Wireless Sensor Networking for Smart Campus Water Management

Prepared by Samanvita Sastry

Abstract—**This report gives a brief insight into some important features of wireless sensor networks. In this report, the constrained network conditions in which the wireless communication motes operate and the routing protocol preferred for such constrained networks have been described. Contiki operating system which is generally used for constrained wireless networks has also been explained. This report is the study done as the basis to understanding the Smart Campus Water Management project. The very basic knowledge required to study about the wireless networks are the IP addresses and how they are used in the transmission of data packets from source to destination. The constrained network considered in this project uses 6LoWPAN as adaption layer, routing protocol used is the IPv6 Routing Protocol for LLNs (RPL), operating system used to program the motes is Contiki (version 3.x). An enhancement to RPL known as LinkPeek, which aims at reducing the packet drops during network data transmissions is incorporated in this project.**

***Keywords*—Smart Campus Water Management, RPL, 6LoWPAN, IP Address, Contiki, LinkPeek**

I. Introduction

Smart Campus Water Management is a project in which water level sensors are placed inside the water tanks in the campus. These sensors send the water level data to their neighboring motes, each of which forwards the data to the next hop until the data has reached the network monitoring database, where the water levels can be observed. This is a great advantage in the present world water-crisis situation to save water, by noting the water levels during the day. This observation helps in calculating the amount of water leakage and also overflow, which can then be regulated manually or automatically.

In order to realise the Smart Campus Water Management project, it is essential to know a few basics about how the motes communicate wirelessly. These motes operate under very constrained conditions and have limited resources such as power, memory, etc. In order to provide the best possible performance with the available resources, these constrained networks known as Low Power and Lossy Networks (LLNs) are setup slightly differently when compared to the wired networks. This report aims at describing all the features of such networks and everything that is essential to setup such a network for the best possible performance with the availbale resources.

Section II explains about the IPv6 Routing Protocol for LLNs (RPL), Section III explains IPv6 over Wireless Low Power Personal Area Networks (6LoWPAN), Section IV about IP addresses and Section V explains about Contiki Operating System. Finally, in Section VI we will see how RPL can be modified to reduce the packet drops using LinkPeek.

II. RPL

IPv6 Routing Protocol for LLNs (RPL) is a routing protocol that is used for Low Power and Lossy Networks (LLNs) [1] [3]. Such networks operate under constrained conditions.

- The routing protocol design in this type of networks should be sensitive to how much data a network can handle the speed and the devices capabilities.
- LLNs may be transient and unpredictable. Thus the routing protocol must be robust and be prepared to deal with these network characteristics. In traditional networks any loss of connectivity triggers a desire to quickly re-converge and find alternate routing paths. This action is seen as overkill in LLNs due to the lossyness being transient and would unavoidably lead to lack of stability and unacceptable control plane overhead. A preferred model should be to smooth over the transient loss of connectivity and have a confidence-monitoring model before triggering a full re-convergence, in order to avoid the instability.
- Also, Routing in such networks should be capable of self-management and be able to heal itself without requiring human (explicit) intervention.

Traditional routing protocols are designed to minimize the convergence time due to the voice and video traffic requirements. However this approach if used for LLN would lead to routing instabilities, various oscillations and routing loops. Furthermore, smart objects do not send a large amount of traffic unlike voice and video traffic on high-speed IP networks, rather they operate under very constrained conditions. In these environments, it is a reasonable expectation that during transient instabilities the traffic is locally redirected to an alternate next hop without triggering a global re-convergence. Another aspect of these networks is the dynamic nature of the metrics. Traditional routing protocols have experimented and avoided the use of dynamic metrics due to risk of route oscillations and network instability. But unavoidably, in LLNs, node and link metrics change over a period of time (link quality going down due to interference, CPU overloading, node switching from line power to battery power, etc) and the routing protocol should be able to adapt to it. To help minimize resource consumption, RPL uses a slow proactive process to construct and maintain a routing topology but a reactive and dynamic process to resolving routing inconsistencies. RPL operates by creating and maintaining what is known as a Destination-Oriented Direct Acyclic Graph (DODAG). These graphs are generated using a number of control messages known as the DODAG Information Object (DIO), DODAG Advertisement Object (DAO), DODAG Information Solicitation (DIS) [1]. RPL also includes methods for detecting and avoiding loop formations. It uses an Objective Function (OF) in order to form the DODAG. This model represents a MP2P (multipoint-to-point) forwarding model where each node of the graph has reach-ability toward the graph root.

A. *RPL Instance*

An RPL Instance contains one or more DODAG roots [1] [3]. A RPL Instance may provide routes to certain destination prefixes, reachable via the DODAG roots or alternate paths within the DODAG. These roots may operate independently, or they may coordinate over a network that is not necessarily as constrained as an LLN. RPL also provides the ability to perform Multi-Topology Routing (MTR) thanks to the concept of a DODAG instance identified by an instance-id. The idea is to construct and identify multiple graphs (DODAGs) over the same physical topology. This provides a way to provide paths based on different optimization objectives as specified by the OF and the routing/constraint metrics. A node can only join a single graph within an instance-id but can be associated with several instance-ids simultaneously. This is helpful to build multiple routing topologies on a physical mesh network. For example, non-critical traffic should follow a path avoiding battery-powered nodes whereas more critical traffic should follow a path of minimum latency.

B. *DODAG Formation*

The graph building process starts at the LowPAN Border Router (LBR) [1], which is configured by the system administrator. There could be multiple roots configured in the system. The RPL routing protocol specifies a set of new ICMPv6 control messages to exchange graph related information -DIO, DIS, DAO.

C. *Objective Function*

The Objective Function (OF) consists of two parameters: metrics and constraints, based on which DODAG formation takes place. As the DODAG is formed each node gets a rank. Higher the rank of a node, the deeper it lies in the DODAG. There could be several OFs in operation on the same node and mesh network because deployments vary greatly with different objectives and a single mesh network may need to carry traffic with very different requirements of path quality. An Objective Code Point (OCP) is an identifier that indicates which OF the DODAG uses.

Metrics and Constraints: Metric is a scalar quantity which is used as the input for the best- path-selection in the process of DODAG formation. Whereas a Constraint is used as an additional criterion to trim links and nodes that do not meet any particular set of constraints as maybe required by the constrained network. Metrics and constraints can be either node-based or link-based [1]. Examples of node level metrics are node state attribute, node energy state etc., while link level metrics can be latency, reliability, link color etc. The metrics and constraints can be dynamic. Additionally, metrics can be recorded or accumulated. Recorded metrics carry distinct values of each path and the accumulated metric is an aggregation of values along the path.

D. *DIO*

This message is broadcasted initially by the border router. Every node which receives it decides whether the sender is an acceptable parent or not, based on which it either becomes a child of that particular node or does not. Once this step is completed each receiver in turn sends its own DIO [1] to its children. And the process continues until all the nodes in the DODAG have received DIOs. DIO is used for upward routing. If the node is a leaf node, it simply joins the DODAG and does not do anything. But, if the node is configured to act as a router, then it starts advertising the graph information with the new information to its neighbouring peers.

E. *DAO*

This message is intended to inform the parent node about the sub-DODAG formed by its child. All nodes send DAOs [1] to their parents and this information is critical for the downward routing. That is to say, the DAO messages are used to advertise prefix reach-ability towards the leaf nodes in support of the down traffic. a node or root can poll the sub-dag for DAO message through an indication in the DIO message. As each node receives the DAO message, it processes the prefix information and adds a routing entry in the routing table. It optionally aggregates the prefix information received from various nodes in the sub- dag and sends a DAO message to its parent set.

In case of non-storing mode, the DAOs are cumulative and the entire information about the tree is stored at the border router. In the case of storing mode, each node contains a routing table which stores the information of all of its sub-DODAGs. The nodes include the parent information in the transit-info field of the DAO message.

F. *DIS*

The DODAG Information Solicitation message is used by the nodes to proactively solicit graph information (via DIO) from the neighboring nodes when it becomes active in a stable graph environment using either the poll or the pull model of retrieving graph (network) information.

G. *Rank*

Rank is a unique number assigned to each of the nodes in a DODAG. The scope of Rank is a DODAG Version. establishes a partial order over a DODAG Version, defining individual node positions with respect to the DODAG root. Usually the rank of a parent node and a child node differ by a value which can be configured, known as the MinHopRankIncrease[1]. Also, as we will see later, there exists another parameter known as the MaxDepth in order to avoid loops in the DODAG. However, the rank of a node need to remain same throughout the operation of the network. Based on the circumstances and conditions, the rank value of any node can change as long as it follows the rules of rank assignment. The Rank is not a path cost, although its value can be derived from and influenced by path metrics. The exact calculation of the Rank is left to the OF.

H. Loops

Loop detection and avoidance is a differentiating aspect of a routing protocol for smart object networks (with constrained network conditions) compared to traditional networks. In traditional networks, temporary loops are formed due to topology changes and also because of lack of synchronization between its nodes. These loops need to be detected as quickly as possible to avoid packet drops and link congestion in a constrained network, therefore various optimization mechanisms have been proposed and put in place to avoid such micro-loops. The following are a few such mechanisms.

- Loop avoidance: RPL specifies two rules for loop avoidance. These rules rely on the rank property of the nodes. 1. As a part of the max_depth rule, a node is not allowed to select as a parent a neighboring node that is deeper (i.e. node with a higher rank) such that the node will end up advertising a value (node_rank+max_depth), where max_depth is a configurable value specified at the root. 2. A node is not allowed to be greedy and attempt to move deeper in the graph to increase the number of parents. These two methods however cannot guarantee that absolutely no loops will exist in the DODAG formed by RPL. There are good chances that despite using the max_depth rule, loops will still be formed. Hence, there arises the need for a loop detection and elimination mechanism.
- Loop Detection: RPL employs Loop Detection mechanism as it is very critical for the efficient functioning of the DODAG. This can be achieved by setting bits in the RPL routing header information specially dedicated to signally the presence of loops in the formed DODAG. Processing of these bits is included as a part of the data-path validation for RPL. The bits are set and processed as the packet moves in the DOD AG along the edges (links) of the graph (network) and check for any inconsistencies in the values which would indicate the presence of loops. Most commonly this mechanism is implemented using a single bit. Loops in the DAO path can be detected by using this down bit in the RPL routing header. When a node sends a packet destined to any of its children in the downward direction, it sets this down bit and forwards the concerned packet to the next hop node. On receiving the packet with the down bit set, if the routing table of the receiving node indicates that the packet has to be forwarded in the up direction in order to reach its destination, it means that there is an inconsistency in the formed DODAG and that the DODAG consists of a loop. The packet needs to be discarded and a local repair needs to be triggered. However this is not the standard optimization for loop detection; similar other optimization mechanisms may also be employed.

I. DODAG repairs

In a wireless network, since a dedicated medium does not exist between the connected nodes, inconsistencies may creep up during the operation of the network. In RPL, the DODAGs [1] are formed based on a metric like RSSI. It is possible that due to some obstacle or mobility a particular link of the DODAG so formed may become incapable of carrying packets between its nodes either temporarily or permanently. In such situations the network is no longer complete and not all packets can be transmitted successfully. It is then necessary to find an alternative to the existing link. This process of finding and setting up an alternate link as a replacement to a disturbed link is known as repair. There are two types of repair done by RPL- local repair and global repair [1] [3]. However, care must be taken to avoid triggering a re-build in transient conditions as it may lead to inconsistencies in the graph (network).

- Local Repair: When a link or neighboring node fails or becomes temporarily unavailable, the node has no other route to forward its packet in the upward direction, then a repair is triggered to quickly find an alternate parent/path. This technique of finding an alternate parent without resolving to rebuilding the entire DODAG is known as local repair.
- Global Repair: As local repairs take place the graph may start to diverge from its optimum shape. At some point it might be necessary to rebuild the graph (DODAG). The mechanism of rebuilding the entire DODAG is known as Global Repair. However, global repair should be minimised as much as possible since it takes a lot longer to carry out the rebuilding of the entire graph when compared to just finding an alternative to the poor link in the network.

J. Timer Management

Most routing protocols use periodic keep-alives (keep-alive is a message sent by one device to another to check that the link between the two is operating, or to avoid the breaking of this link) to maintain routing adjacency and to keep routing tables up to date. But this would be costly in LLNs where resources are constrained and scarce. As a solution to this problem, RPL uses an adaptive timer mechanism called the trickle timer. This mechanism controls the sending rate of DIO messages. RPL treats building of graphs as a consistency problem and uses trickle timers [1] [3] to decide when to multicast DIO messages. The value of the trickle timer decreases as the network becomes more unstable, and it decreases as the network becomes more stable. That is to say, the rate of DIO messages sent is higher for an inconsistent state of the network and lower for the stabilised consistent state of the network. The inconsistencies in the network could result when a node detects a loop in the network, or when a node joins the network or moves within the network.

K. Security Modes

When made available, RPL nodes can operate in three security modes [1].

- Unsecured mode: In this mode, the RPL control messages are sent without any additional security mechanisms. Unsecured mode implies that the RPL network could be using other security primitives like the link-layer security to meet application security requirements.
- Pre-installed mode: Nodes joining an RPL instance have pre-installed keys that enable them to process and generate secured RPL messages.
- Authenticated mode: Nodes can join as leaf nodes using pre-installed keys as in pre-installed mode, or join as a forwarding node by obtaining a key from an authentication authority.

III. 6LoWPAN

6LoWPAN [4] [6] is the abbreviation for IPv6 over Low-Power Wireless Personal Area Networks. It is a networking technology that allows IPv6 packets to be carried efficiently within small link layer frames, such as those defined by IEEE 802.15.4. A powerful feature of 6LoWPAN is that although it was originally conceived to support IEEE 802.15.4 low-power wireless networks in the 2.4-GHz band, it is now being used over a variety of other networking media including Sub-1 GHz low-power RF, Bluetooth, Smart, Power Line Control (PLC) and low-power Wi-Fi.

6LoWPAN makes the technology ideal for projects such as home automation with sensors and actuators, street light monitoring and control, residential lighting, smart metering and generic IoT applications with Internet connected devices. Todays deployments use both 2.4 GHz and Sub- 1 GHz, building on the IEEE 802.15.4 advantages including support for large mesh network topology, robust communication and very-low power consumption.

A. 6LoWPAN Network Architecture

The uplink to the Internet is handled by Access Point (AP) acting as an IPv6 router. Several different devices are connected to the AP in a typical setup, such as PCs, servers, etc. The 6LoWPAN network is connected to the IPv6 network using what is known as an edge router [6].

The edge router has mainly three functions as stated below:

- Data exchange between 6LowPAN and the Internet (or other IPv6 networks)
- Local data exchanges between the devices within the 6LowPAN
- The generation and maintenance of the radio sub-net (the 6LowPAN network)

It is important to note that the Edge routers forward data at the network layer. Hence they do not maintain any application layer state. This lowers the burden on the Edge Router in terms of processing power. Because of this property of the edge routers, it is possible to use embedded devices that are less expensive, run simpler software and have less complex hardware.

These 6LowPAN networks can be connected to other networks simply by the use of IPv6 routers. Any data which enters the 6LowPAN network through the edge router must be destined to one of its nodes. One 6LoWPAN network can be easily connected to other IP networks through one or more edge routers that forward IP datagrams between different media. Connectivity to the other IP networks may be provided by any type of link viz Ethernet, WiFi, 3G/4G etc.

There are mainly two types of devices used in a 6LoWPAN network: hosts and routers [6]. The main function of the router, as the name itself implies, is to route the data destined to some node in the 6LoWPAN. Host is nothing but an end device. It cannot route data in the network. In a constrained network, a host can be a sleepy device, i.e., a device which wakes up periodically to check its parent (i.e. a router) for data and then again goes back to sleeping mode in order to minimise the power consumption in the network.

B. Stack Overview

The application protocols used with 6LoWPAN could be Constrained Application Protocol (CoAP) or Message Queue Telemetry Transport (MQTT), etc. The security services like DTLS are employed in the 2nd layer from the top. RPL may be employed in the third layer.

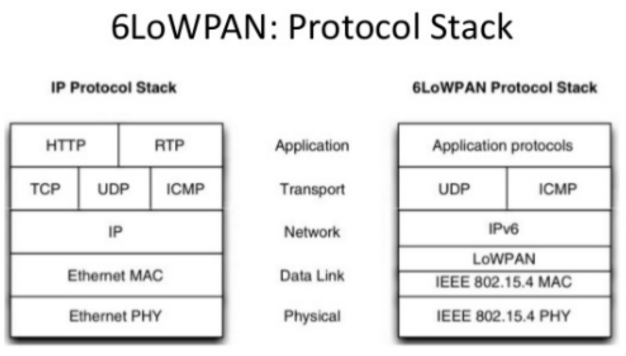

Fig. 1. Protocol Stack

6LoWPAN uses IPv6 rather than IPv4. IPv6 increases the minimum Maximum Transmission Unit (MTU) from 576 to

1280 bytes. IPv6 also reflects changes and advances in link layer technologies the Internet uses. However, devices used to implement 6LoWPAN are constrained in terms of resources, having about 16 kB RAM and 128 kB ROM. IPv6 packets over low-power and lossy networks. Due to the above resource constraints and 6LoWPAN multi-hop topology, supporting IPv6 over IEEE 802.15.4 networks present several challenges:

- IPv6 datagrams are not a natural fit for IEEE 802.15.4 networks. Low throughput, limited buffering and datagrams that are one-tenth of IPv6 minimum MTU make header compression [4] and data fragmentation a necessity. For example, IEEE 802.15.4 link headers can limit the effective possible payload to 81 bytes. This makes IPv6 (40 bytes), TCP (20 bytes) and UDP (8 bytes) headers seem way too large.
- Since IEEE 802.15.4 is both low power and low throughput, in addition to the use of RF as media, it is more prone to spurious interference, link failures and asymmetric links. Those characteristics require the network layer to be adaptive and responsive at the same time as low power and efficient.
- The most common network topology for 6LoWPAN is a low-power mesh network. This negates the assumption that a link is a single broadcast domain, something that is very important since the very foundation of IPv6 such as neighbor discovery relies on it.

These issues have to be accounted for by the 6LoWPAN standard.

C. 6LoWPAN Adaptation Layer

When sending data over MAC and PHY layers, an adaptation layer is always used. The main focus of the Internet Engineering Task Force (IETF) working group, 6LoWPAN WG, was to optimize the transmission of IPv6 packets over Low-power and Lossy Networks (LLNs) such as IEEE 802.15.4 and led to the publication of RFC 6282 specifying:

- HEADER COMPRESSION: Header Compression [4] compresses the 40-byte IPv6 and 8-byte UDP headers by assuming the usage of common fields. Header fields are elided when they can be derived from the link layer. The way the headers can be compressed is one main reason which led to the standard only supporting IPv6 rather than IPv4. However, there is nothing stopping one from running TCP in a 6LoWPAN system. The only reason why UDP is preferred over TCP is that TCP has more overhead and takes more processing time which has to be avoided in constrained environments like 6LoWPANs.

- FRAGMENTATION AND REASSEMBLY: The data link of IEEE 802.15.4 with a frame length of maximum 127 bytes does not match the MTU of IPv6, which is 1280 bytes. It should be noted that the frame format of IEEE 802.15.4g does not have the same limitation.
- STATELESS AUTO-CONFIGURATION: Stateless Auto-Configuration [4] is the process where devices inside the 6LoWPAN network automatically generate their own IPv6 address. There are methods to avoid the case where two devices get the same address; this is called Duplicate Address Detection (DAD).

Throughout the 6LoWPAN adaptation layer, the key concept is to use stateless or shared-context compression to elide header fields. This can compress all headers (adaptation, network and transport layers) down to a few bytes. It is possible to compress header fields because the header fields often carry common/similar values. Common values occur due to frequent use of a subset of IPv6 functionality, namely UDP, TCP and ICMP. Assumptions regarding shared context can also be made, such as a common network prefix for the whole 6LoWPAN system. The 6LoWPAN adaptation layer also removes duplicated information that can be derived from other layers, such as the IPv6 addresses and UDP/IPv6 length fields.

D. IPv6 Header Compression

The traditional way of performing IP header compression is status based, which is used at point-to-point connections where a flow between two end points is stable. This implementation is effective in static networks with stable links. Communication over multiple hops requires hop- by-hop compression and decompression. The routing protocols (e.g., RPL) [1] [3] normally running in 6LoWPAN systems obtain receiver diversity by rerouting, which would require state migration and hence severely reduce the compression efficiency. For dynamically changing networks, with multiple hops and infrequent transmissions like a 6LoWPAN radio network, another method has to be applied.

Instead in 6LoWPAN stateless and shared-context compression is used, which does not require any state and lets routing protocols dynamically choose routes without affecting compression ratio.

In Fig. 2, three communication scenarios are displayed:

- Communication between two devices inside the same 6LoWPAN network, using link-local addresses, the IPv6 header can be compressed to only 2 bytes.
- Communication destined to a device outside of the 6LoWPAN network and the prefix for the external network is known, where the IPv6 header can be compressed to 12 bytes.
- Similar to (b), but without knowing the prefix of the external device, that gives an IPv6 header of 20 bytes.

The best case is case (a), but this is rarely useful in sending application data, this case can be used only to send data to direct neighbours. The worst case here is case (c), which still provides us with a good 50 percent compression ratio. In the image, it has been assumed that the Interface ID (IID) for the

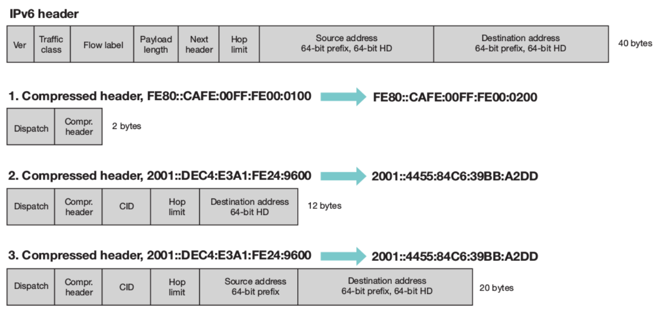
IPv6 header
Ver
Traffic class
Flow label
Payload length
Next header
Hop limit
Source address 64-bit prefix, 64-bit HD
Destination address 64-bit prefix, 64-bit HD
40 bytes
1. Compressed header, FE80::CAFE:00FF:FE00:0100
FE80::CAFE:00FF:FE00:0200
Dispatch
Compr. header
2 bytes
2. Compressed header, 2001::DEC4:E3A1:FE24:9600
2001::4455:84C6:39BB:A2DD
Dispatch
Compr. header
CID
Hop limit
Destination address 64-bit HD
12 bytes
3. Compressed header, 2001::DEC4:E3A1:FE24:9600
2001::4455:84C6:39BB:A2DD
Dispatch
Compr. header
CID
Hop limit
Source address 64-bit prefix
Destination address 64-bit prefix, 64-bit HD
20 bytes

Fig. 2. Header Compression

IPv6 header has been derived from the MAC address of the device.

E. *Fragmentation and Reassembly*

In order to successfully transfer data over the IEEE 802.15.4 radio links of the 6LoWPAN networks, it is essential to divide the data into several smaller segments. If the data has to be divided in this way, then it becomes necessary to include additional information in the header which stores the sequence number of each segment so that they can be reorganised in the right order at the receiver. When the data segments are regrouped into their complete data, this additional information is extracted and the packets are restored to their initial IPv6 format. The fragmentation [4] [5] [6] sequence of the segments is based on the type of routing protocol used by the network.

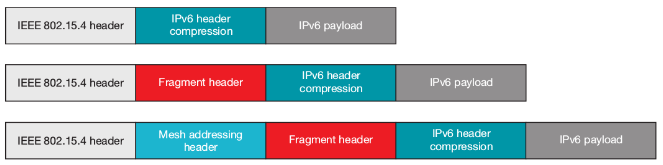

Fig. 3. Fragmentation and Reassembly

In the case of mesh-under routing (routing at the link layer), fragments will be reassembled and restored to their initial data only at their final destination. In this case, most of the network traffic is generated quickly as all fragments are passed immediately. If any fragments are missing at the receiver, the entire fragment must be re-transmitted. In a mesh-over routing protocol, fragmentation must be avoided as much as possible since it has a negative impact on the battery life of the device.

In the case of route-over networks (routing over network layer), the reassembly of the data segments occurs at every hop. This means that in a route-over network, each hop must have enough resources to store all fragments.

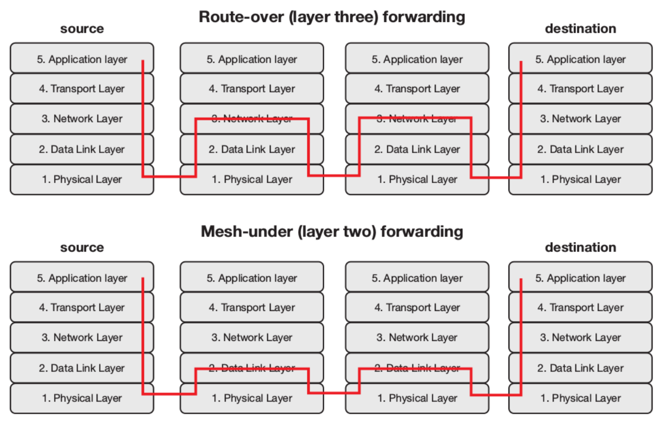

Fig. 4. Route-over vs Mesh-under forwarding

F. *Stateless Auto-configuration*

Auto configuration is the autonomous generation of a devices IPv6 address. The process is essentially different between IPv4 and IPv6. In IPv6 it allows a device to automatically generate its IPv6 address without any outside interaction with a DHCP server or such. To get an address, a host can communicate via Neighbor Discovery Protocol (NDP), however many of the NDP [4] features are also included in RPL. The procedure described here is valid for RPL also, and involves four message types:

- Router Solicitation (RS)
- Router Advertisement (RA)
- Neighbor Solicitation (NS)
- Neighbor Advertisement (NA)

IPv6 Neighbor Discovery (ND) enables a device to discover its neighbors, maintain reachability information, configure default routes, and propagate configuration parameters.

The RS message mainly includes the IPv6 prefix of the network. All routers in the network periodically send out the RS messages. If a host wants to participate in a 6LoWPAN network, it assigns itself a link-local unicast address (FE80::IID), then sends this address in an NS message to all other participants in the sub-net to check if the address is being used by someone else. If it does not receive in return, an NA message, within a defined time frame, it assumes that the address is unique. This procedure is called Duplicate Address Detection (DAD). Now, to get the network prefix, the host sends out an RS message to the router to get the correct prefix. Using these four messages, a host is able to assign itself a worldwide unique IPv6 address.

Using source address auto configuration, each host generates a link-local IPv6 address using its IEEE 802.15.4 EUI-64 address, 16-bit short address or both. In a mesh-under configuration, the link- local scope covers the entire 6LoWPAN network, even over multiple hops, and a link-local address is sufficient for communication happening in the

6LoWPAN. The only time a routable IPv6 address is needed is when communicating outside of the 6LoWPAN network. In a route-over configuration, a link-local address is sufficient to communicate with nodes that are within radio coverage, but a routable address is required to communicate with devices several hops away.

IV. IP ADDRESSES

The Internet was earlier purely based on IPv4 and used 32-bit addresses, which limits the address space to 4,294,967,296 unique addresses. As addresses were assigned to users (and devices), the number of unassigned addresses naturally decreased. Despite the use of network changes such as Network Address Translation (NAT), although delayed significantly, inevitably the IPv4 address exhaustion occurred on Feb. 3, 2011.

In order to smoothly deal with this major and inevitable limitation of IPv4, the development of IPv6 commenced in the 1990s, and it has been in commercial deployment since 2006. IPv6 covers an address space of 2Î28 and 3.4*103̂8 unique addresses. This has been assumed (calculated) to be enough for Internet to scale for decades to come, even with the promise of the Internet of Things, which according to estimates might include 50 billion connected devices by the year 2020.

IPv6 increases the minimum Maximum Transmission Unit (MTU) from 576 to 1280 bytes. IPv6 also reflects changes and advances in link layer technologies the Internet uses.

A. IPv6 Link Local Address

The IPv6 link-local address is formed by appending the Interface Identifier (IID) to the prefix FE80::/64. ie., bits 0-9 are 1111111010, bits 10-63 are all zeroes, bits 64-127 are the bits of the IID.

B. Unicast Address Mapping

The addresses in IPv6 may be either EUI-64 or 16-bit short addresses.

Fig. 5 and Fig. 6 show the two type of addresses. For the option fields in the images:

- Type: 1 for source link-layer address ; 2 for target link-layer address
- Length This is the length of the option in units of Bytes. For EUI-64, length is 2 and for 16-bit short address, it is 1.

C. Multicast Address Mapping

A multicast address has its first three bits as 100.

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length=1   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     16-bit short Address      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-          Padding            -+
|         (all zeros)           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 5. 16-bit short address

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |    Length=2   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-        IEEE 802.15.4        -+
|          EUI-64               |
+-                             -+
|                               |
+-          Address            -+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                               |
+-          Padding            -+
|                               |
+-        (all zeros)          -+
|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 6. 64-bit EUI

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 0 0|DST[15]* |   DST[16]     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 7. Multicast format

D. Header Compression

Header compression may result in alignment not falling on an octet boundary. Since hardware typically cannot transmit data in units less than an octet (byte), padding must be used. Padding is done as follows:

- The entire series of contiguous compressed headers is laid out.
- Zero bits should be added as appropriate to align to an octet boundary. This mechanism counteracts any potential misalignment caused by header compression, so subsequent fields (e.g., non-compressed headers or data payloads) start on an octet boundary and follow as usual.

V. Contiki Operating System

Typical sensor devices are equipped with 8-bit micro-controllers, code memory of the order of 100 kB, and less than 20 kB of RAM. Moores law predicts that these devices can be made signicantly smaller and less expensive in the future. While this means that sensor networks can be deployed to greater extents, it does not necessarily imply that the resources will be less constrained. Wireless sensor networks are composed of large numbers of tiny networked devices (motes and sensors) that communicate with each other. For large scale networks it is important to be able to dynamically download code into the network. This can be done with the use of the Operating System known as Contiki [7] (developed by Adam Dunkels), which is a lightweight operating system with support for dynamic loading and replacement of individual programs and services. Contiki is built around an event-driven kernel but provides optional pre-emptive multithreading that can be applied to individual processes. Dynamic loading and unloading is feasible in a resource constrained environment, while keeping the base system lightweight and compact. This Operating System is written in the C language.

Since our smart campus water management project uses wireless network with sensors to detect the water level in the tanks and a combination of RE-motes and wismotes for transmitting the data, we have resolved to the use of Contiki Operating System.

A. Simulations

The code written on the Contiki OS can be simulated using the tool known as Cooja [7]. Cooja Simulator can be used by entering the directory ”cooja” on the terminal and typing the command ”ant run”.

B. Contributions of Contiki

Contiki has mainly two contributions to the world of sensor networking:

- The first contribution is the feasibility of loadable programs and services even in a constrained sensor device. The possibility to dynamically load individual programs leads to a very exible architecture, which still is compact enough for resource constrained sensor nodes.
- The second contribution is more general in that the pre-emptive multi-threading does not have to be implemented at the lowest level of the kernel but that it can be built as an application library on top of an event-driven kernel. This allows for thread-based programs running on top of an event-based kernel, without the overhead of re-entrancy or multiple stacks in all parts of the system.

C. Downloading code at run-time

In practice, the wireless sensor networks usually large scale, with hundreds or even thousands of nodes per network. When developing software for such large sensor networks, we must keep in mind that we should be able to dynamically download program code into the network. It is not feasible to physically collect and reprogram all sensor devices and hence in-built mechanisms are required for this purpose.

D. System Overview

A running Contiki system consists of the kernel, libraries, the program loader, and a set of processes. A process may be either an application program or a service. A service implements functionality used by more than one application process. All processes, both application programs and services, can be dynamically replaced at run-time. Communication between processes always goes through the kernel. The kernel does not provide a hardware abstraction layer, but lets device drivers and applications communicate directly with the hardware. A process is defined by an event handler function and an optional poll handler function. The process state is held in the process private memory and the kernel only keeps a pointer to the process state. On the ESB platform, the process state consists of 23 bytes. All processes share the same address space and do not run in different protection domains. Inter- process communication is done by posting events.

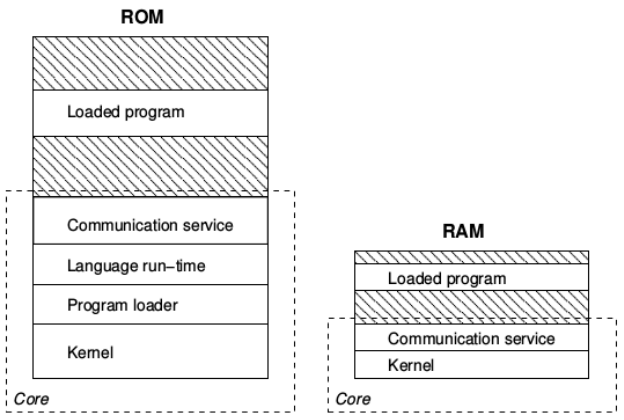

Fig. 8. System Overview

E. KERNEL ARCHITECTURE

The Contiki kernel consists of a lightweight event scheduler that dispatches events to running processes and periodically calls processes polling handlers. All program execution is triggered either by events dispatched by the kernel or through the polling mechanism. The kernel does not pre-empt an event handler once it has been scheduled. Therefore, event handlers must run to completion.

Two-Level Hierarchy

All event scheduling in Contiki is done at a single level and events cannot pre-empt each other. Events can only be pre-empted by interrupts. Normally, interrupts are implemented using hardware interrupts but may also be implemented using an underlying real-time executive. The latter technique has previously been used to provide real-time guarantees for the Linux kernel. In order to be able to support an underlying real-time executive, Contiki never disables interrupts. Because of this, Contiki does not allow events to be posted by interrupt handlers as that would lead to race-conditions in the event handler. Instead, the kernel provides a polling flag that it used to request a poll event. The flag provides interrupt handlers with a way to request immediate polling.

Loadable Programs

Loadable programs are implemented using a run-time relocation function and a binary format that contains relocation information. When a program is loaded into the system, the loader first tries to allocate sufficient memory space based on information provided by the binary. If memory allocation fails, program loading is aborted. After the program is loaded into memory, the loader calls the programs initialization function. The initialization function may start or replace one or more processes.

Power Save Mode

In sensor networks, being able to power down the node when the network is inactive is an often required way to reduce energy consumption. Power conservation mechanisms depend on both the applications and the network protocols. The Contiki kernel contains no explicit power save abstractions, but lets the the application specific parts of the system implement such mechanisms. To help the application decide when to power down the system, the event scheduler exposes the size of the event queue. This information can be used to power down the processor when there are no events scheduled. When the processor wakes up in response to an interrupt, the poll handlers are run to handle the external event.

F. SERVICES

In Contiki, a service is a process that implements functionality that can be used by other processes. A service can be seen as a form of a shared library. Services can be dynamically replaced at run-time and must therefore be dynamically linked. Typical examples of services includes communication protocol stacks , sensor device drivers and higher level functionality such as sensor data handling algorithms. Services are managed by a service layer conceptually situated directly next to the kernel. The service layer keeps track of running services and provides a way to find installed services. A service is identified by a textual string that describes the service. The service layer uses ordinary string matching to querying installed services. A service consists of a service interface and a process that implements the interface. The service interface consists of a version number and a function table with pointers to the functions that implement the interface.

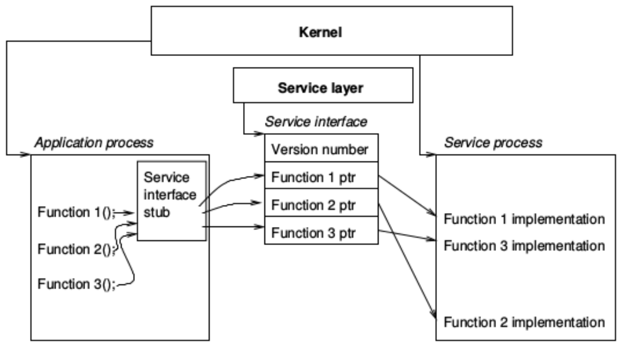

Fig. 9. Services

G. LIBRARIES

The Contiki kernel only provides the most basic CPU multiplexing and event handling features. The rest of the system is implemented as system libraries that are optionally linked with programs. Programs can be linked with libraries in three different ways. First, programs can be statically linked with libraries that are part of the core. Second, programs can be statically linked with libraries that are part of the loadable program. Third, programs can call services implementing a specific library. Libraries that are implemented as services can be dynamically replaced at run-time. Typically, run-time libraries such as frequently-used parts of the language run-time libraries are best placed in the Contiki core. Rarely used or application specific libraries, however, are more appropriately linked with loadable programs.

Libraries that are part of the core are always present in the system and do not have to be included in loadable program binaries.

H. COMMUNICATION SUPPORT

Communication is a fundamental concept in sensor networks. In Contiki, communication is implemented as a service in order to enable run-time replacement. Implementing communication as a service also provides for multiple communication stacks to be loaded simultaneously. In experimental research, this can be used to evaluate and compare different communication protocols. Furthermore, the communication stack may be split into different services as shown in the Fig. 10. This enables run-time replacement of individual parts of the communication stack.

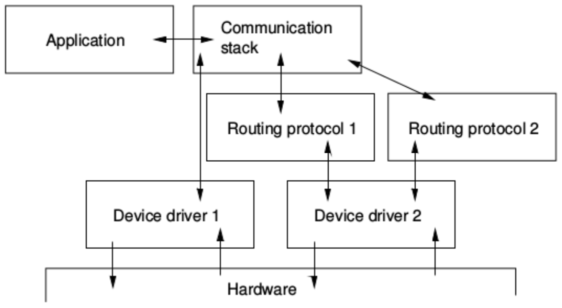

Fig. 10. Communication support

I. PRE-EMPTIVE MULTITHREADING

In Contiki, pre-emptive multi-threading is implemented as a library on top of the event-based kernel. The library is optionally linked with applications that explicitly require a multi-threaded model of operation. The library is divided into two parts: a platform independent part that interfaces to the event kernel, and a platform specific part implementing the stack switching and pre-emption primitives. Usually, the pre-emption is implemented using a timer interrupt that saves the processor registers onto the stack and switches back to the kernel stack. In practice very little code needs to be rewritten when porting the platform specific part of the library.

Unlike normal Contiki processes each thread requires a separate stack. The library provides the necessary stack management functions. Threads execute on their own stack until they either explicitly yield or are pre-empted.

Code Size

An operating system for constrained devices must be compact in terms of both code size and RAM usage in order to leave room for applications running on top of the system. The replicator service consists of the service interface stub for the service as well as the implementation of the service itself. The program loader is currently only implemented on the MSP430 [7] platform.

Pre-emption

The purpose of pre-emption is to facilitate long running computations while being able to react on incoming events such as sensor input or incoming communication packets. The figure below shows how Contiki responds to incoming packets during an 8 second computation running in a pre-emptible thread. The curve is the measured round-trip time of 200 ping packets of 40 bytes each. The computation starts after approximately 5 seconds and runs until 13 seconds have passed. During the computation, the round-trip time increases slightly but the system is still able to produce replies to the ping packets.

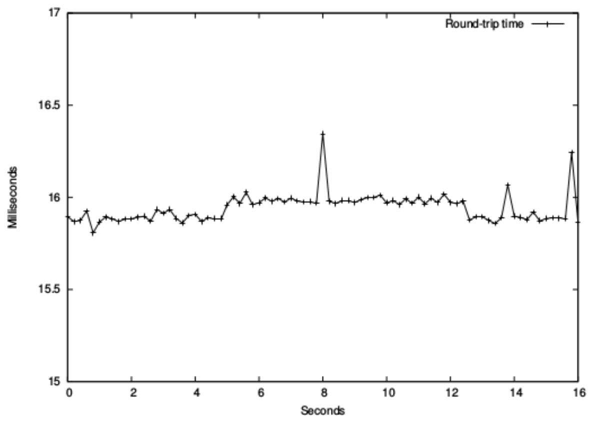

Fig. 11. Preemption graph

J. PORTABILITY

Contiki has been ported to a number of architectures, including the Texas Instruments MSP430 and the Atmel AVR, the Hitachi SH3 and the Zilog Z80. The porting process consists of writing the boot up code, device drivers, the architecture specific parts of the program loader, and the stack switching code of the multi- threading library. The kernel and the service layer does not require any changes. Since the kernel and service layer does not require any changes, an operational port can be tested after the first I/O device driver has been written.

VI. LinkPeek

LinkPeek [8] is an enhancement of the Contiki RPL with the focus on minimizing the packet drops during data transfers within the network. With the Contiki RPL, if a parent node is unavailable or unreachable due to the obstacles and other circumstances, then the packet will be dropped until the repair (global or local repair) process is complete and a new parent has been selected by the child node. LinkPeek ensures higher Packet Delivery Ratio (PDR) and low latency. Packet Replication and multipath forwarding would not be a wise solution to reduce packet drops because it gives rise to additional traffic, which implies a higher energy consumption. It may also result in unpredictable interaction with other existing nodes.

LinkPeek makes the effort to reduce these packet drops by storing the list of potential parents (in decreasing order of their path metrics) to each node right from the beginning. It does not interfere with the working of RPL until a parent is unavailable during the data transfer. When the RPL default parent becomes unavailable, instead of dropping the packet, LinkPeek makes a copy of the packet and tries re-routing that packet through the alternate parents. If none of the alternate parents are available, then there will be packet drop. In this way, the number of packets lost during an obstacle will be greatly reduced with the use of LinkPeek.

However, one drawback of LinkPeek is that alternate parent selection incurs additional delay. But, this compromise is acceptable in return for the improvement in the PDR that is attained using LinkPeek.

Conclusion

Wireless networks are usually LLNs and are constrained in terms of the resources available at the nodes. RPL is widely used for wireless sensor networks. It forms a virtual acyclic graph from the nodes present in the network and maintains a neighbour table at each node (in storing mode) or only at the border router (in non-storing mode). 6LoWPAN networks are ideal for use-cases such as home automation, street light monitoring, etc. It is ideal for a small scale wireless communication with few network motes. These motes are usually constrained in terms of their resources like power consumption, memory, etc. Therefore, the operating system used to program these motes has to be lightweight. Contiki operating system is the ideal operating system for this purpose. With the incorporation of LinkPeek, the performance of RPL is greatly enhanced, as the packets will be buffered and sent by an alternate route in cases when the default route fails. The Smart Campus Water Management has been implemented using these concepts. It uses RPL for routing the data packets within the network. The network is a 6LoWPAN with the RPL-border-router acting as the edge router.

Acknowledgment

Dr. Malati Hegde, Professor, Electrical Communications Department IISc and S.V.R. Anand have constantly provided their guidance and support throughout this project.

References

[1] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, JP. Vasseur, R. Alexander, *"RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks"* RFC:6550, March 2012.

[2] J. Hui, JP. Vasseur *"The Routing Protocol for Low-Power and Lossy Networks (RPL) Option for Carrying RPL Information in Data-Plane Datagrams"* RFC:6553, March 2012.

[3] JP Vasseur, Navneet Agarwal, Jonathan Hui, Zach Shelby, Paul Bertrand, Cedric Chauvenet, Watteco SAS, *" RPL: The IP routing protocol designed for low power and lossy networks"*, April 2011.

[4] G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, *"Transmission of IPv6 Packets over IEEE 802.15.4 Networks"* RFC:4944, September 2007.

[5] J. Hui, P. Thubert, *"Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks"* RFC:6282, September 2011.

[6] Jonas Olsson, Texas Instruments, *"6LoWPAN Demystified"*, October 2014.

[7] Adam Dunkels, Bjorn Gronvall, Thiemo Voigt *"Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors"*.

[8] Lohith YS, Sathya Narasimman T, S.V.R. Anand, Malati Hedge, Electrical and Communication Engineering, Indian Institute of Science, *"LinkPeek: A Link Outage Resilient IP Packet Forwarding Mechanism for 6LoWPAN/RPL based Low-Power and Lossy Networks (LLNs)"*, 2015 IEEE.