>
<tr>
<td>100</td>
<td>1655.98</td>
<td>1664.2</td>
<td>1657.5</td>
</tr>
<tr>
<td>150</td>
<td>2499.783</td>
<td>2498.06</td>
<td>2497.8</td>
</tr>
<tr>
<td>200</td>
<td>3326.866</td>
<td>3333.6</td>
<td>3321.41</td>
</tr>
<tr>
<td>250</td>
<td>4156.7</td>
<td>4168.133</td>
<td>4152.15</td>
</tr>
</tbody>
</table>

Figure 4.9 Transmission time of Routing protocols
SKIPJACK algorithm takes extra transmission time of approximately 2 to 10 s for various numbers of packets such as 10, 50, 100, 150, 200 and 250 and RC5 algorithm takes extra transmission time of approximately 2 to 5 s for various numbers of packets as above when compared with RPL without security. It is obvious from Figure 4.9 that the transmission time of data packets is less for RC5 Secure TinyRPL than for SKIPJACK Secure TinyRPL.

![Image]

**Figure 4.10 Delay in Received packets of Routing Protocols**

It is evident from Figure 4.10 that SKIPJACK Secure TinyRPL algorithm has approximately 0.3 ms more delay for the transmission of various numbers of packets such as 10, 50, 100, 150, 200 and 250 and RC5 Secure TinyRPL algorithm has approximately 0.2 ms more delay for transmission of the same number of packets when compared with RPL stack without security.

SKIPJACK Secure TinyRPL algorithm has throughput in the range of 60-100 and RC5 algorithm has throughput in the range of 30-50. However,
it is clearly depicted in Figure 4.11 that the stability of throughput is high for RC5 Secure TinyRPL than that for SKIPJACK Secure TinyRPL.

**Figure 4.11 Throughput of Routing protocols**

**Figure 4.12 Memory Occupation of Routing protocols in Root node**
The memory consumption of the RPL protocol, RC5 Secure TinyRPL protocol and SKIPJACK Secure TinyRPL protocol in RAM and ROM of TelosB motes is shown in Figure 4.12. For a root node, TinyRPL stack without security occupies 45KB of ROM and 9 KB of RAM (i.e. occupies 93% of ROM and 90% of RAM space). Secure TinyRPL with SKIPJACK algorithm consumes 48 KB of ROM and 9 KB of RAM (i.e. almost the brim of ROM space and 90% of RAM space) and with RC5 consumes 47 KB of ROM and 9 KB of RAM (i.e. 97% of ROM and 90% of RAM).

![Memory Consumption](image)

**Figure 4.13 Memory Occupation of Routing protocols in Leaf node**

For a leaf node as shown in Figure 4.13, TinyRPL stack without security occupies 29 KB of ROM and 7 KB of RAM (60% of ROM and 70% of RAM). RPL with SKIPJACK Secure TinyRPL algorithm consumes 32 KB of ROM and 8 KB of RAM (67% of ROM and 80% of RAM) and with RC5 Secure TinyRPL consumes 31 KB of ROM and 8 KB of RAM (64% of ROM and 80% of RAM). The control message length is increased to 17 bytes due to
the addition of security fields. Since new keys are updated periodically, key source and key index occupies 9 bytes of packet.

A trial implementation of Secure TinyRPL with RC5 is used in a fall detection system that is implemented in real time. This can be used in health care application for patient monitoring. The details about the real time setup and its implementation are discussed clearly in Chapter 8.

4.7 SUMMARY

Security to RPL is implemented with two security algorithms. RPL implementation consumes 45 KB of memory for the base station and 29 KB of memory for the other nodes with no security features. Since TelosB motes have 48 KB of memory, only 3 KB of security coding could be added. While implementing SKIPJACK Secure TinyRPL, it has been identified that the protocol has reached the brim of the memory space and further implementation of the application is not possible using this algorithm. A real time setup for fall detection is implemented with RC5 Secure TinyRPL algorithm. Therefore security features such as confidentiality and replay attack protection only could be provided to the RPL with no authentication. To have an increased level of security, motes having higher memory capacity of Class III type discussed in chapter 2 can be used.

Even though TinyOS has a huge flexibility for any problem specific application design, it poses the need for code verification, debugging and optimization during execution and compilation time. Even after taking great pains to ascertain the correctness of the code and the performance of the application, there exist problems in communication among the nodes. Also the situation becomes still worse, during substantial increase in the number of participating nodes. These circumstances emphasize the value of conducting trial execution of network establishment using simulators.