Chip Monitor

# Overview

The chip monitoring service allows the application to fetch the changes in the values of the device core temperature, Voltage of VBAT, the external ADC and the estimated current consumption of Talaria TWO device.

The power measurement performed on Talaria TWO device is an estimated value based on the state that the Talaria TWO device is in and the duration that the Talaria TWO device stays in that particular state. The Talaria TWO device can be in any of the four possible states:

1. Suspend mode
2. CPU mode
3. Rx mode
4. Tx mode

Hence:

|  |
| --- |
| estimated power consumption of T2 = Power consumption of the device in a particular state x Duration for which the device stays in that state |

Since the power consumption of Talaria TWO in a particular state is a known value, it is possible to calculate the approximate power consumption value by measuring the duration for which the device stays in a particular state and multiplying it with the known power consumption value of the device in that state.

This application note provides details on using the chip monitoring services for the changes in the values of core temperature, voltage of VBAT, the external ADC and the estimated current consumption of Talaria TWO modules using:

1. A standalone application running on the Talaria TWO module which provides the change in the values of the services periodically when varied by a threshold.
2. An application running on Talaria TWO that responds to the requests from an application running on the host. The python script – talaria\_cli.py present in the sdk/scripts directory will be used as the application running on the host PC.

# Chip Monitoring Services Standalone Application

The application running on Talaria TWO can register to any (or all) of the chip monitoring services i.e., the core temperature of the chip, voltage of VBAT, the external ADC to monitor the change in the values and the estimated current consumed by the T2 module by a threshold value set. The registered service callback will get invoked when there is a variation of the threshold value set.

The sample application described in this application note will subscribe to all the four chip monitoring services.

# Building

To build the sample application, execute the following commands:

|  |
| --- |
| cd examples/chip\_monitor  make |

The make command should generate the chip\_monitor.elf in the out directory.

# Chip monitor APIs

## chip\_monitor\_power\_init()

Allocates energy reports for power measurements. Callers who want to subscribe on power measurement need to call this before calling chip\_monitor\_start().

## chip\_monitor\_start()

Starts subscription for changes on specified source.

To calculate the Talaria TWO current power consumption, it is required that chip\_monitor\_power\_init() is called prior to chip\_monitor\_start(). The time for a measurement is dictated by time between the measurements plus the possible time the chip is suspended.

## chip\_monitor\_stop()

Stops the subscription service.

# Sample Code Walkthrough

**chip\_monitor.c**

The chip monitor sample code allows the device to subscribe to services for measuring the estimated power consumption of Talaria TWO, changes in core temperature, voltage and ADC.

The sample application subscribes to all the four services and sets a threshold value of 1 to be able to print the information in the console if there is a change from the last value captured by a threshold of 1. The threshold value in the sample example is set to 1 for the purpose of demonstration. The threshold value can be set to any valid value.

The registered call back service\_fn\_clbk gets invoked for change in the values of the estimated power consumed, core temperature, VBAT and ADC by a value of 1. If the threshold value is set to 0, the call back gets invoked for the period specified by the minterval irrespective of the change in the measured value.

The main function starts with receiving the boot arguments for the following parameters:

1. minterval: Time interval in seconds for measuring the value of the subscribed chip monitor service.
2. measure\_count: Total number of estimated power consumption measurements to be made.

|  |
| --- |
| uint32\_t minterval = os\_get\_boot\_arg\_int("minterval", 10);  uint32\_t li = os\_get\_boot\_arg\_int("li", 15); |

chip\_monitor\_power\_init() allocates energy reports for power measurements. This API needs to be called before calling chip\_monitor\_start() for the estimated power measurements on Talaria TWO.

|  |
| --- |
| chip\_monitor\_power\_init();  ctms\_t0 = os\_systime64(); |

Depending on the of bootargs value (0:Service not subscribed; 1:Service subscribed) for the following services, a service is either subscribed or not subscribed. If a service is subscribed, the change in the measured value of that service is notified through the registered callback.

1. adc\_service
2. vbat\_service
3. temperature\_service
4. power\_service

By default ,all the services are subscribed.

|  |
| --- |
| if(os\_get\_boot\_arg\_int("adc\_service", 1) == 1)  {  os\_printf("Enabling ADC service\r\n");  enable\_chip\_monitor\_service(CHIP\_MSOURCE\_ADC, true);  }  if(os\_get\_boot\_arg\_int("vbat\_service", 1) == 1)  {  os\_printf("Enabling VBAT service\r\n");  enable\_chip\_monitor\_service(CHIP\_MSOURCE\_VBAT, true);  }  if(os\_get\_boot\_arg\_int("temperature\_service", 1) == 1)  {  os\_printf("Enabling Temperature service\r\n");  enable\_chip\_monitor\_service(CHIP\_MSOURCE\_CORE\_TEMP, true);  } |

The enable\_chip\_monitor\_service() API subscribes or unsubscribes the given service that is provided as an argument.

The power service will also be enabled by default if the Wi-Fi connection is successful. The chip\_monitor\_wifi\_conn() establishes a Wi-Fi connection and sets the power management configuration for the Wi-Fi interface and returns 0 upon successful Wi-Fi connection. This sample application measures the power consumption of Talaria TWO (estimated power consumption) with idle Wi-Fi connection.

First, The Wi-Fi network interface is created using wcm\_create()and wcm\_notify\_enable() enables the callback function.

|  |
| --- |
| wcm\_handle = wcm\_create(NULL);  wcm\_notify\_enable(wcm\_handle, cm\_wcm\_notify\_cb, NULL); |

network\_profile structure adds a network profile to WCM. np\_conf\_path pointer variable contains the path of the network profile file (a JSON file) present in Talaria TWO filesystem. network\_profile\_new\_from\_file\_system()API builds a network profile from the network profile file in the filesystem and the path to this file is provided by the np\_conf\_path variable that receives the path to the network profile file through a boot argument.

|  |
| --- |
| const char \*np\_conf\_path = os\_get\_boot\_arg\_str("np\_conf\_path")?: NULL;  struct network\_profile \*profile;  if (np\_conf\_path != NULL)  {  rval = network\_profile\_new\_from\_file\_system(&profile, np\_conf\_path);  } |

wcm\_add\_network\_profile() adds the network profile to WCM and wcm\_auto\_connect() starts the auto connection with Wi-Fi network.

|  |
| --- |
| rval = wcm\_add\_network\_profile(wcm\_handle, profile);  if (rval < 0) {  pr\_err("could not associate network profile to wcm %d\n", rval);  return 0; }  if(wcm\_auto\_connect(my\_wcm\_handle, 1) == 0) |

After the Wi-Fi connection is successful, the system is triggered to enter into the suspend state depending upon the value of the boot argument – suspend provided.

1. suspend =1; Triggers the Talaria TWO device to enter into suspend state.
2. suspend =0; Talaria TWO device does not enter into suspend state.

By default, the system is not triggered to enter the suspend state. However, for the power measurements in idle mode, Talaria TWO device will be triggered into suspend state through the boot argument.

The os\_suspend\_enable()API triggers Talaria TWO into suspend state.

|  |
| --- |
| if (os\_get\_boot\_arg\_int("suspend", 0) != 0)  os\_suspend\_enable(); |

Similarly, the gratuitous ARP can either be disabled or enabled based on the value provided through the boot argument – arp\_enable.

1. arp\_enable = 0; Gratious ARP is disabled.
2. arp\_enable =1; Gratious ARP is not disabled. This is the default setting.

The reception of multicast frames is disabled and the power management configuration for the Wi-Fi interface is set. The li value for the power management configuration is received as a boot argument. By default, the value of listen interval is set to 10, the traffic time out to 12, and the power management flags to 0.

Refer to the API Reference Guide (*sdk\_x.y/doc/api\_reference\_guide/T2-RM001-Vxy-Talaria\_TWO\_SDK\_API\_Reference\_Guide.pdf*) for more information on the description of all these APIs.

**Note**: x and y in sdk\_x.y refer to the SDK release version.

|  |
| --- |
| pm\_mask |= WIFI\_PM\_STA\_RX\_NAP | WIFI\_PM\_STA\_ONLY\_BROADCAST | WIFI\_PM\_TX\_PS | WIFI\_PM\_MCAST\_DONT\_CARE | WIFI\_PM\_DTIM\_ONLY;  wcm\_pm\_config(wcm\_handle, li, 12, pm\_mask); |

chip\_monitor\_wifi\_conn() returns 0 upon successful Wi-Fi connection and setting the power management configuration. Now, execution of the main thread is suspended for a period of 15 seconds for the device to stabilize before starting the power measurements. The power measurement service will be enabled by default since the default value of the boot argument - power\_service is set to 1. if the value of the boot argument – power\_service is set to 0. Then the power measurement service will be disabled.

The enable\_chip\_monitor\_service() function enables/disables a chip monitor service and takes the following boot arguments:

1. threshold: A threshold value to trigger the registered call back for a subscribed service. Any change in the measured value by the threshold value will trigger the call back and prints the measured value
2. curr\_value: It is the last sample value measured. This value gets updated every time the measurement is made.

Depending on the arguments provided to the function, enable\_chip\_monitor\_service(enum chip\_monitor\_source service, bool enable), the selected chip monitor service is either enabled or disabled. The chip\_monitor\_start() API registers a call back function- service\_fn\_clbk() for a given service.

|  |
| --- |
| uint32\_t threshold = os\_get\_boot\_arg\_int("threshold", 0);  uint32\_t curr\_value = os\_get\_boot\_arg\_int("curr\_value", 0);  struct chip\_monitor\_serv \*temp\_serv = NULL , \*vbat\_serv= NULL, \*adc\_serv= NULL, \*power\_serv = NULL;  switch(service){  case CHIP\_MSOURCE\_CORE\_TEMP:  if(enable == false){  chip\_monitor\_stop(temp\_serv);  break;  }  temp\_serv = os\_zalloc(sizeof \*temp\_serv);  assert(temp\_serv != NULL);  chip\_monitor\_start(temp\_serv, CHIP\_MSOURCE\_CORE\_TEMP, curr\_value, threshold, minterval, (chip\_mon\_notify\_t)service\_fn\_clbk);  break;  case CHIP\_MSOURCE\_VBAT:  if(enable == false){  chip\_monitor\_stop(vbat\_serv);  break;  }  vbat\_serv = os\_zalloc(sizeof \*vbat\_serv);  assert(vbat\_serv != NULL);  chip\_monitor\_start(vbat\_serv, CHIP\_MSOURCE\_VBAT, curr\_value, threshold, minterval, (chip\_mon\_notify\_t)service\_fn\_clbk);  break;  case CHIP\_MSOURCE\_ADC:  if(enable == false){  chip\_monitor\_stop(adc\_serv);  break;  }  adc\_serv = os\_zalloc(sizeof \*adc\_serv);  assert(adc\_serv != NULL);  chip\_monitor\_start(adc\_serv, CHIP\_MSOURCE\_ADC, curr\_value, threshold, minterval, (chip\_mon\_notify\_t)service\_fn\_clbk);  break;  case CHIP\_MSOURCE\_POWER:  if(enable == false){  chip\_monitor\_stop(power\_serv);  break;  }  power\_serv = os\_zalloc(sizeof \*power\_serv);  assert(power\_serv != NULL);  chip\_monitor\_start(power\_serv, CHIP\_MSOURCE\_POWER, curr\_value, threshold, minterval, (chip\_mon\_notify\_t)service\_fn\_clbk);  break; |

The service\_fn\_clbk()gets invoked if there is change by the threshold in values measured for a given service i.e., core temperature, voltage, ADCs or the estimated power measured.

|  |
| --- |
| static void service\_fn\_clbk(struct chip\_monitor\_serv\* serv, enum chip\_monitor\_source source, uint32\_t last\_value){  ….  ….  switch(source){  case CHIP\_MSOURCE\_CORE\_TEMP:  os\_printf("Time:%llu sec Chip Core Temp: %u C\n", now, last\_value);  break;  case CHIP\_MSOURCE\_VBAT:  os\_printf("Time:%llu sec Chip Vbat: %u dV\n", now, last\_value);  break;  case CHIP\_MSOURCE\_ADC:  os\_printf("Time:%llu sec Chip ADCin: %u \n", now, last\_value);  break;  …  … |

For the power measurement service, the average of the measured current values has to be considered. Hence, the sum of the values measured/the number of samples is also displayed on the console. The total number of samples to be captured is specified through the boot argument measure\_count.

|  |
| --- |
| case CHIP\_MSOURCE\_POWER:  os\_printf("Chip mscource power\r\n");  if (start\_count\_down == 0) {  count++;  total += last\_value;  os\_printf("Average Current:%duA\n", total/count);  if(count > measure\_count){  os\_printf("Completed %d measurements\r\n",measure\_count);  chip\_monitor\_stop\_power\_measurement();  } |

After capturing the total number of power measurements, the chip\_monitor\_stop\_power\_measurement() is called to stop the measurements after displaying the average current measured.

The chip\_monitor\_stop\_power\_measurement()function stops the power measurement by calling the enable\_chip\_monitor\_service(CHIP\_MSOURCE\_POWER, false) function and shutting down and cleaning up a WCM interface.

|  |
| --- |
| enable\_chip\_monitor\_service(CHIP\_MSOURCE\_POWER, false);  wcm\_remove\_network(wcm\_handle, ssid, NULL);  wcm\_destroy(wcm\_handle);  wcm\_handle = NULL; |

The print\_wifi\_config() prints the Wi-Fi configuration parameters.

|  |
| --- |
| void print\_wifi\_config()  { uint32\_t current\_li;  uint32\_t current\_traffic\_tmo;  uint32\_t current\_pm\_flags;  int current\_sleep\_period;  wcm\_pm\_config\_get(wcm\_handle,&current\_li, &current\_traffic\_tmo,&current\_pm\_flags);  wcm\_pm\_get\_sleep\_period(wcm\_handle, &current\_sleep\_period);  os\_printf("\n\*\*\*\*\*\*\n");  os\_printf("li: %d\n", current\_li);  os\_printf("traffic\_tmo: %d\n", current\_traffic\_tmo);  os\_printf("pm\_flags: 0x%x\n", current\_pm\_flags);  os\_printf("sleep\_period: %d ms\n", current\_sleep\_period/1000);  os\_printf("\*\*\*\*\*\*\n"); } |

# Running the Application

### Programming Talaria TWO using the Download Tool

Program chip\_monitor.elf *(sdk\_x.y\examples\chip\_monitor\bin)* using the Download tool:

1. Launch the Download tool provided with InnoPhase Talaria TWO SDK.
2. In the GUI window:
   1. Boot Target: Select the appropriate EVK from the drop-down
   2. ELF Input: Load the chip\_monitor.elf by clicking on Select ELF File.
   3. AP Options: Provide the SSID and Passphrase under AP Options to connect to an Access Point.
   4. Boot Arguments: Pass the following boot arguments to enable power service and disable all other services:

**Enabling only Power\_service**:

|  |
| --- |
| adc\_service=0,vbat\_service=0,temperature\_service=0,power\_service=1,suspend=1,arp\_enable=0,li=10,minterval=10,measure\_count=15 |

* 1. Programming: Prog RAM or Prog Flash as per requirement.

For more details on using the Download tool, refer to the document: UG\_Download\_Tool.pdf (path: *sdk\_x.y/pc\_tools/Download\_Tool/doc*).

**Note**: x and y refer to the SDK release version. For example: sdk\_2.5/doc.

Expected Output:

On flashing the application using the Download Tool, the console output is as follows. The application displays the current measured and the Average current in µA.

|  |
| --- |
| UART:SNWWWWWAEBuild $Id: git-ba65998b7 $  adc\_service=0 vbat\_service=0 temperature\_service=0 power\_service=1 suspend=1 arp\_enable=0 li=10 minterval=10 measure\_count=15 np\_conf\_path=/data/nprofile.json ssid=InnoPhase passphrase=43083191  === CMS - Chip Monitor Service ===  addr e0:69:3a:00:13:90  disabling gratitous arp  Wait for 15 seconds for the device to stabilize before starting power measurement service  [2.098,936] CONNECT:00:5f:67:cd:c5:a6 Channel:11 rssi:-49 dBm  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_UP  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS  [4.087,115] MYIP 192.168.0.104  [4.087,164] IPv6 [fe80::e269:3aff:fe00:1390]-link  Enabling power consumption service  == Calibrating ==  Chip mscource power  Waiting to start power measurement!  == Calibrating ==  Chip mscource power  Waiting to start power measurement!  == Calibrating ==  Chip mscource power  Time:47 sec;Power measurement-1:879uA  Average Current:879uA  == Calibrating ==  Chip mscource power  .  .  .  Time:182 sec;Power measurement-14:545uA  Average Current:892uA  == Calibrating ==  Chip mscource power  Time:192 sec;Power measurement-15:580uA  Average Current:871uA  Completed 15 measurements  Stopping power measurements  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_LINK\_DOWN  [194.439,425] DISCONNECTED  wcm\_notify\_cb to App Layer - WCM\_NOTIFY\_MSG\_ADDRESS |

**Enabling adc, vbat and temp services**:

In the GUI, Boot Arguments: Pass the following boot arguments to disable power service and enable all other services:

Add the following boot arguments to disable power service and enable all other services:

|  |
| --- |
| adc\_service=1,vbat\_service=1,temperature\_service=1,power\_service=0 |

Expected Output:

On flashing the application using the Download Tool, the console output is as follows.

The application displays the raw value that varies according to the input provided to the ADC pin. Apart from the raw value measured out of the ADC pin, the internal temperature in °C and VBAT measured from source in mV are also displayed.

The following is the expected output:

|  |
| --- |
| UART:NWWWWWAE4 DWT comparators, range 0x8000  Build $Id: git-7e2fd6a94 $  app=gordon  flash: Gordon ready!  UART:NWWWWWAE4 DWT comparators, range 0x8000  Build $Id: git-7e2fd6a94 $  app=gordon  flash: Gordon ready!  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790da1-b-7  ROM yoda-h0-rom-16-0-gd5a8e586  FLASH:PNWWWWWAEBuild $Id: git-65f6c1f46 $  adc\_service=1 vbat\_service=1 temperature\_service=1 power\_service=0  === CMS - Chip Monitor Service ===  Enabling ADC service  Enabling VBAT service  Enabling Temperature service  == Calibrating ==  Time:0 sec Chip ADCin: 548  == Calibrating ==  Time:0 sec Chip Vbat: 3314 dV  == Calibrating ==  Time:0 sec Chip Core Temp: 23 C  ….  ….  == Calibrating ==  Time:20 sec Chip Vbat: 3314 dV  == Calibrating ==  Time:30 sec Chip ADCin: 112 |

# Chip Monitoring Services - Hosted Application

The application running on an external host can communicate with Talaria TWO and fetch the information on the change in the values of the core temperature, External ADC, VBAT and the estimated current measured values. The procedure to fetch this information is described in the following sections.

## Application Running on External Host

The talaria\_cli.py script running on the host (PC) will fetch required information from stw application running on the Talaria TWO to fetch the required information. The following are the steps to be executed:

1. Flash the apps/stw.elf onto Talaria TWO (refer steps in section 9.1.1 for more details on flashing the elf onto Talaria TWO).

Console output:

|  |
| --- |
| WWWWAE4 DWT comparators, range 0x8000  Build $Id: git-7e2fd6a94 $  App=gordon  Flash: Gordon ready!  Y-BOOT 208ef13 2019-07-22 12:26:54 -0500 790dal- b- 7  ROM yoda- h0- rom- 16- 0- gd5a8e586  FLASH: PNWWWWWAEBuild $Id: git – d468c7b54 $  Serial – to – Wireless: Ready |

1. Execute the following in Talaria TWO command line:

|  |
| --- |
| python3 ./script/talaria\_cli.py /dev/ttyUSB2 |

Console output:

Text

Description automatically generated

Figure : Talaria TWO - Command Line Output

1. In the Talaria CLI, create a WCM handle and connect to a network.

|  |
| --- |
| [talaria-2]$ create  [talaria-2]$ add\_network <SSID> -p <Password>  [talaria-2]$ auto\_connect |

The execution of these commands and the output is as follows:

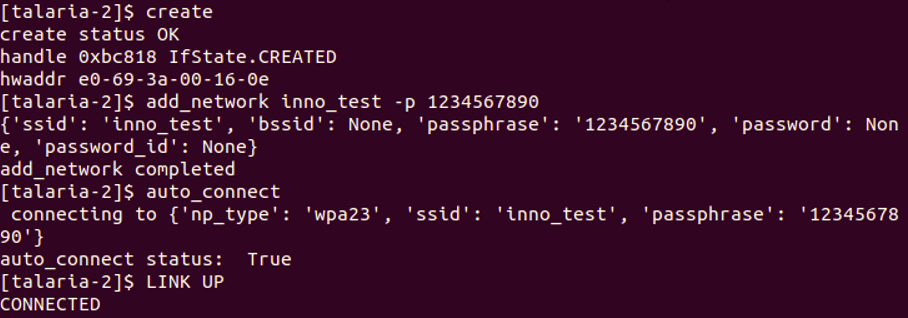


Figure : Creating a WCM handle and connecting to a network

1. Enable device suspend

|  |
| --- |
| [talaria-2]$ suspend 1 |

1. Execute chip\_monitor to fetch data from Talaria TWO:
   1. To initiate chip\_monitor, execute the following command:

|  |
| --- |
| [talaria-2]$ chip\_monitor -ip |

* 1. To start chip\_monitor, execute the following command:

|  |
| --- |
| [talaria-2]$ chip\_monitor --start -s<0/1/2/3> -i<interval> |

Console outputs for the sources – Estimated Current measurement, ADC, Internal temperature and VBAT are as follows:

1. Estimated current measurement:

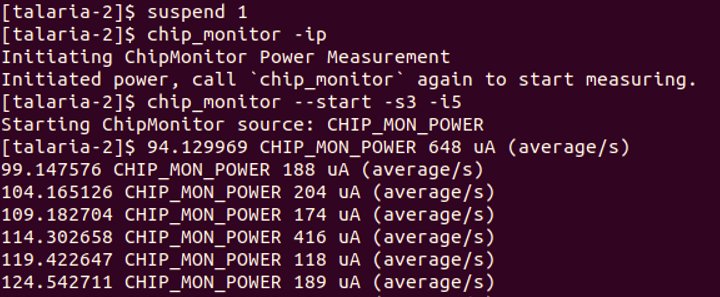


Figure : Estimated current measurement

1. ADC:

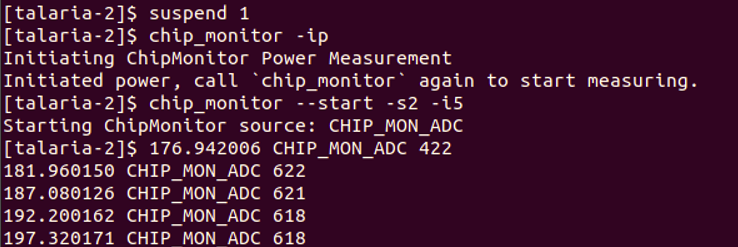


Figure 4: ADC

1. Internal Temperature:

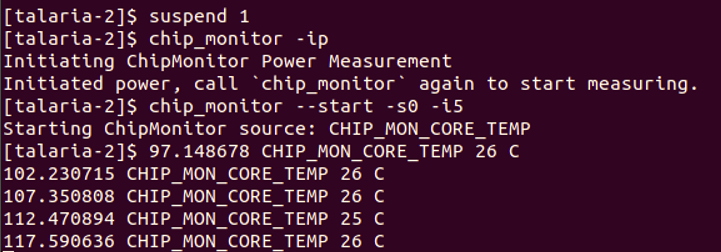


Figure 5: Internal temperature

1. VBAT:

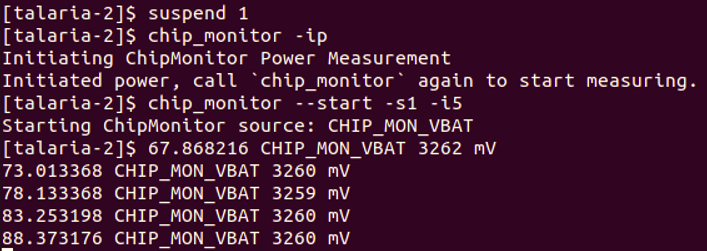


Figure 6: VBAT

## Measured Current Values for multiple DTIM Intervals

For reference, the estimated power consumption values measured using the chip\_monitor app for various DTIM intervals (clean environment current numbers) are provided in Table 1.

|  |  |
| --- | --- |
| **DTIM** | **Current consumption (µA)** |
| 1 | ~395 |
| 3 | ~140 |
| 10 | ~52 |
| 100 | ~22 |

Table : DTIM intervals