KTH Electrical Engineering

CTP over IEEE 802.15.4

Multihop routing over a time scheduled network

MUHAMMAD ALTAMASH AHMED KHAN

Stockholm July 31, 2012

TRITA-EE 2XXX:YYY
Version: 1.0

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless sensors have recently been the focus of research activities in many areas like wireless process control, precision agriculture, supply chain management, health care etc. They have the ability to sense the physical world, process the gathered information and inform the other sensors about it. A network of sensors can be formed where they can communicate with each other and which can also be connected to the internet. An of application of the usage of WSN is in the Networked control systems (NCS). In these systems, the control loop (feed-forward, feed-back or both) can be comprised of a WSN. Figure **??** describes a typical NCS,

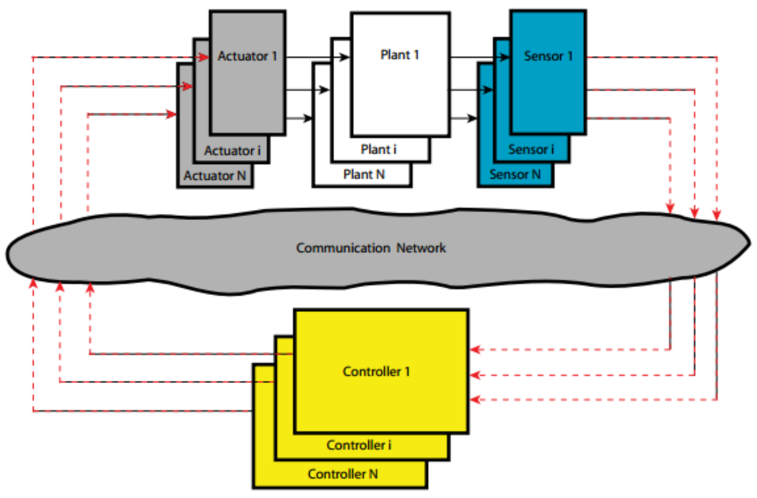


Figure 1.1: An illustartion of a network control system

The commuincation protocol stack for a WSN is slightly different from stacks for other networks. Its generally does not have all the layers of the OSI model. Low power consumption is also a key aspect of the protocol functionality. The performance of a NCS is affected by the delay and the packet loss rate in the WSN, especially when it contains multi-hops.

This manual describes the working of multi-hop routing over a time division multiple access (TDMA) based wireless sensor network.

1.1 MAC layer

All data communication is handled by the physical layer (PHY) of the protocol stack. The physical layer is responsible for frequency selection, carrier frequency generation, signal detection, modulation, and data encryption etc. The Medium Access Control (MAC) layer is responsible for the multiplexing of data streams, data frame detection, and medium access. It ensures reliable point-to-point and point-to-multipoint connections in a communication network. It manages the device to device data communication link, sharing the communication resources between sensor nodes [4]. Most MAC protocols fall either into the categories of contention-free or contention-based protocols. In the first category, MAC protocols provide a medium sharing approach that ensures that only one device accesses the wireless medium at any given time. Example of this is a time division multiple access (TDMA) or a frequency division multiple acces based scheme. In contrast to contention-free techniques, contention-based protocols allow nodes to access the medium simultaneously, but provide mechanisms to reduce the number of collisions and to recover from such collisions [3]. Its example include a carrier sense multiple access - collision avoidance (CSMA-CA) scheme. IEEE has specified the PHY and MAC protocol functionalities in a standard, referred as IEEE 802.15.4 [7]. This particular standard will be the focus of discussion in the report.

1.2 IEEE 802.15.4 MAC

IEEE 802.15.4 standard [7] specifies the PHY and MAC layer for low-rate WPANs . Some of the main characteristics of the IEEE 802.15.4 are:

- 250 kbps, 40 kbps and 20 kbps data rates.
- Two addressing modes, 16-bit short and 64-bit IEEE addressing.
- CSMA-CA channel access.
- Automatic network establishment by the coordinator of the network.
- Power management control.
- 16 channels in the 2.4 GHz ISM band, 10 channels in the 915 MHz ISM band and one channel in the 868 MHz band

IEEE 802.15.4 specified two types basic of MAC operation, Beacon Enabled and Non-Beacon Enabled. In Beacon Enabled, a node called `Coordinator` periodically broadcasts a short message called a Beacon. Any other node in the vicinity, called `Device`, that listens to the beacon, adjusts its timer to beacon period , or synchronize with the `Coordinator` node, and continues to track the future incoming beacons. A coordinator bounds its channel time using a super-frame structure. A Super-frame is bounded by the transmission of a beacon frame and can have an active portion and an inactive portion. The coordinator may enter a low-power (sleep) mode during the inactive portion. Devices adjust their transmission time with respect to the beacon. The active portion of the super-frame is divided in to two portions. A Contention Access Period (CAP) where the devices compete for a collision free transmission slot, and a Collision Free Period (CFP) where devices are allocated a time slot by the coordinator, such that there is only one device transmitting in a given time slot. `Coordinator` decides a `Time-slot Schedule` for the devices in the network.

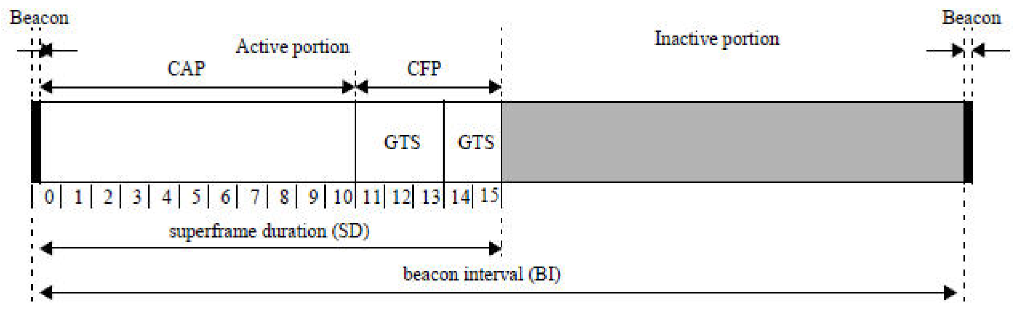


Figure 1.2: A Super-frame in a beacon enabled mode

Here the Beacon Interval (BI) is given by,

$$BI = aBaseSuperframeDuration \times 2^{BO} \tag{1.1}$$

and the Super frame duration (SD) is given by,

$$SD = aBaseSuperframeDuration \times 2^{SO} \tag{1.2}$$

In the above equations BO and SO are the Beacon Order and Super frame Order respectively.

$$0 \leq SO \leq BO \leq 14 \tag{1.3}$$

For more details please refer to [1] and [6]. In the second category of operations, there is no `Coordinator` node and `Devices` transmit data at intervals controlled by some algorithm, such that collision between different transmission in minimized. Here, the focus of discussion will be beacon enabled operations.

1.3 Muti-hop Routing

Generally, a WSN has a hierarchial, tree based structure, where the upper most node, called `Sink`, collects the information from the network. Each node in the network, tries to send its data towards a node, one step higher in the tree, which is termed as its `Parent` nodes. The node that sends data is called the `Child` of that particular `Parent` node. A `Parent` node can have many `Child` nodes. Each node can generate its own data and send upwards in the network, together with the received data from its `Children`, or, it can just act as a relay node and forward their data. Finally, the data from all over the network is received, at the `Sink` node.

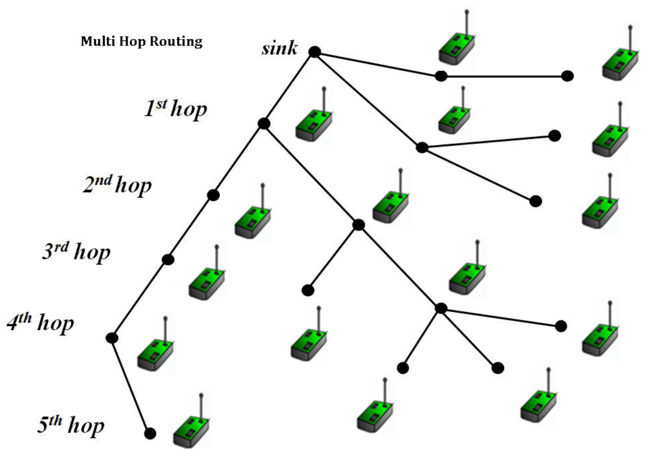


Figure 1.3: A Multi hop Wireless Sensor Network

1.4 Collection Tree Protocol (CTP)

There are many algorithms and protocols available to perform data collection in wireless sensor networks (WSNs) . The Collection Tree Protocol (CTP) is widely regarded as the reference protocol for data collection. CTP provides for "best-effort anycast datagram communication to one of the collection roots in a network " [9]. Further description of the CTP is mentioned in the section 2.1.

1.5 Main Issues (CTP)

For running the CTP over IEEE 802.15.4 MAC, the Extended GTS implementation (E-GTS) [5] was chosen. The main difference between the IEEE standard and this modification is that, the number of time-slots in a Super frame have been increased from 16 to 32. Also, any number of TS can be used for CFP as opposed to a maximum of 7 in standard.

MAC Layer Issues The E-GTS implementation assumes that the transmission happens only between the `Coordinator` and a `Device`. Also, a `Device` gets woken up in its allocated time slot period, transmits / receives in it, and then goes to the sleep mode. For CTP to run, both of these things have to be modified. First, the CTP communication can be between any two nodes in the network, depending upon the tree hierarchy. Therefore, the `Device` to `Device` communication has to be made possible. Also, once a schedule is decided and broadcasted by the `Coordinator`, then a `Device` continues to transmit / receives in it's allocated time slot. In case of a parent change by the CTP layer, an appropriate strategy has to be decided such that the whole operation is seam-less from the routing layer perspective. Also, its is assumed that the `Coordinator` can reach all the `Devices` in the network, but the opposite might not be true.

Routing Layer Issues As mentioned before, CTP generates two types of traffic, a routing control traffic which is sent aperiodically and a data traffic which is sent periodically. CTP is normally run over a CSMA-CA based MAC, such that there is no fixed time slot allocated to a given node and all messages generated by the routing layer (whether data or beacon frame) are sent the time they are generated. It means that there is very little delay in actual generation of a message and its transmission. Also, if a data packet is not acknowledged, it is re-transmitted instantly. Re-transmission is done until the packet is acknowledged or re-tries exceed a fixed number. But in case of Beacon Enabled network, the two types of have to be routed carefully through two Super frame periods.

Chapter 2

CTP over TDMA-MAC

This chapter describes the TinyOS communication stack, standard CTP implemenation and the modifications done in the MAC and the Routing layer to make run the CTP over the 802.15.4, Beacon Enabled network.

2.1 TinyOS Communication Stack

In TinyOS, the communication process can be described in terms of a stack, shown in the figure 2.1

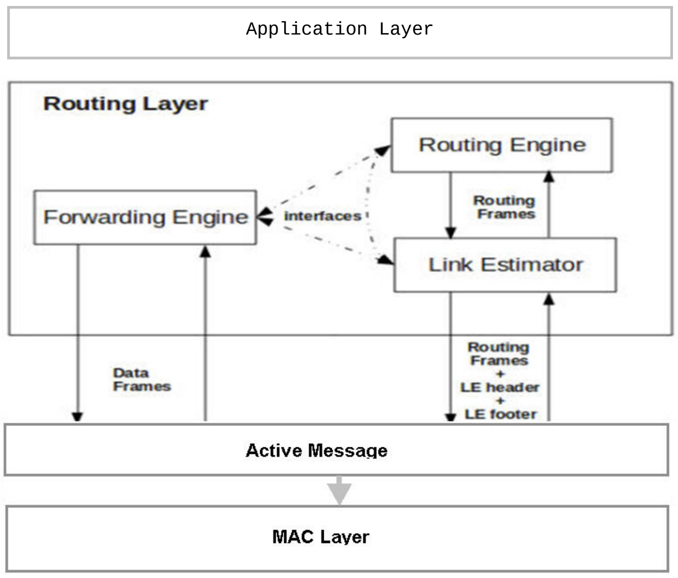


Figure 2.1: TinyOS Communication Stack

User data is generated in the application layer. It is passed to the routing layer. Routing layer, as mentioned before, employ CTP for selecting next hop destination address or the `Parent`. CTP is a distance vector routing protocol designed for sensor networks. CTP computes the routes from each node in the network to the root (specified destinations) in the network. Its tasks include, radio link quality estimation, parent selection, topology update, loop detection, duplicate packet suppression etc. The standard implementation of CTP uses three logical modules namely, `Routing Engine - RE`, `Forwarding Engine - FE`, `Link Estimator - LE`. Following paragraphs, briefly describe the working of these modules.

Routing Engine - RE The Routing Engine, an instance of which runs on each node, takes care of sending and receiving beacons as well as creating and updating the routing table. This table holds a list of neighbors from which the node can select its parent in the routing tree. The table is ?lled using the information extracted from the beacons. Along with the identifer of the neighboring nodes, the routing table holds further information, like a metric indicating the "quality" of a node as a potential parent. In the case of CTP, this metric is the ETX (Expected Transmissions), which is communicated by a node to its neighbors through beacons exchange. A node having an ETX equal to `n` is expected to be able to deliver a data packet to the sink with a total of `n` transmissions, on average. More details can be found in [2].

Forwarding Engine - FE The Forwarding Engine, takes care of forwarding data packets which may either come from the application layer of the same node or from neighboring nodes. FE is also responsible of detecting and repairing routing loops as well as suppressing duplicate packets.

Link Estimator - LE The Link Estimator takes care of determining the inbound and outbound quality of 1-hop communication links. Metric that expresses the quality of such links is referred as the 1-hop ETX. The LE computes the 1-hop ETX by collecting statistics over the number of beacons received and the number of successfully transmitted data packets. From these statistics, the LE computes the inbound metric as the ratio between the total number of beacons sent by the neighbor over the fraction of received beacons. Similarly, the outbound metric represents the expected number of transmission attempts required by the node to successfully deliver a data packet to its neighbor. To gather the necessary statistics and compute the 1-hop ETX, the LE adds a 2 byte header and a variable length footer to outgoing routing beacons.

CTP generates two types of messages. First are the routing messages (also called routing layer beacons) that are used for tree construction and maintenance. For the transmission of these messages, CTP uses adaptive beaconing. When the topology is inconsistent and has problems, beacons are sent at a fast rate. The beaconing rate is reduced exponentially as the network settles down . Thus, CTP can quickly respond to adverse wireless dynamics while incurring low control overhead in the long term. The other type of messages are the CTP data messages, that are generally sent periodically, to report application data to the sink. Figure 2.2 shows different types of CTP messages.

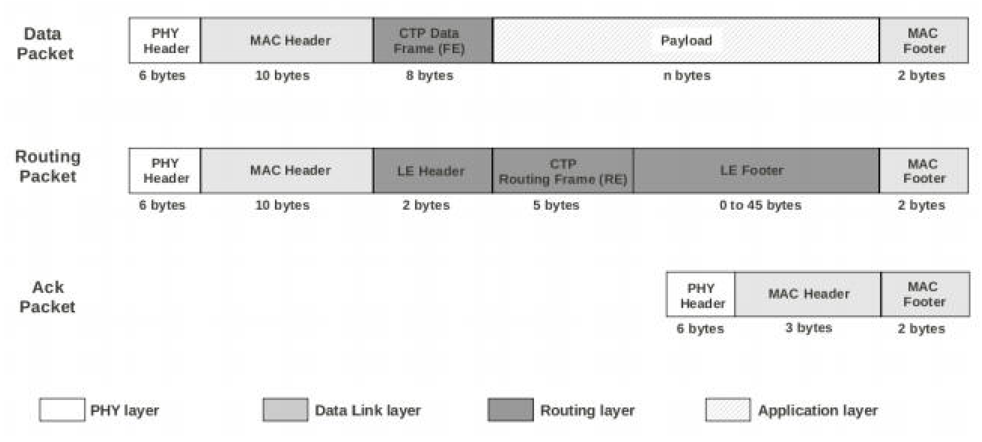


Figure 2.2: CTP Messages

CTP beacon frames are broadcasted where as the CTP data frame are unicasted (directed towards the parent).Routing beacons are passed over by the RE to the LE before transmission. Upon reception of a beacon, the LE extracts the information from both the header and footer and includes it in the so-called link estimator table . This table holds a list of identifiers of the neighbors of a node, along with related information, like their 1-hop ETX or the amount of time elapsed since the ETX of a specific neighbor has been updated. In contrast to the routing table, which is maintained by the RE, the link estimator table is created and updated by the LE. These tables are however tightly coupled. The information available from the link estimator table is used to fill the footer of outgoing beacons, so that nodes can efficiently share neighborhood information [2]. CTP frames (both beacon and data) are shown in figure 2.3. Each of the messages is assigned an CTP Identifier.

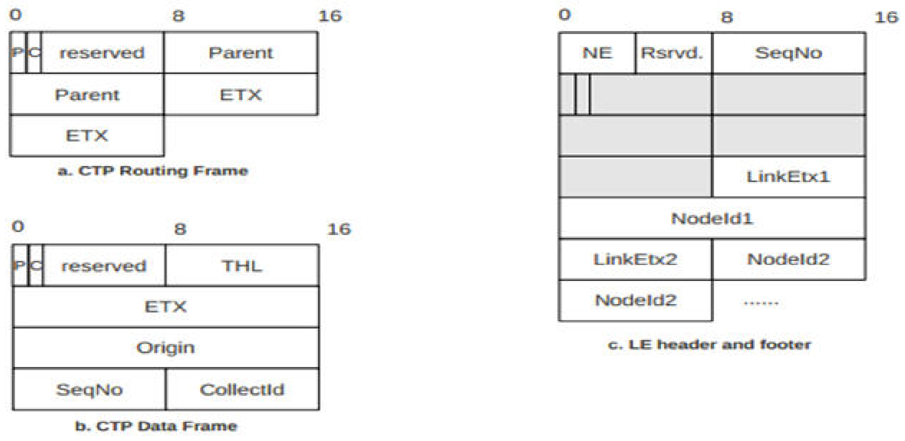


Figure 2.3: CTP Messages

After the message has been through Routing layer, it is passed on to the Active Message

(AM) layer. AM layer is a sort of buffer layer between the routing layer above and the MAC layer beneath. It is here that the message arriving from the routing layer (CTP beacon or CTP data frames) are sorted according to their Identifier and sent to the MAC layer with appropriate setting e.g. CTP beacons are sent with out acknowledgement request and are broadcasted, where as the CTP data frames are sent to MAC layer with both Acknowledgement request and to a specific destination address. Reverse is also true for the case of reception of data from the MAC layer below i.e. AM layer sends the messages to the right component, data to FE and beacons to RE. Typically , CTP uses default AM layer that connects it with TinyOS UnSlotted-CSMA MAC.

MAC layer is normally the performing tasks related to the CSMA-CA, serves the send request from the AM layer or pushes the data upwards after the reception of data.

2.2 Application Layer Modification

The basic application for the CTP is defined in [8]. Here that example is modified for our purpose. Application layer generates the data. In anetwork running CTP, there can be three types of nodes. For each type, a seperate code folder is created.

Originator Node

A Originator node generates the application layer data. It sends it own data to its `Parent`, and it can also forward the data received from its parent. The code for this type of node is located at
`/local/src/tinyos-2.x/apps/EasyCollection/Originator`.

Relay Node

A relay node doesnot generate information of its own, rather it just forwards the data it receives. The code for this type of node is located at
`/local/src/tinyos-2.x/apps/EasyCollection/Relay`.

Sink Node

A Sink node is the collector of all information in the network. Every node that generates data, sends it to the Sink node.The code for this type of node is located at
`/local/src/tinyos-2.x/apps/EasyCollection/Sink`.

2.3 Routing Layer Modifications

We know that CTP generates two types of traffic, a periodic data traffic and a beacons that are sent at a exponentially decreasing rate that is decided by the trickle timer, defined in the RE [9].

Routing of CTP traffic

In one Super frame, a device might need to transmit more than one times. Indeed, when the trickle timer is reset (e.g. in case of topology update the), the period of the timer is set to be around 8 ms. It roughly doubles after transmission of every CTP beacon frame.

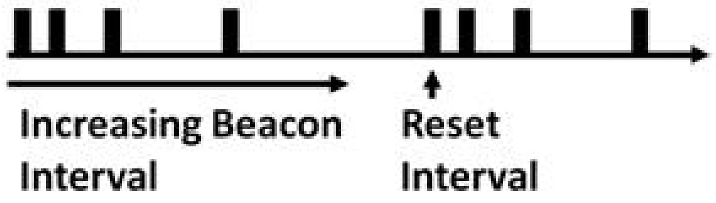


Figure 2.4: Adaptive beaconing [9]

It is also known that, a super frame has to portions in the Active period, a CAP and a CFP, 1.2. Now either a node should be assigned multiple time slots for transmission in a SF or each node should be allocated one time slot and the two traffic be routed through two different portions of SF. In the first case, we see that there is a greater loss of system capacity than the second case. Also, CTP beacons are broadcasted and data frames are supposed to be unicast. Therefore it was decided to route the CTP beacons through the CAP and CTP data frames though the CFP. In CAP, all devices transmit with CSMA-CA enabled. Although, there are still some time slot allocated for the CTP beacon broadcast (a necessary evil here), the reduction in capacity is far lesser than in the other option.

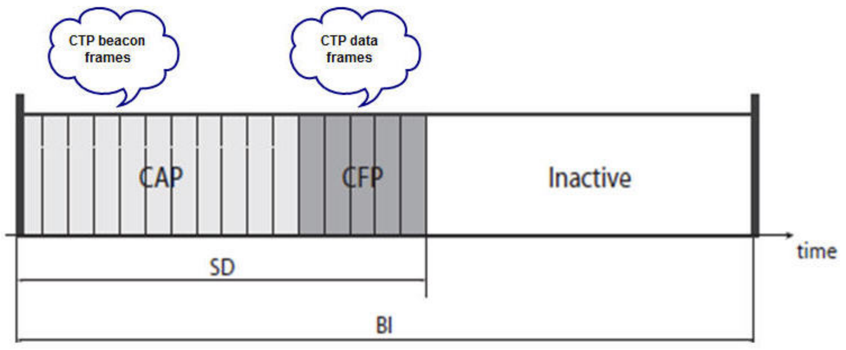


Figure 2.5: CTP traffic routing

An event is signalled from the AM layer that tells the Routing layer about the start and the end of CAP and the start of the time slot. It is done so that the CTP sends the two types of the traffic in the right portion of the SF. For this purpose an interface named `Capisrunning` is created that has following events.

```
interface Capisrunning {

  /**
   * Tells that the CAP has started
   *
   * @param 'uint32_t t0'  - Super Frame start
   * @param 'uint32_t dt'  - CAP duration
```

```
    * @return void.
    */

   async event void Caphasstarted(uint32_t t0 , uint32_t dt);

   /**
    * Tells tha CAP has finished
    *
    * @return void.
    */

   async event void Caphasfinished(void);

   /**
    * Tells that my TS has begun
    *
    * @return void.
    */

   async event void MyTShasStarted(void);
}
```

Trickle timer re-adjustment

The minimum period of the trickle timer was changed from 8ms to 24 ms. This was done because there are only limited number of time slots in CAP allocated to for CTP beaconing.

Tracking the beacons not sent

Due to the fact mentioned above, it is quite possible that the trickle timer fires in the CFP, when CTP beacons are not allowed to be transmitted. Therefore, the idea is to count the number of times the trickle timer in `RE` is fired during CFP and to transmit it during the next CAP.

Sink as parent

In the current implementation, the `Sink` node (routing layer entity) is assumed to be same as the `Coordinator`. A problem can arise when the coordinator can reach all of nodes in the network (which it has to do because every node listen to its TDMA-beacon, and synchronizes themselves with it), but viceversa is not true. This can be the case when nodes other than the `Coordinator` are transmiting with a low power level. In that case, majority of the nodes will try to select `Coordinator` as parent because of its low 1-hop ETX. But on other hand, `Coordinator` might not be listening their CTP-beacon. An example of such as scenario is depicted in figure 2.6

This will lead to wrong parent selection. To avert this scenario, each node checks the footer of incoming CTP routing frame from the `Coordinator`. If a node (`Originator` or `Relay`) find its $< NodeID, Etx >$ tuple in the footer (please refer to figure 2.3), then

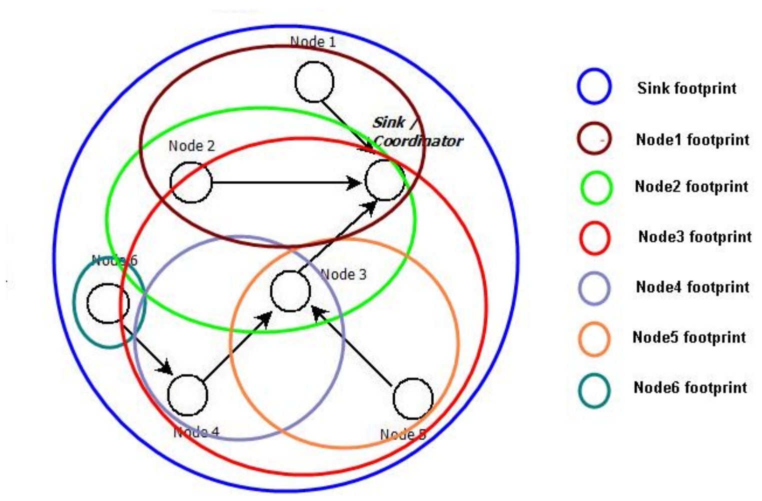


Figure 2.6: Limited range sceanrio

beacon is passed from `LE` to `RE`, and it can add the `Sink / Coordinator` to its Neighbor list. Otherwise, `LE` drops that CTP beacon message.

CTP data aggregation

Lets assume that all of the node in the network are the `Originators`. This means that in addition to sending their own data to their respective `Parents`, they will also forward the data from their `Children`. As standard CTP works with the CSMA-CA, forwarding is straight forward. Node send their own data first and then forward the enqueued messages of their children [2]. Unless there is a collision and a subsequent re-transmission, the delay in the data forwarding is quite small. In the current case, CTP is run over a beacon enabled 802.15.4 network. Here each node has only one time slot in CFP for the data transmission. Three soultions are possible for this problem. The first is that nodes should be assigned multiple time slots in CFP in a SF. But this lead to reduction in the system capacity. The second solution could be to do both sending and forwarding of messages in a single timeslot. Here the problem is that a time slot, depending on SO, is not very long. E.g. for BO = 6 and SO =5, timeslot duration is about 30 ms. Depending on the total transmission time, which is not in-significant as compared to the time slot duration, there are a limited number of messages that can be sent in timeslot duration. The last option is to aggregate all the received messages from the `Children`, append own data to it and send the aggregated message. In the current implementation, this approach is followed.

To make this happen, the six reserved bits in the first byte (Options byte) of the CTP data frame header are used to indicate the number of appended messages in any frame. Lets start with the farthest nodes in the network, which put the bits value to 0 and sends the message to its `Parent` . A `Parent`, upon receiving the message, first checks the value of the reserved bits, extract the payload and saves in a buffer. If the value of the reserved bits is greater than 0 than a loop is run to extract all of the appended message. This is done for message received from all the children. A counter value is incremented for each of individual message. Finally, the node constructs a new message in which, it put it own application payload to the beginning, and appends the buffer containing the aggregated received message, after its own payload. Finally the six reserved bits of the Options byte of the CTP data frame header are set to the counter value plus one (one for own payload). Then the message is sent further upstream (towards the `Sink`). The process is done at every intermidiate node until the message arrives at the `Sink`. Figure 2.7 shows the process.

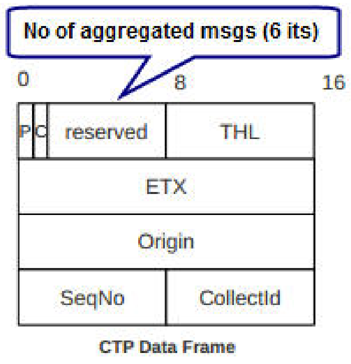


Figure 2.7: CTP data frame aggregation

Data re-transmissions

In the standard CTP running over the CSMA-CA, message is re-transmited for a maximum of 30 times before the message is discarded (or re-transmission is successful). Again, re-transmission takes place as soon channel is sensed clear. This clearly can not be done in the current case, as there is only one timeslot per super frame per node for data transmission. So for data re-transmission two things are done. First, re-transmission is done in the CAP portion of the super frame. The rationale is that, for most of the times, CAP can be found sufficiently empty (due to adaptive nature of CTP beaconing) and re-tries can be done here. This is done inside the event `Capisrunning.Caphasstarted(uint32_t t0 , uint32_t dt)`. Second, the maximum number of re-tries is reduced to 5. This is quite reasonable. First because due to GTS, there is a high guarentee that the message will reach destination in the first attempt (of course if there is external interference, nothing can be said with certainty). And therefore, at a time when the network is stable (beacons being sent less often), channel can be thought to be relatively idle and the re-attempt will likely to be successful. Second, as the number of time slots in the CAP can be limited, 5 seems to be a reasonable number.

The above mentioned changed were made to the components at the location `/local/src/tinyos-2.x/tos/lib/net/ctp_ tdma`

2.4 AM Layer Modifications

As mentioned in the previous sections, AM acts like a buffer layer between the Routing and the MAC layer. In the standrad CTP, default AM layer for the TinyOS is used. It is located at `/local/src/tinyos-2.x/tos/lib /mac/tkn154/TKNActiveMessageP.nc`. This AM layer is wired to the default CSMA-CA based MAC modules. For the current case a new MAC has to be defined. In fact there should be two instances of AM layers, one for the `Sink / Coordinator` node and one for the `Originator / Relay` node. It was also mentioned that each type of the CTP messages, is assigned one unique ID also called AM message type. In this way the AM layer can provide different services to the routing layer, which does not have to bother about way a particular message is sent. E.g. a CTP beacon is broadcasted with out the acknowledgement request (through the CAP), while the CTP data frame is sent to a particular node with acknowledgement request (through the CFP). Routing layer just specifies the AM messages types using one 1 byte. The one byte becomes the first byte of the MAC payload and is appended to the left of the CTP frames as shown in the figure 2.8,

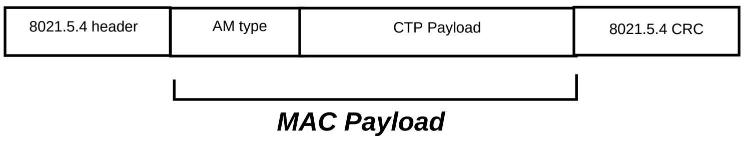


Figure 2.8: A 802.15.4 MAC frame showing the AM message type byte

Same is true for the reception of the frame from the MAC layer below. CTP beacons and data frames are sent to the appropriate components (beacons to LE and data frames to FE) by looking at the AM type byte.

Sink node AM Layer

The main difference of this new AM layer with the default one mentioned in the above paragraph is that, here the network is beacon enabled and therefore the `Coordinator` has to send periodic beacons and maintain a super frame. As in the current implemenetation, `Sink` and the `Coordinator` are the same node, therfore specifications for the MAC are defined here like

- Node ID definition
- BO definition
- SO definition
- Frequency channel selection
- Transmit power selection

The new layer is defined in `/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/ tkn154-gts/tos/lib/mac/t`

Originator / Relay node AM layer

Here again, in contrast with the CSMA-CA based MAC, `Originators` or `Relays`, have to perform extra tasks. They include

- Node ID definition
- Beacons searching from a coordinator
- Beacon its tracking
- Beacon loss detection and subsequenct action

This new MAC layer is defined in `/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts/tos/lib/mac/tkn154/ TKNActiveMessageP.nc`.

Also, the events for the interface `Capisrunning` are also defined here, that are signalled to the CTP component as mentioned in the previous section.

In the routing layer, the component `CtpP.nc` is wired to the component `ActiveMessageC.nc`, where the new AM layer is defined.

2.5 MAC Layer Modifications

Before discussing the modification done to the MAC layer, it is better to present an example here. Lets consider a WSN with the topology as mentioned in the figure 2.9 and the CTP is running over this network.

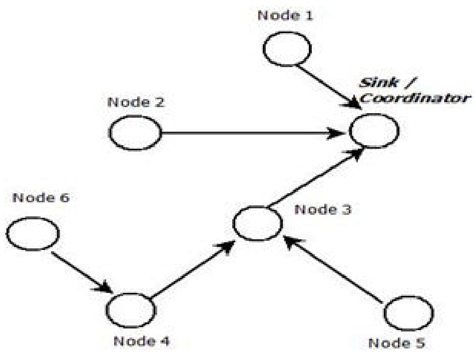


Figure 2.9: An example WSN

In the CTP network, a `Device` and change its `Parent`. It was mentioned before that standard E-GTS implementation allows communication between the `Coordinator` and `Devices`. Hence the MAC layer always fills up the destination ID in the message header as Coordinator ID. But in the case of CTP, communication happens not only between the `Coordinator` and `Devices`, but also amoungst `Devices` themselves. For CTP to run over the E-GTS code, this issue has to be solved. Another point is that during the CTP operation, any node can re-select its `Parent`. What should a MAC layer do in case of a parent change, is also to be addressed.

Parent change problem

There are at least two ways in which the time slots can allocated to the nodes in the network. The first one is to assign a time slot to each individual link in the network.

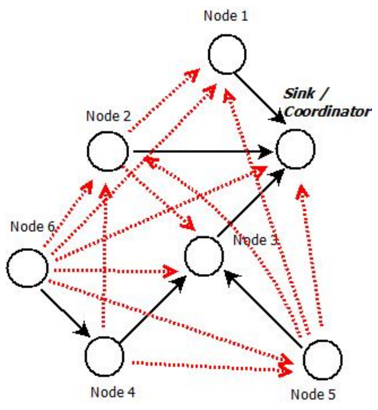


Figure 2.10: A timeslot per possible link

This means that when a device switches its parent, it simply hops over to a new time slot, corresponding to that of the new link. Here, we see a big loss of system capacity. Hence, this solution is not feasible. The other possible solution is to change the destination ID in the MAC header, in case of a parent change by the CTP. Now there is no time slot switching. Also, system capacity is not reduced by following this approach. This is shown in the figure 2.11,

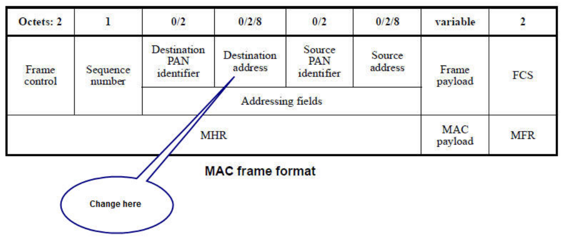


Figure 2.11: A MAC frame

Majors changes were made to the `DataP` component, located in
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts/tos/lib/mac/tkn154/DataP.nc`
and
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts/tos/lib/mac/tkn154-sink/DataP.nc`.

Node state change

In the E-GTS implementation [5], each node just wake up during its own time-slot and sleep for the rest of the CFP duration and in the in-active period as well. But if the time slot allocation strategy discussed in the previous section is followed, every node should always be awake during the whole active period. Also, during the CAP nodes have to be made awake so that they can send / receive the CTP beacons and the CTP data frames re-transmissions. For this matter, significant changes were made to the `CfpTransmitP` and `DispatchSlottedCsmaP` component located in
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts/tos /lib/mac/tkn154/DispatchSlottedC`
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts/tos /lib/mac/tkn154-sink/DispatchSlo`
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts-mod-all/tos/lib/mac /tkn154/CfpTransmitP.nc`
and
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/tkn154-gts-mod-all/tos/lib/mac /tkn154-sink/CfpTransmitP.nc`.

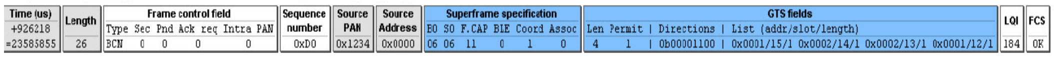
Chapter 3

Scheduling

This chapter describes the scheduling method used. Two methods were tried, a static time-slot allocation method and dynamic time slot allocation method.

3.1 Static Scheduling

Static scheduling refers to the fixed time slot assignment per node by the `Coordinator`. In this case, whene ever a node joins in the network, it searches the TDMA-beacon from the `Coordinator` for its time slot information. For instance, figure 3.1 shows a snapshot of a TDMA beacon message, captured by the Packet Sniffer,



```
Time (us)  Length  Frame control field             Sequence  Source  Source   Superframe specification       GTS fields                                                                                LQI  FCS
+926218            Type Sec Pnd Ack req Intra PAN  number    PAN     Address  BO SO F.CAP BLE Coord Assoc   Len Permit | Directions | List (addr/slot/length)
=23585855  26      BCN  0   0   0       0          0xD0      0x1234  0x0000   06 06 11     0   1     0       4   1      | 0b00001100 | 0x0001/15/1 0x0002/14/1 0x0002/13/1 0x0001/12/1   184  OK
```

Figure 3.1: A TDMA Beacon

Here the nodes with ID 1 and 2 are assigned two time slots each, one for reception and one for transmission. Time slot length for both nodes is 1 (please refer to [6, p.22] for more details). When any of these nodes joins the the network, it searches the TDMA beacon, and then it finds timeslot for transmission /. reception. Here one point should be made clear that the figure above was generated using the normal GTS code (16 time slots) and not with the E-GTS code (32 time slots), as the purpose is just to illustrate the concept.

The code for static time slot allocation is located in

`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/ tkn154-gts/tos/lib/mac/tkn154-sink/TKNActiveMe`

This is done by using the command `GtsUtility.addGtsEntry`, to add a new entry in the Coordinator GTS database, corresponding to nodes in the network. Then the MAC layer is updated by signalling the event `GtsSpecUpdated.notify`

```
ieee154_GTSdb_t* GTSdb;
GTSdb = call SetGtsCoordinatorDb.get();
GTSdb->numGtsSlots = 0;

call GtsUtility.addGtsEntry(GTSdb,1,15,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,2,14,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,1,13,1, GTS_RX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,2,12,1, GTS_RX_ONLY_REQUEST);

signal GtsSpecUpdated.notify(TRUE);
```

The first argument of the command `GtsUtility.addGtsEntry` is the reference to the GTS database, the second argument is for the Node ID, third argument is for the Timeslot Number (starts with the, Maximum no. of TS in a super frame - 1) and the forth argument is for the Transmission direction (GTS_ TX_ ONLY_ REQUEST means node TX and GTS_ RX_ ONLY_ REQUEST means that node RX during that).

Here the problem can arise if the schedule for a node is not set by the `Coordinator`, and the node joins the network. In that case the node cannot transmit the data, as it has to timeslot allocated.

3.2 Dynamic Scheduling

To alleviate the problem faced by the static scheduling, a dynamic approach is required where the schedule is not pre-decieded and the nodes are assigned timeslot based on their request. Generally, the time-slot allocation request by a node is done, at the time of its entry in to the network.

Strategy

The dynamic scheduling is envisioned as a four step process. Lets assume the same WSN as mentioned in the figure 2.9

CTP tree construction phase

In the first phase, all `Devices` first synchronize to the `Coordinator`. Then they start sending the CTP beacons. The process continues until every one in the network has selected a parent. At tha moment, it is said that the CTP tree has been constructed

Topological information propagation phase

This is the second phase, where the each node in the network propagates its location in the network. This is done by each node sending its hop count to its `Parent`. A node at the end of the network will start the hop count from 0. `Parent` increments the hop count of it `Children`, appends it own hop-count (which is zero) and forward the whole message to it's `Parent`. The process is continued until the `Coordinator` has heard from every node in the network. The number of individual messages at the `Coordinator` will tell about the number of basic braches and the hop-count of any node will tell about its position or depth in the tree.

Time slot allocation phase

Once the `Coordinator` has heard from every body in the network, then it assign the time-slots to the nodes based on their hop-count. The idea is that the time slot allocation should be in ascending order of hop-count i.e. nodes with hop count zero should be assigned time slot first, thtn to the ones with hop-count value 1 , 2 and so on. This makes sure that a `Child` does not get the time slot after its `Parent`. After the schedule has been decided, it is transmitted in the TDMA beacon by the `Coordinator`.

CTP data flow

Devices continously look for the schedule in the received TDMA beacons. As soon as they find their timeslot in it, they start sending the data.

Implementation

In order to implement the dynamic scheduling algorithm, many changes had to be done to the standard CTP code as well as the AM layer code. The implemenatational details of the above mentioned strategy are mentioned here.

Algorithm

The main concept in the dynamic scheduling is the topological information propagation. This is a way for the `Coordinator` to have an idea about the location of other nodes in the network. The idea is to create a special message that is aggregated in the similar way as that of CTP data frame. This is done by modifying the CTP routing frame. Instead of using the tuple of $< 1 - hopETX, NodeID >$, a tuple of $< Hopcount, NodeID >$ is used. Second difference is that, instead of broadcasting, the modified CTP routing frame (MRF) is sent by a node towards it `Parent`. Third difference is that there is no embedded CTP beacon message inside the modified routing frame, as it is in the case of normal routing frame. Fourth, the header of the two mesasges are also different. In case of a modified routing frame, header consist of the number indicating the no. of appended tupels in the messages, similar to the case of data frame aggregation.

Each node keep track of its children. Similar to the concept of a neighbour table, a children table is kept by each node. Table contains three entries for each Child. They are the, node Id of the `Child`, a flag indicating whether modified routing frame has been heard from that child and a flag indicating whether that child has a time slot or not. Time slot information is conveyed by using the 6 reserved bits in the CTP beacon embedded in the routing frame. A node sets those six bits to a value of 1 if it does not have a time slot and to 0 if it has a time slot. Once the tree construction phase is over, each node generates a

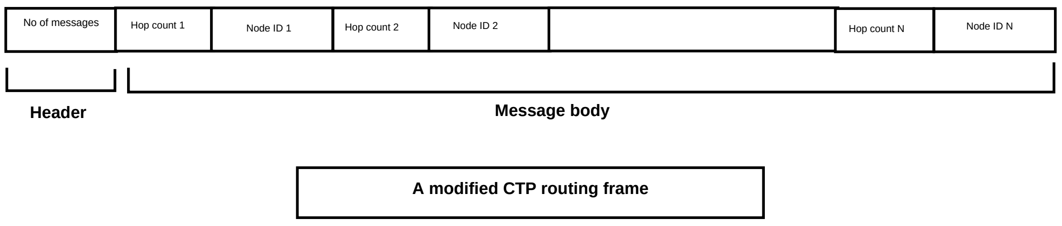


Figure 3.2: A modified CTP routing frame, used to carry hop-depth of a node

MRF and sends it to the parent. Parent waits until it gets MRFs from all its `Children`. At that moment the `Parent` check whether it has already sent one MRF before. If this is false, it puts its own tuple at the beginning of the aggregated tuples and pushed the data towards its own `Parent`. On the other hand, if a node has already sent a MRF, then instead of putting its Node ID in the first tuple, it puts 1000. This can be the case when a new node appears in a already scheduled network. In that case, 1000 in the node ID tells the `Coordinator` to not to re-assign a time slot to the node that is forwarding the messge, rather to assign the time slot only to the node which has a valid Node ID in the MRF (a valid node ID is assumed to be quite less than 1000).

In the similar way, the MRF gets bigger and bigger as it reaches near `Sink / Coordinator`. Coordinator sorts the received tuple w.r.t the hop-count and put it in a array. It ignore the tuples with Node Id as 1000. Coordinator waits for a specified duration for the reception if MRFs from all over the network. Once that time is over, it sends the array to the AM layer for the time slot allocation process (scheduling).

Link Estimator (LE) Modifications Three new functions are defined in the `LinkEstimator`.

```
/* signals to RE that a modified beacon has been
 * received from this child*/
event void receivedMB(am_addr_t neighbor);

/* lets RE notifies the LE that it can send the MB now */
command error_t canTxMB(am_addr_t neighbor, bool FirstMBSent);

/* lets RE notifies the LE, for the Sink node, that
*  it can pass scheduleing message to the MAC Layer */
command error_t PassMsgToMAC(void);
```

Routing Engine (RE) Modification RE is changed so that it keep track of the children as well. Also, the major logic for the dynamic scheduling algorithm is contained in the RE component. Also before the parent reselection, it is checked whether the new parent has a time slot before or after the time slot for the node its self. If it is before, the new potential parent is rejected. It is selected otherwise

AM Layer Modifications AM layer is also changed to maintain the schedule. Both `Coordinator` and the `Devices` keep track of the overall schedule. `Devices` keep track of the schedule by listening to the TDMA beacons and saving the beacon payload. New interface has been created to let the routing layer inquire the AM layer about the schedule or setting the schedule.

Chapter 4

Test Examples

This chapter describes the test examples for both the static and dynamic scheduling case. Go to the following location:

```
/home/david/local/src
```

There will be many folder with the name starting with `tinyos-2.x`. Each one of the folder contains a copy of the TinyOS source code with different modifications. First decide what topology is needed in the network i.e. how many `Originators` and how many `Relays`. It is assumed that test bed is not in place and the motes are very close to each other. In that case, a multi-hop network cannot be formed as all of the nodes will try to connect to the `Coordinator / Sink`. To circumvent the problem, certain nodes can be barred from entring the neighbor list of a node. In this way the parent selection can be controlled. This is done by going to the RE component located in the folder

```
/local/src/tinyos-2.x/tos/lib/net/ctp\_ tdma
```

and making changes in the **`event`** `message_ t* BeaconReceive.receive`.
Lets consider the following network.

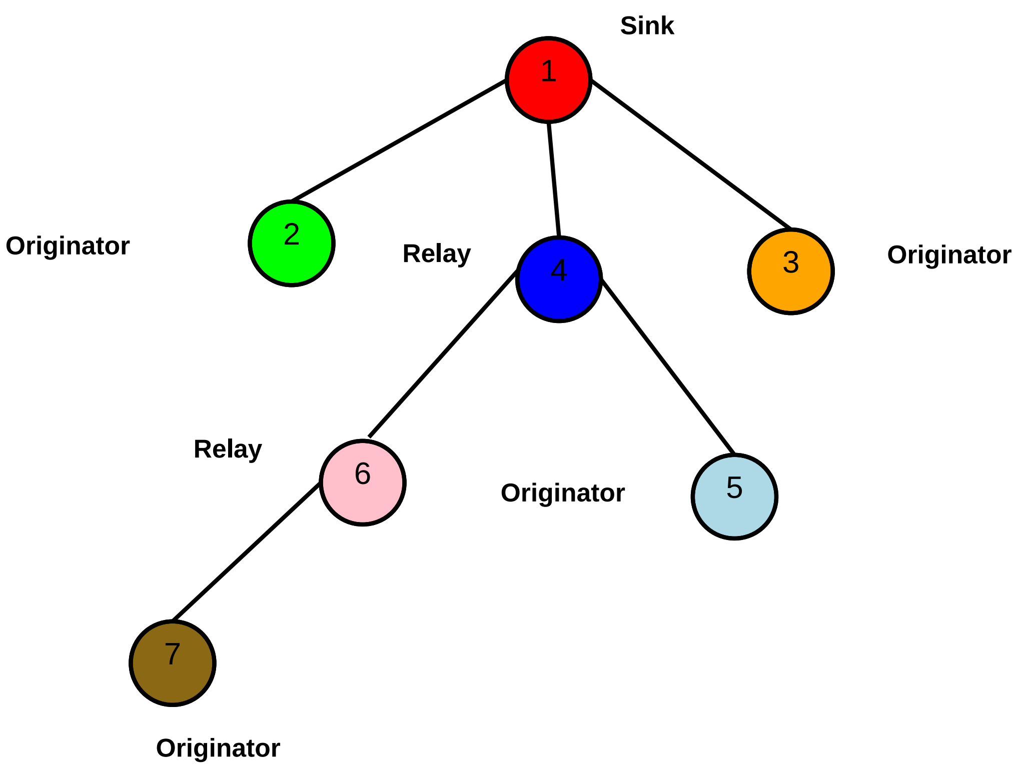


Figure 4.1: An example network

There node 2, 3 and 4 are directly connected to the Sink whose ID is 1. Nodes 5 and 6 are connected through 4 and node 7 is connected to the sink via 6 and 4. Node 4 and 6 are `Relays`, Node 1 is the `Sink` whereas the other nodes are `Origninators`. First decide the schedule by going to location,

`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/ tkn154-gts/tos/lib/mac/tkn154-sink/TKNActiveMe`

and modifying the function **void** `setDefaultGtsDescriptor()`

```
ieee154_GTSdb_t* GTSdb;
GTSdb = call SetGtsCoordinatorDb.get();
GTSdb->numGtsSlots = 0;

call GtsUtility.addGtsEntry(GTSdb,2,31,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,3,30,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,4,29,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,5,28,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,6,27,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,7,26,1, GTS_TX_ONLY_REQUEST);

signal GtsSpecUpdated.notify(TRUE);
```

To set up a network with this topology replace the code in the **`event message_ t* BeaconReceive.receive`** by following code

```
am_addr_t from;
ctp_routing_header_t* rcvBeacon;
bool congested;

// Received a beacon, but it's not from us.
if (len != sizeof(ctp_routing_header_t)) { return msg;}

//need to get the am_addr_t of the source
from = call AMPacket.source(msg);

rcvBeacon = (ctp_routing_header_t*)payload;

congested = call CtpRoutingPacket.getOption(msg, CTP_OPT_ECN);

if (rcvBeacon->parent == TOS_NODE_ID){
childrenTableUpdateEntry( call AMPacket.source(msg),
call CtpRoutingPacket.getTsInfo(msg) );
}

if( ( my_ll_addr == 5 && (from == 1 || from == 2 || from == 3) ) ||
( my_ll_addr == 6 && (from == 1 || from == 2 || from == 3 ) ) ||
( my_ll_addr == 7 && (from == 1 || from == 2
        || from == 3 || from == 4 || from == 5) ) ){}

else {
        if (rcvBeacon->parent != INVALID_ADDR) {
            if (rcvBeacon->etx == 0) {
                call LinkEstimator.insertNeighbor(from);
                call LinkEstimator.pinNeighbor(from);
            }

            routingTableUpdateEntry(from,
                rcvBeacon->parent, rcvBeacon->etx);

            if (rcvBeacon->parent == TOS_NODE_ID){
            childrenTableUpdateEntry( call AMPacket.source(msg),
            call CtpRoutingPacket.getTsInfo(msg) );
            }

             call CtpInfo.setNeighborCongested(from, congested);
        }

        if (call CtpRoutingPacket.getOption(msg, CTP_OPT_PULL)) {
                resetInterval();
```

```
                  childTableClear();   // Clear the child table
         }

    }
  return msg;
}
```

This code will bar nodes 1, 2 and 3 from entring in the negighbor table of node 5 and 6, and nodes 1 , 2 , 3 , 4 and 5 from entring in the neighbor table of node 7.

4.1 Test with static scheduling

Go to folder

```
/home/david/local/src
```

The folder with the title `tinyos-2.x-static-scheduling-final` contains the code with static scheduling. Rename it to `tinyos-2.x`. After that open a terminal window, and navigate to the folder

```
/home/david/local/src/tinyos-2.x/apps/EasyCollection/Sink
```

This will make the terminal, available for viewing the screen prints of the `Sink` nodes.

Like wise, open 4 terminal windows, and navigate in each one of them to the to the folder,

```
/home/david/local/src/tinyos-2.x/apps/EasyCollection/Originator
```

These will be the windows for viewing the output of the four `Originator` nodes.

In the same maner open 2 windows, and navigate in each one of them to the to the folder,

```
/home/david/local/src/tinyos-2.x/apps/EasyCollection/Relay
```

These windows will allow us to program the `Relay` nodes and view their outputs.

After all of this is done, program the individual motes by running the command

```
make telosb install.x bsl,/dev/ttyUSBy
```

Here **x** is the node ID as mentioned in the figure 4.1 and **y** is the USB port that particular mote is connected to. After running this command on all of these motes, run the following command to view the ooutput on each of the motes

```
java net.tinyos.tools.PrintfClient 1 y
```

Again, **y** is the USB port that particular mote is connected to.

User can add any number of nodes with an arbitrart topology, but then the code for static allocation located in the AM layer component located at
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/ tkn154-gts/tos/lib/mac/tkn154-sink/TKNActiveMe`
has to be modified to include the GTS entries as mentioned in section 3.1, for rest of the nodes.

4.2 Test with dynamic scheduling scheduling

For testing network with dynamic scheduling, rename the folder titled `tinyos-2.x-dynamic-scheduling-final` to `tinyos-2.x`. It contains the code with dynamic static scheduling. Repeat the same process as mentioned before. User can have any topology, as now the time slot allocatio will be done automatically. Another thing is that, there is an initialization period of around 1 minute, in which the `Coordinator / Sink` waits for the MRFs from every body in the network. Therefore please be patient for that time in the beginning.

4.3 Test for multiple redundant paths

In this application, there are alleast two parallel redundant path of `Relay` nodes. There is one `Originator` node and one `Sink` node. The `Originator` node can select any of the two branches at any given time.

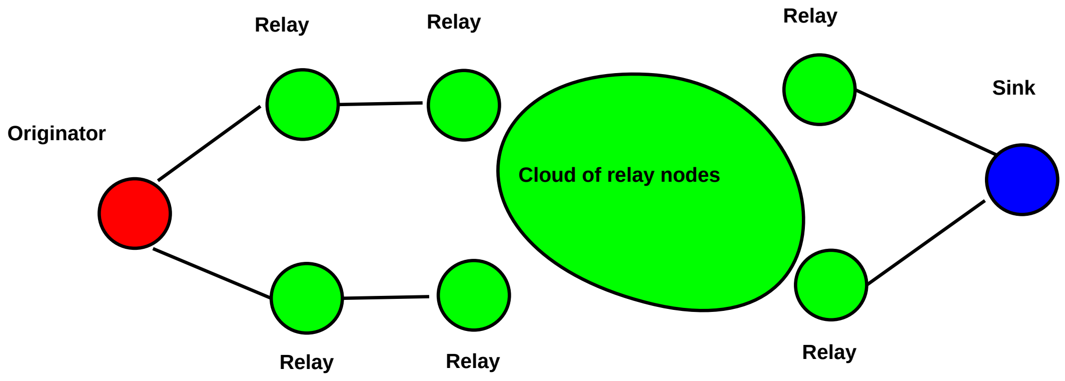


Figure 4.2: Redundant branches

Schedule for this network can be provided similar to the one mentioned in section 4.1, but assign same time slot to the branches at the same hop depth. e.g.Decide the schedule by going to location,
`/local/src/tinyos-2.x/tinyos-2.x-contrib/kth/ tkn154-gts/tos/lib/mac/tkn154-sink/TKNActiveMe`

```
ieee154_GTSdb_t* GTSdb;
GTSdb = call SetGtsCoordinatorDb.get();
GTSdb->numGtsSlots = 0;

call GtsUtility.addGtsEntry(GTSdb,2,31,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,3,31,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,4,30,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,5,30,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,6,29,1, GTS_TX_ONLY_REQUEST);
call GtsUtility.addGtsEntry(GTSdb,7,29,1, GTS_TX_ONLY_REQUEST);

signal GtsSpecUpdated.notify(TRUE);
```

Break a link in any of the two branches and you will see that the node `Originator` will switch to the redundant path.

References

[1] David Andreu. Implementation and performance evaluation of ieee 802.15.4 protocol. Technical report, Automatic Control Lab, KTH, Stockholm, 2011.

[2] Ugo Colesanti and Silvia Santini. The collection tree protocol for the castalia wireless sensor networks simulator. Technical report, Institute for Pervasive Computing, ETH Zurich, June 2011.

[3] Walenegus Dargie and Cristian Poellabauer. *Fundamental of Wireless Sensors Networks, Theory and Practice.* John Wiley and Sons, 2010.

[4] Ian F.Akyildiz and Mehmet Can Vuran. *Wireless Sensors Networks.* John Wiley and Sons, 2010.

[5] Aitor Hernandez. Modification of the ieee 802.15.4 implementation extended gts implementation. Technical report, Automatic Control Lab, KTH, Stockholm, July 22 2011.

[6] Aitor Hernandez and Pangun Park. Ieee 802.15.4 implementation based on tkn15.4 using tinyos gts implementation. Technical report, Automatic Control Lab, KTH, Stockholm, January 14 2011.

[7] IEEE. Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). Technical report, The Institute of Electrical and Electronics Engineers Inc., 3 Park Avenue, New York, NY 10016-5997, USA, October 2003.

[8] Rodrigo Fonseca Omprakash Gnawali Kyle Jamieson Sukun Kim Philip Levis and Alec Woo. Network protocols: Dissemination and collection, http://docs.tinyos.net/tinywiki/index.php/network_ protocols. Technical report.

[9] Rodrigo Fonseca Omprakash Gnawali Kyle Jamieson Sukun Kim Philip Levis and Alec Woo. Tinyos enhancement proposal (tep) 123: The collection tree protocol (ctp), www.tinyos.net/tinyos-2.x/doc/pdf/tep123.pdf. Technical report.