Network Management Entity Implementation of Centralized Performance Management

Prepared by Samanvita Sastry

***Abstract*—This report explains about how Constrained Application Protocol (CoAP) can be integrated with the Path Computation Element (PCE) using Network Management Entity (NME) in order to periodically query each node in the network and obtain the details such as battery value, number of frames dropped and frames sent (totally as well as on a per-parent basis). Link cost with parent and IP address of preferred parent are also details that greatly aid in the working of the PCE. It also gives a brief explanation on the working of PCE and also explains how to integrate new resources with the Contiki code, in order to observe these resources through CoAP. The resources can be queried either from the browser or using NME. The working of NME is explained in detail in this report. This report covers the basics of IPv6 Routing Protocol for LLNs (RPL), Contiki's Cooja simulator and briefly explains how to add resources to be queried using CoAP.**

***Keywords*—Network Management Entity, Path Computation Element, Resources, Constrained Application Protocol**

I. Introduction

In recent times, the use of sensors has increased manifold. Sensors like magnetic or radio sensors, humidity sensor, fluid velocity sensors, pressure sensors, temperature sensors, etc are used in a lot of applications in our day-to-day lives. These are usually employed with wireless networks. These networks are generally constrained in terms of their resources. They are known as Low Power and Lossy Networks (LLNs) [5]. At the application layer of these networks, implementing protocols such as HTTP may not be very feasible because HTTP is based on TCP protocol using point-to-point (P2P) communication model that is not suitable for notification push services. HTTP has high computation complexity, low data rate and high energy consumption. Hence, we switch to using light-weight protocols such as Constrained Application Protocol (CoAP).

CoAP can be used with the Copper plug-in for Firefox browser (on the mote side) and californium (on server side). In this report, we will be focusing on the mote side of the network. Also, instead of manually querying each node in the network, we can automate the querying process such that each time we send a query, we get the information back from the node in bulk. Each node in the network will be queried periodically. In order to program the motes, we will use the Contiki [4] operating system. With this operating system, dynamic loading and unloading is feasible in a resource constrained environment, while keeping the base system lightweight and compact. In our project, we have used Contiki 3.x version. We use the IPv6 Routing Protocol for LLNs (RPL) [5] [6] [3] with constrained networks, and this routing protocol is efficient for small networks of up to 50 nodes.

In Section II, the background required to understand the project; about the Network Management Entity, Path Computation Element, CoAP and RPL is given. Section III briefly explains about how to use Contiki's Cooja simulator to simulate a network before testing it on real motes in the real world. The following section explains how to add resources and integrate them with the server code and view them on the CoAP browser. We will then see how to automate the process of querying the nodes using coapthon [7].

II. Background

In order to clearly understand this project, it is necessary to know the basic concepts about Network Management Entity, Path Computation Element, CoAP and RPL.

A. Network Management Entity

The Network Management Entity (NME) closely observes the changes in the RPL network. NME incorporates within it, the Path Computation Element (PCE), the latest version of the network topology, whenever the need arises. NME keeps track of the global network which contains more than one border router. Thus it keeps track of multiple DODAGs (DODAG instances) [5] and can give the PCE a clear picture of the global network. NME uses the application layer protocol, Constrained Application Protocol (CoAP) to connect to the network [2]. CoAP is used so as to meet the QoS demands of LLNs [5].

The function of the NME is to periodically request each of the nodes, the resources that are needed by the PCE. NME will also provide PCE, the DODAG representation of the network. In order to provide the latest version of each DODAG, the NME can query the border router to see if the DODAG version [5] has changed or not. The periodic interval in which the NME queries each nodes for the representation of the network, can be optimized according to individual application circumstances.

B. Path Computation Element

RPL used to route the LLN networks attains a Packet Delivery Ratio (PDR) in the range of 98-100%. The Path Computation Element (PCE) focuses on enhancing the RPL protocol in such a way that the PDR increases further. The

PDR values reach high values because RPL constructs the DODAG strictly based on the Objective Function (OF). RPL has been explained in Section II D. By computing the path metric based on the OF and also checking the constraints simultaneously, the DODAG is formed such that each node is assigned a rank. Based on the rank, each node selects its RPL default parent from among its neighbours. Thus, according to the OF used, RPL maintains a designated route such that each node in the network is assigned a preferred parent that will be used to forward packets in the upward direction. Considering the possibility that few of the packets that are dropped due to congestion at a certain node, load-balancing can be used to reduce the packet drops. In order to overcome this problem, the idea of PCE was initiated.

Generally, after the configured number of retransmissions, RPL identifies this inefficiency and reorganizes the entire DODAG by a mechanism known as global repair, to setup optimal routes for each node. PCE avoids this reorganisation by diffusing the congestion that has occurred due to the factors, that are not included in the calculation of the rank. The system proposed contains three units.

- LLN connected to an RPL Border Router.
- Network Management Entity
- Path Computation Element

The overall diagram of the system is shown in Fig. 1.

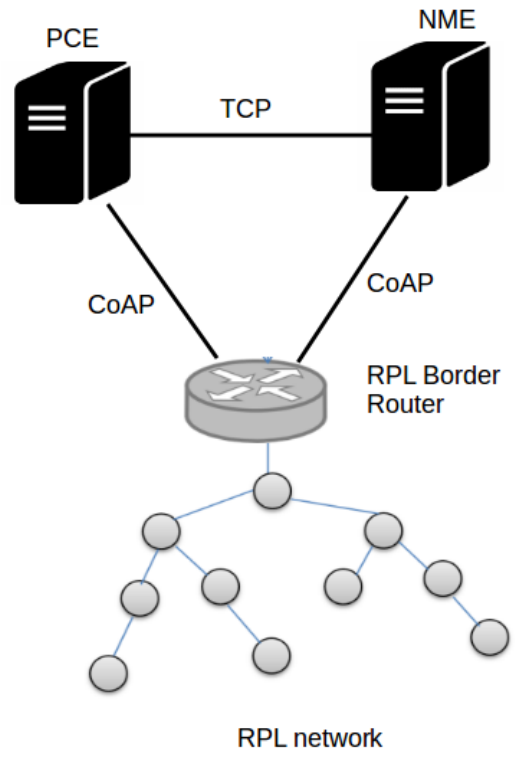

Fig. 1. The Big Picture demonstrating the working of PCE

C. Constrained Application Protocol

IoT needs to integrate various sensors, computer and communication equipment, which are using different communication protocols. Wireless protocols are mainly used in four layers, which are PHY layer, MAC layer, Network/Communication layer and Application layer. Application layer usually employ HTTP to provide web service, but HTTP has high computation complexity, low data rate and high energy consumption. Therefore, IETF has developed several lightweight protocols.

CoAP [1] is one of the latest application layer protocol developed by IETF for smart devices to connect to the Internet. As many devices exist as components in vehicles and buildings with constrained resources, it leads to a lot of variation in power computing, communication bandwidth etc. Thus lightweight protocol CoAP is to be used and considered as a replacement to HTTP for being an IoT application layer protocol. CoAP would become the standard protocol to enable interaction between devices and to support IoT applications. The Constrained RESTful Environments (CoRE) is the workgroup in IETF that is designing the CoAP protocol. CoAP needs to consider optimizing length of datagram and satisfying REST protocol to support Uniform Resource Identifier (URI). It also needs to provide dependable communication based on UDP protocol. The features of CoAP are shown in Fig. 2.

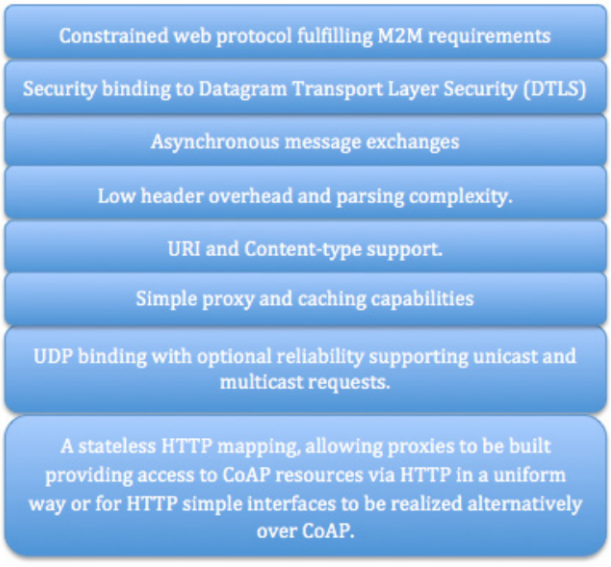

Fig. 2. CoAP features

COMPARISON BETWEEN CoAP AND HTTP :
CoAP is a network-oriented protocol, using similar features to HTTP but also allows for low overhead, multicast, etc. As HTTP protocol is a long-term successful standard, it can use small script to integrate various resources and services. Inter-operation provided by HTTP is the key point of IoT, for

this, HTTP is employed in application level. HTTP is based on TCP protocol using point-to-point (P2P) communication model that is not suitable for notification push services. Also, for constrained devices, HTTP is too complex. Unlike HTTP based protocols, CoAP operates over UDP instead of using complex congestion control as in TCP. CoAP is based on REST architecture, which is a general design for accessing Internet resources. In order to overcome disadvantage in constrained resource, CoAP need to optimize the length of datagram and provide reliable communication. On one side, CoAP provides URI, REST method such as GET, POST, PUT and DELETE [2]. On the other side, based on lightweight UDP protocol, CoAP allows IP multicast, which satisfies group communication for IoT. To compensate for the unreliability of UDP protocol, CoAP defines a retransmission mechanism and provides resource discovery mechanism with resource description. Fig. 3 shows the HTTP and CoAP protocol stacks.

Fig. 3. HTTP stack vs CoAP stack

D. RPL

IPv6 Routing Protocol for LLNs (RPL) is a routing protocol that is used for LLNs [5]. Traditional routing protocols are designed to minimize the convergence time due to the voice and video traffic requirements. 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 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 for resolving routing inconsistencies.

RPL basically 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) [3] [5]. RPL also includes methods for detecting and avoiding loop formations in the network. 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. The result of the DODAG formation, is that each node is assigned a rank, starting with rank 0 for the border router. The graph formed depends upon a function known as the OF which is composed of the metrics and constraints for DODAG formation. 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 (LLN). The Objective Code Point (OCP) is an identifier that indicates which OF the DODAG uses.

RPL uses a timer known as trickle timer instead of using periodic keep-alives (keep-alive is a message sent by one device to another to check that the link between the two nodes is functional, or to avoid the breaking of this link). The trickle timer value increases when the network becomes more stable, while it decreases when the network becomes inconsistent due to various reasons like detection of loop, addition or removal of nodes from the network. When loops are detected, RPL triggers either the local or global repair. In local repair, only an alternate parent will be chosen for one of the nodes. In global repair, the entire DODAG will be re-formed.

III. Using Cooja Simulator

The Contiki operating system provides an in-built network simulator known as the Cooja simulator. With the help of this simulator, it is possible to test the working of a network without physically burning the code on to the motes and testing them in the real world. We can first form the network with the help of Cooja and evaluate the working of the motes. If the network does not function as required, the code can be modified and the simulation can be refreshed. When the network is finally functioning as desired, we can burn the code onto the motes and deploy them in the real world. This saves a lot of time which may have otherwise been wasted by burning the code every time and testing it physically in order to debug the code. In this section we will see how to use the Cooja simulator.

Open the terminal and open the path tools/cooja inside the Contiki folder. Type the command ant run on the terminal. The Cooja simulator will open. Now click on file menu, and then on ”create new simulation”. Name your simulation and click

on ”Create”. Now, click on ”Motes”,and then create new mote. Select the type of mote you want to create, and then browse the code you want to burn onto the mote. Compile the code. Errors and warnings will be displayed in the log. If there are errors, go back to the original code and debug the errors. If the code compiles successfully, then click on ”Create”. You will be able to see the mote on the screen of the simulator. Fig. 4 is an example of a network created with one RPL border router and 5 nodes.

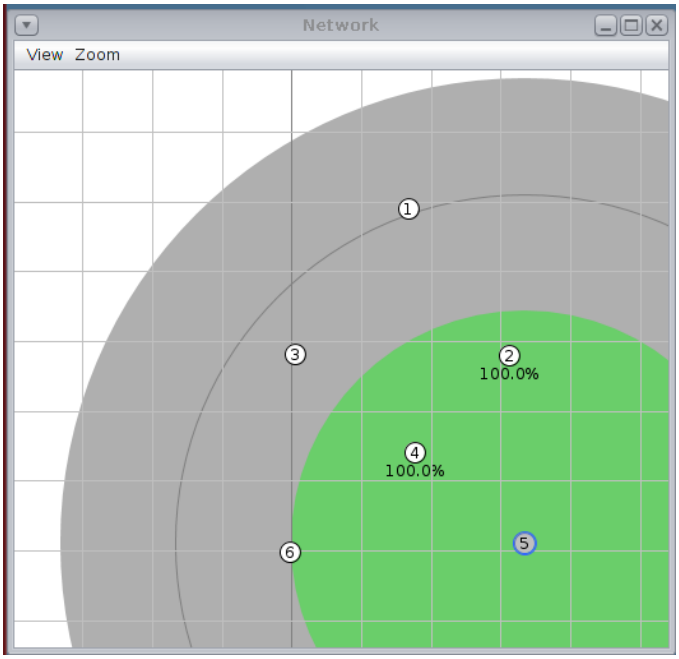

Fig. 4. A sample network of 6 nodes on Cooja simulator

IV. Adding Resources to CoAP

In order to add a resource, create a new resource C-file in the resources sub-directory of the path examples/er-rest-example/ folder within Contiki. Code the c-file and define the resource within it. Now activate this resource using the function rest_activate_resource(&resource_name, path in CoAP). Now open your Internet browser and type the address coap://[mote_address]:port_number. Once your network has been set-up and your border router is up and running, click on the ping button on the tab. If the server mote is detected, there will be a message saying ”PONG: Remote responds to CoAP”. PONG is a reply to the ping from the server and it implies that the node is available for querying. Now click on ”Discover”, the resources that have been activated in the server code will be visible on the left most column of the browser window. You can click on the desired resource and then either GET, POST, PUT, OBSERVE or DELETE [2] in accordance with the definition of the resource in your code and what event will respond to the query. Fig. 2 depicts the CoAP browser window for the active network shown in Fig. 1.

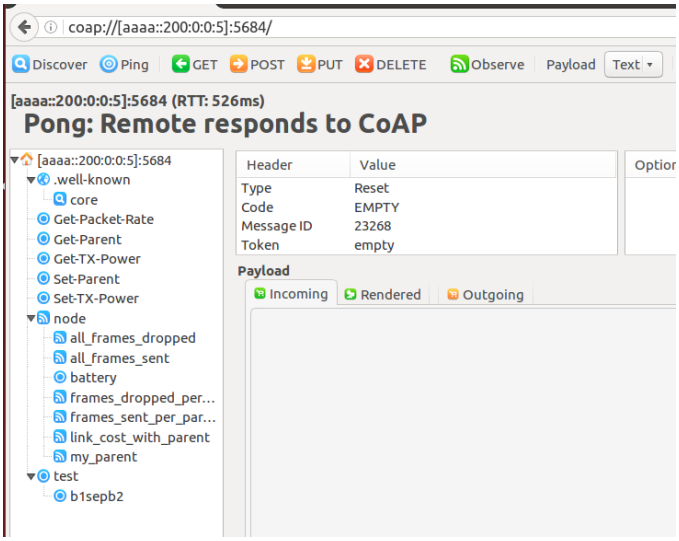

Fig. 5. Copper plug-in for Firefox Browser for the network set-up shown in Fig. 1

V. Coapthon

Coapthon is the implementation of CoAP using Python programming language [7]. With the use of python server, we can automate the process of querying the nodes. As we have seen, using the Copper plug-in for Firefox browser, we can select one resource at a time and query the node for that particular resource. With coapthon, we have an added advantage that we can query a number of resources of the node at once. We have programmed our server node to query each server node one after the other periodically in a loop. For each node, the resources battery, total frames sent, total frames dropped, link cost with parent (ETX), preferred-parent IP address, frames sent per parent, frames dropped per parent and the Packet Delivery Ratio (PDR) with respect to each parent are queried. Fig. 3 is a screen-shot demonstrating the working of the server code written using coapthon.

Fig. 6. Output of the server code run using coapthon

The Get-Parent function returns all the parents of the node 5. For our network as shown in Fig. 4, it returns the IP addresses of nodes 2 and 4. Now if we click on my_parent, it displays the

IP address of the preferred parent (IP address of node 2 in the example network). Now using the PCE that has already been coded, we can set a percentage for the number of packets that should be distributed between the parent nodes (for example; click on Set-parent, and type 30; followed by the IP address of node 4 and click on post). This will ensure that the packets that are sent from node 5 will get distributed among nodes 2 and 4 in the ratio 7:3.

If you want to find out the number of frames that have been sent to (or dropped while being sent to) either of the parents, then click on frames_sent_per_parent (or frames_dropped_per_parent), type in the IP address of the parent i.e. in our example either the IP address of node 2 or the IP address of node 4, and then click on the post button. The corresponding value of the frames sent (or frames dropped) will be displayed by CoAP. You can verify these values by using the all_frames_sent (or all_frames_dropped) functions. The sum of frames sent by each parent will add up to the total frames sent, and the sum of the frames dropped by each parent will add up to the total frames dropped.

CONCLUSION

RPL is a widely used routing protocol for LLNs and the constraints in the OF can be set based on the constraints and available resources in the network. Adding new resources and coding them in accordance with our need of the hour is very useful in aiding PCE. PCE benefits from this project as it can observe exactly where and how the packets are being sent and dropped, and can easily recognise the bottle-necks merely by observing the statistics. The alternate parent and the percentage of datagram packets that should be passed to it can thereby be calculated based on the metrics of the new resources that are added to CoAP. Copper plug-in can be used to view and observe the resources, or python code can be used to automate the same. Coapthon can be very useful as the developer can modify the server code as per their requirement which depends on the number of nodes in their multi-hop network, the resources that they need to observe and the format which they find most convenient to make their observations. The PCE tries to diffuse the congestion by acting as a load-balancer, thereby avoiding the DODAG reconstruction. This load balancing is done by finding an alternate path and diverting some amount of packets through that path. For finding the alternate path, it makes use of a dynamic programming methodology that finds a path optimized to certain constraints.

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] Xi Chen and Prof.Raj Jain, *"Constrained Application Protocol for Internet of Things"*.

[2] Z. Shelby, K. Hartke, C. Bormann, *"The Constrained Application Protocol (CoAP)"* RFC: 7252, June 2014.

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

[5] 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.

[6] 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.

[7] G. Tanganelli, C. Vallati, and E. Mingozzi, *"Coapthon: Easy development of coap-based iot applications with python"*, IEEE World Forum on Internet of Things, 2015.