Sensor Network Lab Report

Anousheh Khajeh Nassiri Angela Mizero Leandre Nzabarushimana
dept. Computer Science
University of Antwerp
Antwerp, Belgium
Angela.Mizero@student.uantwerpen.be
Anousheh.Khajeh.Nassiri@student.uantwerp.be
Leandre.nzabarushimana@student.uantwerp.be

***Abstract*—This report mainly presents Radio Duty Cycle measurements for two different MAC protocols. ContikiMAC is the default MAC protocol which keeps the radio on for some time, while CSMA/C always keeps the radio on. To achieve the lowest RDC and delay but to maximize the reliability, ContikiMAC and CSMA/CA MACs were used with different parameters such as sending intervals, channel check rate and packet size.**

I. Introduction

Power consumption is important for wireless sensor nodes to achieve a long network lifetime. To achieve this, low-power radio hardware is not enough. Existing low-power radio transceivers use too much power to provide long node lifetimes on batteries[1]. The purpose of a power-saving duty cycling protocol is to keep the radio turned off while providing enough rendezvous points for two nodes to be able to communicate with each other. The experiments were done using two different MAC protocols (ContikiMAC and CSMA/CA) along with other parameters to see with which combination we have the lowest RDC (Radio Duty Cycle) and also to investigate delay and reliability. We looked into different parameters like channel check rate, sending intervals and packet size in the following sections in order to see how it affects the sensors' RDC. To do so, we first used the Cooja which is Contiki[2] network simulator we then quickly continued our experiments with Contiki operating system and real nodes. Contiki is an open source operating system for the Internet of Things.

II. Test Setup

Our setup consists of two nodes, each connected to a different PC. One plays the role of client and one plays the role of server. The hardware we used in this lab were Zolertia Z1 motes and as a software platform we used instant Contiki-image. All the experiments done in this paper are based on the rpl-udp example.

A. Explanation of rpl-udp code

1) udp client
   Any Contiki application consists of two parts a process control block and a process thread. In udp-client file, the process is begun by: PROCESS_BEGIN() then a connection is established with the remote server by binding the server IP address with its port. Once the connection is established, a timer with the value of the sending interval is started. Once the timer is started, the client is ready to receive incoming packets. The received packets are handled by the tcpip_handler() method. In the tcpip_handler() method, the received data is displayed if there is any data available. Once the timer expires the client sends data to the server by calling the send_packet() method. In the send_packet() method, the buffer size is set, the client's data is added to the buffer and sent to the specified server's address and port.
2) udp sever
   In the udp-server file the process is begun with PROCESS_BEGIN() then a connection is established with the remote server by binding the server IP address with its port. Once the connection is established, the server is ready to receive incoming packets by calling the tcpip_handler() method. In the tcpip_handler() method, the data received from the client is displayed from the uip_appdata which points to the application data in the packet buffer. The server sends a reply to the client by using the connection that was already established.

B. Test Setup Metrics

RDC, Reliability and Delay are metrics that exist in our setup.

- Duty Cycle
  Contiki uses two different MAC-protocols named ContikiMAC and CSMA/CA. To achieve a long lifetime, the radio transceiver must be switched off as much as possible. But when the radio is switched off, the node is not able to send or receive any messages. Thus the radio must be managed in a way that allows nodes to receive messages but keep the radio turned off in between the reception and transmission of messages. As energy efficiency is difficult to measure, duty cycle is used and it is the percentage of time that the radio is on and it is measured with the formula below:
  $$DutyCycle = \frac{on}{on+off} * 100$$
  To find the amount of time that the radio is off we use the following formula:
  off = all_time-all_radio
  all_time represents the amount of time that the mote is on. It is calculated as follows:
  all_time = all_cpu + all_lpm;

Where all_lpm is the accumulated Low Power Mode energy consumption. It is calculated as follows:
all_lpm = energest_type_time(ENERGEST_TYPE_LPM);
all_cpu is the accumulated CPU energy consumption. It is calculated as follows:
all_cpu = energest_type_time(ENERGEST_TYPE_CPU);
all_radio is the amount of time that the radio is on. It is calculated as shown below:
all_radio = energest_type_time(ENERGEST_TYPE_LISTEN) + energest_type_time(ENERGEST_TYPE_TRANSMIT);
Where energest_type_time(ENERGEST_TYPE_LISTEN) is the listen energy consumption for this cycle and energest_type_time(ENERGEST_TYPE_TRANSMIT) is the transmission energy consumption for this cycle.
In order to have access to the ENERGEST function we had to import the powertrace class with the following line:

#include ”powertrace.h”

These changes were made in udp-client.c file.

- Reliability
  In the reliability section we wanted to see if for a particular window all the packets that are sent from the client are received by the server. Since every packet sent and every packet received are labelled with a sequence ID, it was easy to find out if all the packets that were sent by the client were received by the server. After 5 minutes we compare the last sequence IDs both at the client and at the server. If both sequence IDs are the same we conclude that the communication is reliable otherwise we conclude that some packets that were sent were lost.
- Delay
  The delay is the amount of time elapsed between the time the packet is sent and the the time the packet is received. Calculating the delay would have been hard since we do not have a time synchronization method that will synchronize the time of both nodes. However another alternative is to use the round trip time which is the time it takes for a packet to travel from a specific source to the destination and back again (Aknowlegement). To calculate the RTT we first calculate the time that the packet is sent with the following formulae:
  SEND_TIME_NOW = RTIMER_NOW();
  Then at the reception of the server reply we subtract the SEND_TIME_NOW from the current time as follows:
  ROUND_TRIP=(RTIMER_NOW() - SEND_TIME_NOW)
  In Contiki we can calculate the round trip time in seconds by dividing the total round trip time in system ticks by the system ticks per seconds.

$$RTT(seconds) = \frac{RTT(SystemTicks)}{Ticks(persecond)}$$

C. Test Setup Parameters

As the parameters we considered MAC protocols, channel check rate, packet size and sending intervals.

- MAC protocols
  ContikiMAC is the default Contiki radio duty cycling mechanism. To change the default MAC to CSMA/CA we do as below:
  In the file contiki/platform/z1 contiki project can have an optional per-project configuration file, called ’contiki-conf.h’.
  in contikiconf.h instead of default mode of
  #define NETSTACK_CONF_RDC contikiMAC_driver,
  we change it to :
  #define NETSTACK_CONF_RDC nullrdc_driver.
- Channel Check Rate
  Channel check rate is how often the node will wake up to see whether the channel is busy or not. To try a different rate like 16, we go to contiki folder, platform , and we choose Z1, in conf.h, we change the number from 8 to 16 in front of the line :
  #define NETSTACK_RDC_CHANNEL_CHECK_RATE 16
  we change the number to 16.
- Packet Size
  In the experiment we did with payload size, we wanted to see the effect of the packet size on the duty cycle. At the client side we changed the payload size by changing the following from:
  #define MAX_PAYLOAD_LEN 30
  to:
  #define MAX_PAYLOAD_LEN 151
  and
  #define MAX_PAYLOAD_LEN 36
  We also changed the buffer size to 400.
- Sending Interval
  Sending interval is the time that the client waits between sending two consecutive packets to the server. In the udp-client.c code we changed the intervals as shown below:
  #define
  PERIOD 20 (Interval in seconds)
  #define
  SEND_INTERVAL (PERIOD * CLOCK_SECOND)
  (Interval in system clock time)

III. Results

In this section we investigated:

1) The effect of the channel check rate on delay
2) The effect of the sending interval on the reliability
3) The effect of the channel check rate on the duty cycle
4) The effect of the MAC protocols on the duty cycle
5) The effect of the payload length on the duty cycle

A. Delay

In this section, the effect of the channel check rate on delay (RTT) is investigated. Two different channel check rates(8Hz and 16Hz) were used in order to see the effect that the mote sleeping frequency has on the RTT. The sending intervals were chosen starting from 5 and gradually increasing them in order to investigate how the RTT is affected when packets are sent

as fast as possible as compared to when they are sent at a higher interval.

1) ContikiMAC with Channel Check Rate of 8Hz at both Client and Server
   The test is done with the payload size is 8bytes and random sending time with sending intervals of 5, 20 and 30 seconds. With a channel check rate of 8Hz at the client and server, the average RTT is 0.2198
2) ContikiMAC Channel Check Rate of 16Hz at both Client and Server
   The test is done with the payload size is 8bytes and random sending time with sending intervals of 5, 20 and 30 seconds. With a channel check rate of 16Hz at the client and server, the average RTT is 0.3417

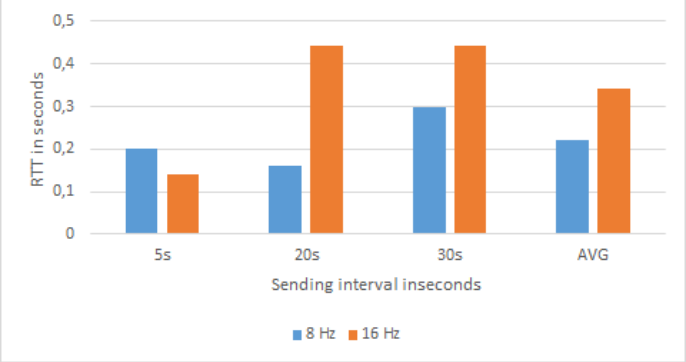

Fig. 1. Resulting RTT from a Channel Check Rate of 8Hz and 16Hz at both Client and Server for ContikiMAC

3) CSMA/CA
   The test is done with the payload size is 8bytes and random sending time with sending intervals of 5, 20 and 30 seconds. However, with CSMA/CA there was no need to change the channel check rate because CSMA/CA always keeps the radio on so the channel check rate parameter is completely ignored.

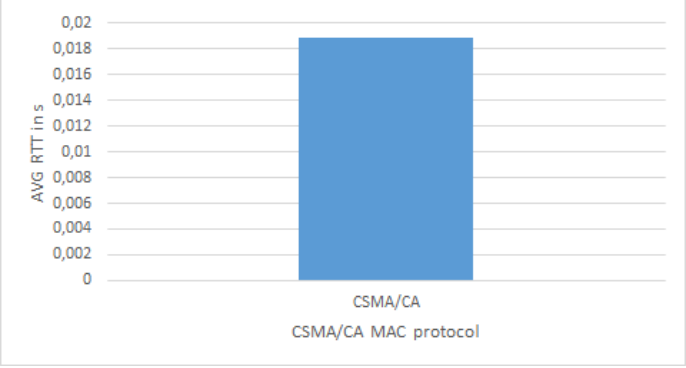

Fig. 2. Resulting RTT for CSMA/CA

- Comparing ContikiMAC RTT results with 2 different channel check rates
  With an average RTT of 0.2198 and 0.3417 for ContikiMAC 8Hz and ContikiMAC 16Hz respectively we conclude that the RTT is lower when both the client and the server run a channel check rate of 8Hz.
- Comparing ContikiMAC and CSMA/CA RTT results
  With an average RTT of 0.0188s for CSMA/CA, CSMA/CA has a lower RTT compared to ContikiMAC with any channel check rate.

B. Reliability

In order to investigate reliability we are going to see the number of packets that are sent from client and then we'll see how many of them are received at the server. We run this for 5 minutes. A 5 minutes running experiment seems to be enough to determine if a communication is reliable or not considering that in our case there are no other external or internal factors affecting the communication. For ContikiMAC 61 packets sent in 5 minutes with the sending interval of 5 seconds are received in 5 minutes 0.5 seconds. We do the same experiment 61 packets in 5 minutes but this time with null rdc_driver and again all the packets are received in 5 minutes 0.5 seconds meaning that every packet we sent is received.

C. Channel check rate with MAC protocols

We calculated the Duty cycle for every MAC protocols with various sending intervals (1s, 2s, 5s, 20s and 30s). Here various sending intervals were chosen in order to investigate what happens to the Duty Cycle when the load of the network increases. With ContikiMAC a channel check rate of 16Hz has more average duty cycle than a channel check rate of 8Hz. Since the Channel Check rate parameter is only used by the RDC Layer of the ContikiMAC MAC protocol, there is no need to investigate the influence of channel check rate on duty cycle for CSMA/CA MAC protocol. Obviously the duty cycle increases drastically when the MAC protocol changes from ContikiMAC to CSMA/CA with an average RDC of 1.745 for ContikiMAC and an average of 99.406 for CSMA/CA. We also noticed with both MAC protocols and both channel check rates that the lower the sending interval the more RDC.

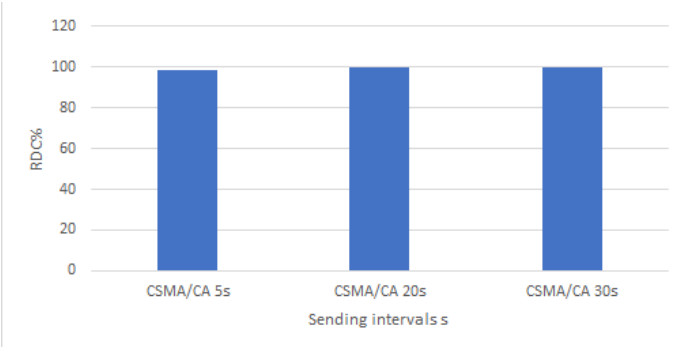

Fig. 3. Resulting RDC% from various Sending Intervals using CSMA/CA

D. The effect of the MAC protocols on the duty cycle

In this section all the RDCs are obtained at the client. In this experiment we used various sending intervals (5s, 20s and

30s) with each MAC protocol and each channel check rate. Because NullRDC is a ”null” RDC layer that never switches the radio off, the duty cycle for CSMA/CA will obviously be more than the Duty cycle of ContikiMAC. With ContikiMAC for 8Hz and 16Hz, we have a duty cycle of 1.187 and 1.359 respectively. With ContikiMAC we notice a difference in duty cycle between using a channel check rate of 8Hz and a channel check rate of 16Hz. The channel check rate of 16Hz results in a higher duty cycle in ContikiMAC than with a channel check rate of 8Hz. Once again using a lower sending interval results in a higher RDC with both protocols.

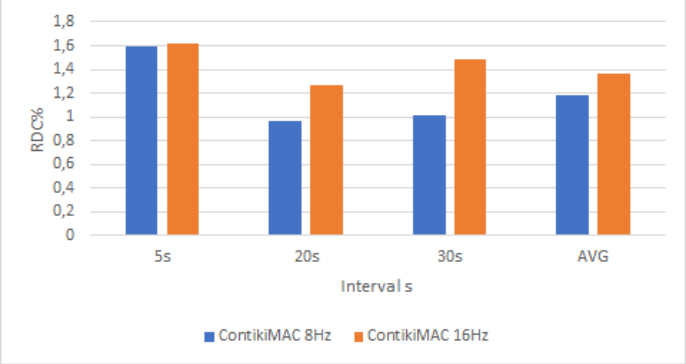

Fig. 4. Resulting RDC% from various Channel Check Rates and Sending Intervals

E. The effect of the payload length on the duty cycle

The purpose of this section is to measure the effect of the payload length(36 bytes and 151 bytes) on the RDC. To that effect, various sending intervals (1s, 2s, 5s, 20s, 30s) were used. 36 is the default payload length and 151 is just below the default buffer size so it should be able to fit in.

1) With 151 bytes: A channel check rate of 16Hz results in a higher duty cycle than a channel check rate of 8Hz with an average of 2.059 and 1.639 respectively.
2) With 36 bytes: A channel check rate of 16Hz results in a higher duty cycle than a channel check rate of 8Hz with an average of 1.727 and 1.445 respectively.

As you can notice from the numbers above a payload size of 151 bytes results in a higher duty cycle than a payload size of 36 bytes for both channel check rates.

IV. Conclusion

From the experiments we conclude that ContikiMAC with the channel check rate of 8Hz is more advisable over ContikiMAC with a channel check rate of 16Hz for two main reasons. First, ContikiMAC 8Hz has a lower RDC than ContikiMAC 16Hz. Second, ContikiMAC 8Hz has a lower RTT than ContikiMAC with a channel check rate of 16Hz.

Comparing ConitikiMAC to CSMA/CA, ContikiMAC is recommended as it consumes less energy. On the other hand CSMA/CA might considered as a suitable MAC protocol for the following reasons. First, it is easy to implement. Second, CSMA/CA has a lower RTT than ContikiMAC. Even

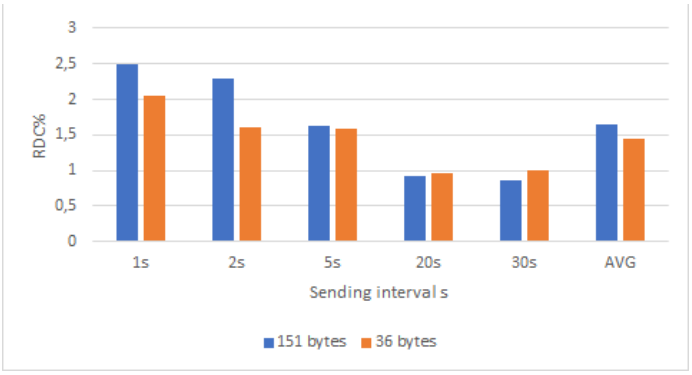

Fig. 5. Resulting RDC from 151 bytes and 36 bytes with a Channel Check Rate of 8Hz

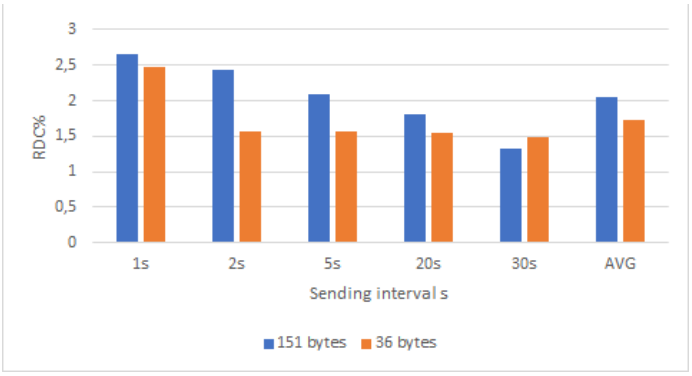

Fig. 6. Resulting RDC from 151 bytes and 36 bytes with a Channel Check Rate of 16Hz

though CSMA/CA might improve collision avoidance, it is not advised in high density networks.

For the purpose of saving energy it is better to use a higher sending interval. Although, depending on the purpose of the experiment one might choose to use a lower sending interval if the speed of the communication is more important than saving energy.

References

[1] https://github.com/contiki-os/contiki/wiki/Radio-duty-cycling
[2] http://www.contiki-os.org/start.html
[3] http://zolertia.sourceforge.net/wiki/
[4] https://zolertia-contiki.wikispaces.com/Using+Instant+Contiki
[5] https://github.com/Zolertia/Resources/wiki/ The-Z1-mote
[6] http://anrg.usc.edu/contiki/index.php/RPL_UDP