EWSN 2019 Dependability Competition

Logistics Information, rev. 1

Carlo Alberto Boano and **Markus Schuß**
Institut für Technische Informatik
Graz University of Technology, Austria

01.10.2018

4rd EWSN Dependability Competition

- Following the success of the past three editions, the International Conference on Embedded Wireless Systems and Networks (EWSN) hosts also this year a dependability competition comparing the performance of IoT communication protocols in harsh RF environments
  - 1st edition (2016): Graz, Austria [link]
  - 2nd edition (2017): Uppsala, Sweden [link]
  - 3rd edition (2018): Madrid, Spain [link]
  - 4th edition (2019): Beijing, China [link]

Format of 2019 Edition

- Remote preparation phase
  - As for the 2018 edition, a testbed facility will be available to all contestants
  - Steps necessary to obtain access to the testbed facility
    1. Competing teams accept the "terms and conditions" for the use of the testbed facility (see competition's blog for more info)
       → Team members agree to use the testbed facility exclusively for research purposes, and not to gain access to or disrupt any service or network
       → A signed copy of the terms and conditions is sent to the organizers
    2. At least one member of each competing team registers to EWSN'19 (http://ewsn2019.thss.tsinghua.edu.cn/registration-info.html)
    3. A single username and password is provided to all team members
  - Roughly two months preparation time before submitting the binary `ihex` file(s) to be used for the final evaluation
    → Teams may still withdraw from the competition before the final evaluation (but the competition abstract will not be published in the EWSN proceedings)

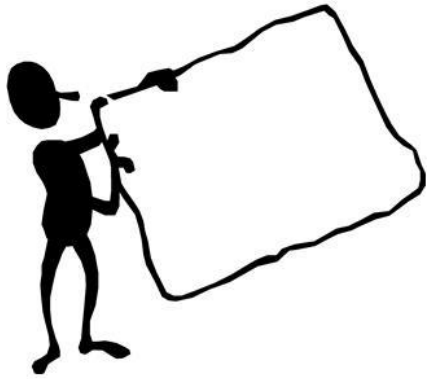
Format of 2019 Edition

- Evaluation phase
  - The binary file(s) provided at the end of the preparation phase will be used to extensively benchmark the performance of the competing solutions
    - → Only one binary file per category for each team!
    - → The evaluation phase will be run using settings similar to the ones provided during the preparation phase
- Submission of camera-ready abstracts
  - Each team participating to the final evaluation can publish a two-page competition abstract in the EWSN proceedings
    - → The abstract must include (i) a description of the competing solution, as well as (ii) preliminary results and lessons learned from the preparation phase 

    - → Competition abstracts will appear in the ACM Digital Library
    - → The camera-ready deadline for the competition abstracts follows the same deadline of EWSN poster abstracts (expected deadline: December 31, 2018) 

Format of 2019 Edition

- EWSN'19 in Beijing
  - As for last year's edition, a poster session dedicated to the dependability competition will take place during the main EWSN conference
    → All teams participating in the final evaluation must present their solution in the poster session and will have the possibility to engage in lively discussions with the other conference attendees



  - The winners of each competition's category will be awarded in a dedicated plenary session during the main conference
    → After receiving the award, the winning teams must hold a plenary short presentation about their solution, followed by a discussion session

Winners of 2018 edition

Format of 2019 Edition

- Two competition categories 

  - Category 1: Data collection for condition monitoring
    - → Up to eight source nodes communicating to a single destination node over a multi-hop network (i.e., *multipoint-to-point* traffic)
  - Category 2: Dissemination of actuation commands
    - → Up to eight source nodes disseminating actuation commands to a specific set of destinations nodes in the network (i.e., *point-to-multipoint* traffic)
    - → Each source node is associated to at most eight destinations
    - → A destination node can receive messages from only a single source node
    - → A destination node cannot act as a source node at the same time
- Several testbeds layouts 

  - For each category, different testbed layouts will be available (i.e., different configurations / sets of source and destination nodes)
    - → During the first days of the preparation phase, only one layout will be available
    - → Additional layouts will be added in the following weeks

Format of 2019 Edition

- Competition scenario
  - Same hardware used in the previous editions
    - → Advanticsys MTM-5000-MSP sensor nodes (TelosB replicas)
    - → This allows contestants to test their protocol on other public testbeds as well
  - Use of different input parameters 

    - → The address of source and destination nodes, the amount of data to be transmitted, and the traffic load are no longer hardcoded this year
    - → To this end, the competition infrastructure will directly inject input parameters into the firmware under test by applying patches to the provided `ihex` binary
    - → Detailed information and code examples follow on later slides
  - Data to be exchanged 

    - → Raw sensor values of different length
    - → The length will be the same for all nodes involved in a given experiment and will be known beforehand
    - → Detailed information and code examples follow on later slides

Format of 2019 Edition

- Competition scenario
  - Sending and receiving data 

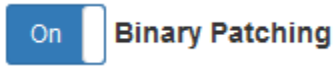
    → Nodes are connected using software I2C to an EEPROM
    → The testbed signals to each source node the availability of new data to be transmitted in the EEPROM by toggling a pre-defined GPIO pin
    → Once receiving a message, each destination node needs to write this data to the EEPROM and toggle a pre-defined GPIO pin accordingly
    → Detailed information and code examples follow on later slides
  - RF interference
    → Competing solutions will be tested both in the absence and in the presence of RF interference across the whole 2.4 GHz ISM band
    → No IEEE 802.15.4 channel(s) are guaranteed to be constantly interference-free
  - Frequency usage
    → Using frequencies below 2400 and above 2483.5 MHz is strictly forbidden (TI CC2420 radio allows operation between 2230 MHz and 2730 MHz)
    → No limitation about the usage of frequencies between 2400 and 2483.5 MHz
    → Frequency usage will be monitored during the evaluation phase: any detected violation will lead to a disqualification 

Preparing your Firmware

Binary Patching

- One of the main changes of this year's edition is the ability of the competition's testbed to directly inject a number of input parameters into the firmware under test
  - More information available on our CPSBench paper

- The testbed injects the following input parameters:
  - Traffic pattern (e.g., point-to-point traffic, multipoint-to-point traffic, …)
  - Node addresses of source and destination nodes
  - Traffic load
    - → Message length and location within the EEPROM
    - → Periodicity of messages (e.g., periodic, aperiodic, …)



Note that you can disable binary patching when queueing your experiment for testing purposes

Binary Patching

- Contestants need to use a pre-defined configuration struct
  - Provided in the `testbed.h` helper file
- An example on how this pre-defined configuration struct can be used is available on the competition's blog
  - Link: https://iti-testbed.tugraz.at/static/upload/competition_example_2019.zip
- This example contains:
  - `testbed.h` → Helper file containing the configuration struct and some helper functions to print the values injected by the testbed
  - `my2c.h / my2c.c` → Example implementation of the software I2C that can be used to read and write data to EEPROM
  - `Makefile` → Contains an example of how to configure the MSP430 linker's LDFLAGS correctly in the for binary patching

Binary Patching

- Contestants need to use a pre-defined configuration struct
  - Provided in the `testbed.h` helper file
- An example on how this pre-defined configuration struct can be used is available on the competition's blog
  - Link: https://iti-testbed.tugraz.at/static/upload/competition_example_2019.zip
- This example contains:
  - `i2c-test.c` → Example application carrying out point-to-point communication between two nodes using Contiki's Rime stack and also showing:
    - → How to print values passed by the testbed (`print_testbed_config`)
    - → Read and write data to EEPROM using software I2C
    - → Configure the GPIO pins and an interrupt service routine

Binary Patching

- Your firmware application needs to include a provided header file (`testbed.h`), which contains a well-known definition of the application's input parameters

```
10
11 #include "testbed.h"
12
```

- In order for the patching to work, these application input parameters need to be linked into a well-known address (`0xd400`) via the `Makefile`

```
17
18 LDFLAGS += -Wl,--section-start -Wl,.testbedConfigSection=0xd400
19
```

Binary Patching

- Your firmware application needs to contain an instance of the `config_t` structure (`cfg` in the example below)
  - `cfg` enables the testbed to change several settings such as traffic pattern and traffic load before execution
  - This avoids hardcoded values in your firmware

```
20
21 //In flash configuration struct, will be replaced by binary patching
22 volatile config_t __attribute__((section (".testbedConfigSection"))) cfg;
```

The attribute tells the MSP430 GCC to put the variable into a new section called `testbedConfigSection` in the resulting `elf` file (`project_name.sky` in Contiki)

The `config_t` type is defined in `testbed.h`

Ensures the compiler does not remove or “optimize” the variable

Input Parameters provided by the Testbed

- The `config_t` structure contains an array of different application input parameters (`pattern_t` struct)
- The `pattern_t` struct contains information about the traffic pattern and load:
  - Traffic pattern: `traffic_pattern`, `source_id[]`, `destination_id[]`
  - Traffic load: `msg_length`, `msg_offsetH`, `msg_offsetL`, `periodicity`, `aperiodic_upper_bound`, `aperiodic_lower_bound`

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Input Parameters provided by the Testbed

- `traffic_pattern` embeds info about the traffic pattern
  - Point-to-point (1), point-to-multipoint (2), multipoint-to-point (3), and multipoint-to-multipoint (4)
  - `traffic_pattern` is 0 if unused

During the EWSN 2019 competition, only the following patterns will be used:
`traffic_pattern=3` (category 1)
`traffic_pattern=2` (category 2)
`traffic_pattern=0` (category 1 & 2) – when unused, see example on a later slide

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Input Parameters provided by the Testbed

- `source_id[TB_NUMNODES] & destination_id[TB_NUMNODES]`
  - Embed info about the address of source and destination nodes
  - Each node is identified with an 8-bit unsigned short address
    → 8-bit unsigned short value (e.g., 100, 103, 200, …)

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;            // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];     // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                 // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;               // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;     // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;     // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

TB_NUMNODES=8

Input Parameters provided by the Testbed

- `source_id[TB_NUMNODES] & destination_id[TB_NUMNODES]`
  - Embed info about the address of source and destination nodes
  - Each node is identified with an 8-bit unsigned short address
    - → 8-bit unsigned short value (e.g., 100, 103, 200, …)
    - → Node ID is stored in flash
    - → We are reusing Contiki's Rime address stored in the 1 MB external flash
    - → Code example on how to read the node ID: https://iti-testbed.tugraz.at/static/upload/nodeid_from_flash.zip

Note that making use of the on-board DS2411 chip may lead to problems, as faulty nodes may be replaced during the competition (i.e., their DS2411 ID would change, whilst the node ID stored in flash would not)

```
47 void
48 node_id_restore(void)
49 {
50   unsigned char buf[4];
51   xmem_pread(buf, 4, NODE_ID_XMEM_OFFSET);
52   if(buf[0] == 0xad &&
53      buf[1] == 0xde) {
54     node_id = (buf[2] << 8) | buf[3];
55   } else {
56     node_id = 0;
57   }
58 }
```

Please inform us if you notice inconsistencies in the node IDs! (in principle, any team could accidentally overwrite the node ID in flash)

Input Parameters provided by the Testbed

- `traffic_pattern`
  - 0: indicates that this pattern is unused and can be ignored
  - 1: only the `source_id[0]` and `destination_id[0]` are used
  - 2: only the `source_id[0]` and all `destination_id[x]!=0` are used (`x = 0 … TB_NUMNODES-1`)
  - 3: all `destination_id[x]!=0` and the `source_id[0]` are used
  - 4: all `source_id[x]!=0` and `destination_id[x]!=0` are used

During the EWSN 2019 competition, `traffic_pattern=1` and `traffic_pattern=4` will not be used

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES];  // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Input Parameters provided by the Testbed

- msg_length
  - Number of bytes to be written and read from EEPROM (whenever an a falling edge occurs, see later slides)
  - Messages will be at most 64 bytes

```
65    //write messages up to the length in the struct
66    for(i=0;i<cfg.p[0].msg_length;i++){
67        my2c_write(ubuffer[i]);
68    }
```

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Input Parameters provided by the Testbed

- `msg_offsetH / msg_offsetL`
  - The high and low byte of the offset address in the EEPROM

```
57  //start i2c communication
58  my2c_start();
59  //write address on the bus
60  my2c_write(0x50<<1);
61  //write memory address (2 bytes)
62  my2c_write(cfg.p[0].msg_offsetH);
63  my2c_write(cfg.p[0].msg_offsetL);
```

- `0x50` is the 7-bit device address of EEPROM
- Lowest bit of I2C is used to indicate read/write (0 is write, 1 is read)
- Selecting the memory location is always a write operation (even when reading from it)

Selective Read in which we load the address of the memory location in the EEPROM where to read/write afterwards (16 bits), see https://www.onsemi.com/pub/Collateral/CAT24M01-D.PDF

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Input Parameters provided by the Testbed

- `periodicity`
  - Contains the period in milliseconds at which new messages are provided in EEPROM (signaled via a GPIO falling edge event, see later slides)
  - A value of 0 for periodicity indicates aperiodic traffic
    - → For aperiodic traffic, one can use the `aperiodic_upper_bound` and `aperiodic_lower_bound` bounds
    - → New messages will be provided at random times between these two bounds
    - → Both bounds are in milliseconds

```
10  typedef struct
11  {
12      uint8_t traffic_pattern;              // 0:unused, 1:p2p, 2:p2mp, 3:mp2p, 4: mp2mp
13      uint8_t source_id[TB_NUMNODES];       // Only source_id[0] is used for p2p/p2mp
14      uint8_t destination_id[TB_NUMNODES]; // Only destination_id[0] is used for p2p/mp2p
15      uint8_t msg_length;                   // Message length in bytes in/to EEPROM
16      uint8_t msg_offsetH;                  // Message offset in bytes in EEPROM (high byte)
17      uint8_t msg_offsetL;                  // Message offset in bytes in EEPROM (low byte)
18
19      uint32_t periodicity;                 // Period in ms (0 indicates aperiodic traffic)
20      uint32_t aperiodic_upper_bound;       // Upper bound for aperiodic traffic in ms
21      uint32_t aperiodic_lower_bound;       // Lower bound for aperiodic traffic in ms
22  } pattern_t;
```

Multiple Patterns

- The `config_t` structure contains an array of different application input parameters (`pattern_t` struct)
  - More than one `pattern_t` can be active at the same time, depending on the competition category
    → With `TB_NUMPATTERN = 8`, we have up to `p[0] … p[7]`
  - Category 1 will only use `p[0]`
    → Multipoint-to-point traffic between up to 8 sources and at most 1 destination node
    → `p[0].traffic_pattern = 3`
    → `p[1].traffic_pattern=0 … p[7].traffic_pattern=0` (unused)
  - Category 2 uses up to `TB_NUMPATTERN` patterns
    → Point-to-multipoint traffic between a fixed number (up to 8) source nodes and a set of destinations (each source can talk to up to 8 destinations)
    → Each pattern `p` uses either `traffic_pattern = 2` (point-to-multipoint) or `traffic_pattern = 0` (unused)
    → A destination receives messages from only one source node

```
24  typedef struct
25  {
26      pattern_t p[TB_NUMPATTERN];          // Up to TB_NUMPATTERN parallel configurations
27  } config_t;
28
```

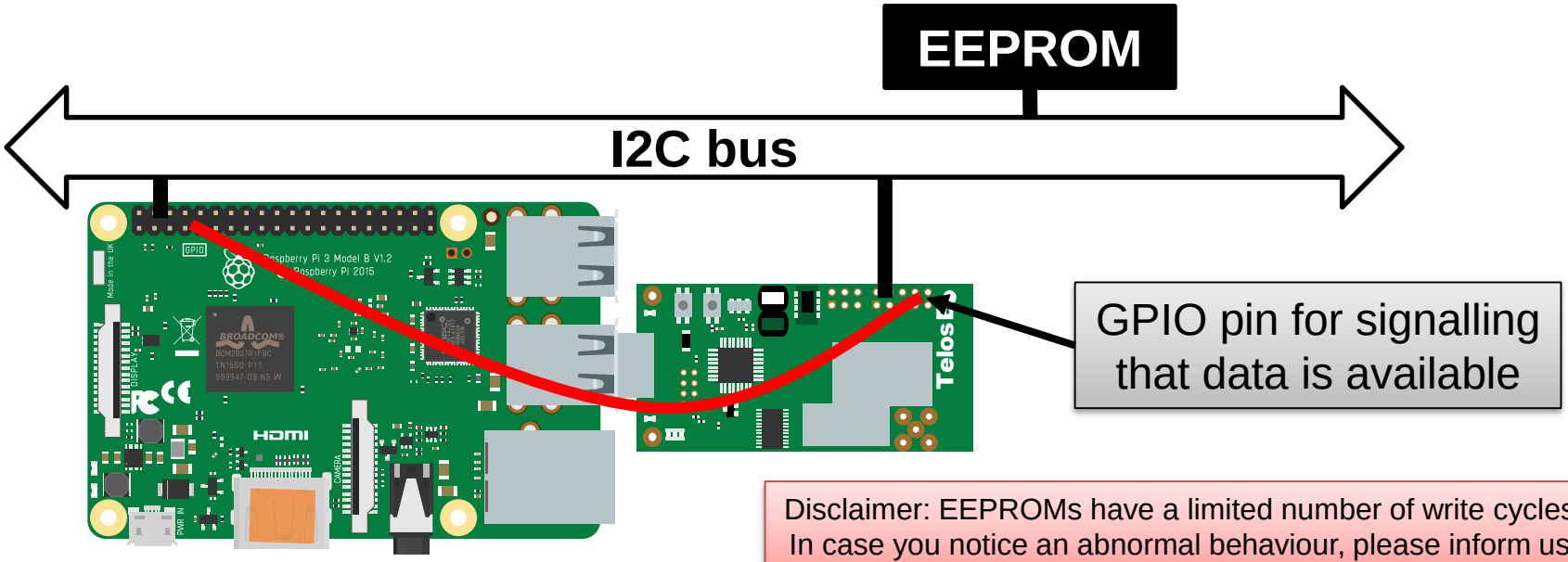
Printing Helper Function

- `print_testbed_config`
  - The `testbed.h` file also provides a function to print the input parameters injected by the testbed
  - You can use this function during the first experiments to make sure that your code works as expected
  - Make sure to enable the logging of serial output in the testbed (see later slide)

```
79  void
80  print_testbed_config(config_t* cfg)
81  {
82      printf("Testbed configuration:\n");
83      uint8_t i;
84      for(i=0;i<TB_NUMPATTERN;i++)
```

EEPROM Communication

- Messages to be sent over the network are available in an EEPROM connected via I2C bus
  - CAT24M01 EEPROM (located at address `0x50` on the bus), datasheet: https://www.onsemi.com/pub/Collateral/CAT24M01-D.PDF
  - The I2C bus is shared between the testbed's observer module (Raspberry Pi 3) and the target node (TelosB replica)
  - A pre-defined GPIO pin is used to signal that data is available



Disclaimer: EEPROMs have a limited number of write cycles. In case you notice an abnormal behaviour, please inform us!

EEPROM Communication

- Messages to be sent over the network are available in an EEPROM connected via I2C bus
  - A pre-defined GPIO pin is used to signal that data is available
    - → The GPIO used is on PORT2 and on the pin specified by `EVENT_PIN` in `testbed.h`
    - → In the provided code example (and during the competition, unless differently specified), `EVENT_PIN=6`, i.e., the pin used is `P2.6` (GIO3)

```
7  #define EVENT_PIN 6
```

    - → Same `EVENT_PIN` on PORT2 is used for both source and destination nodes

EEPROM Communication

- Signalling (source node)
  - If the node is a source, it needs to observe the `EVENT_PIN` on PORT2
  - Ideally interrupts are used to trigger a read operation (minimizes power and latency)
    - → Note that Contiki provides an ISR for this port by default, hence remove the ISR from `contiki/platform/sky/dev/button-sensor.c` first
    - → We provide an example code in the `i2c-test.c` showing how to register an ISR

```
28 ISR(PORT2, __eeprom_isr)
29 {
30   printf("ISR\n");
31   P2IFG&=~BV(EVENT_PIN);
32   process_post(&hello_world_process,FALLING_EDGE_EV,NULL);
33 }
```

Your process here

- → The use of Contiki and its events is optional!

EEPROM Communication

- Signalling (source node)
  - If the node is a source, it needs to observe the `EVENT_PIN` on PORT2
  - This pin is configured to be an input, triggering an interrupt on falling edges

```
111     //configure pin to input with interrupt
112     P2DIR&=~BV(EVENT_PIN);
113     P2SEL&=~BV(EVENT_PIN);
114     //P2IES&=~BV(EVENT_PIN);
115     P2IES|= BV(EVENT_PIN);
116     P2IFG&= ~BV(EVENT_PIN);
117     P2IE  |= BV(EVENT_PIN);
```

Call `__eeprom_isr` on a rising edge

Call `__eeprom_isr` on a falling edge

EEPROM Communication

- Signalling (destination node)
  - If the node is a destination, it needs to actuate the `EVENT_PIN` on PORT2
  - First configure the pin as output, then set to low

```
164     //configure pin to output
165     P2SEL &= ~BV(EVENT_PIN);
166     P2DIR |=  BV(EVENT_PIN);
167     P2OUT &= ~BV(EVENT_PIN);
```

  - Afterwards, the GPIO pin needs to be raised to indicate write operation, and toggled back to zero once the write operation has completed

```
50  //raise gpio pin to indicate write operation
51  P2OUT |= BV(EVENT_PIN);

        … EEPROM writing goes here …

68  //lower gpio pin to indicate finished write
69  P2OUT &= ~BV(EVENT_PIN);
```

EEPROM Communication

- Signalling (destination node)

- On the falling edge the testbed's observer module (Raspberry Pi 3) will try to read the EEPROM
- Make sure the I2C bus has been freed by this point!
- Latency measurement is carried out between falling edges

  - Afterwards, the GPIO pin needs to be raised to indicate write operation, and toggled back to zero once the write operation has completed

```
50  //raise gpio pin to indicate write operation
51  P2OUT |= BV(EVENT_PIN);

        … EEPROM writing goes here …

68  //lower gpio pin to indicate finished write
69  P2OUT &= ~BV(EVENT_PIN);
```

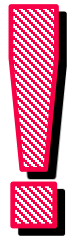
EEPROM Communication

- The I2C bus is shared between the observer module (testbed's RPi3) and the sensor node (TelosB replica)
  - I2C does not really support multi-master without arbitration
  - We use a single GPIO pin on PORT2 (`EVENT_PIN`) to indicate read or write operations
  - Keep in mind that read and write operations take time!

    → Do not write more than once every 20ms to give the observer module time to read the content

    → You can also watch the I2C clock (SCL) for activity to ensure data that has been read

    → The EEPROM will not respond for 5ms after a successful write operation! (check t_{WR} in the EEPROM's datasheet)



EEPROM Communication

my2c_start()
blocks the bus and signals start

my2c_write()
writes one byte on the bus

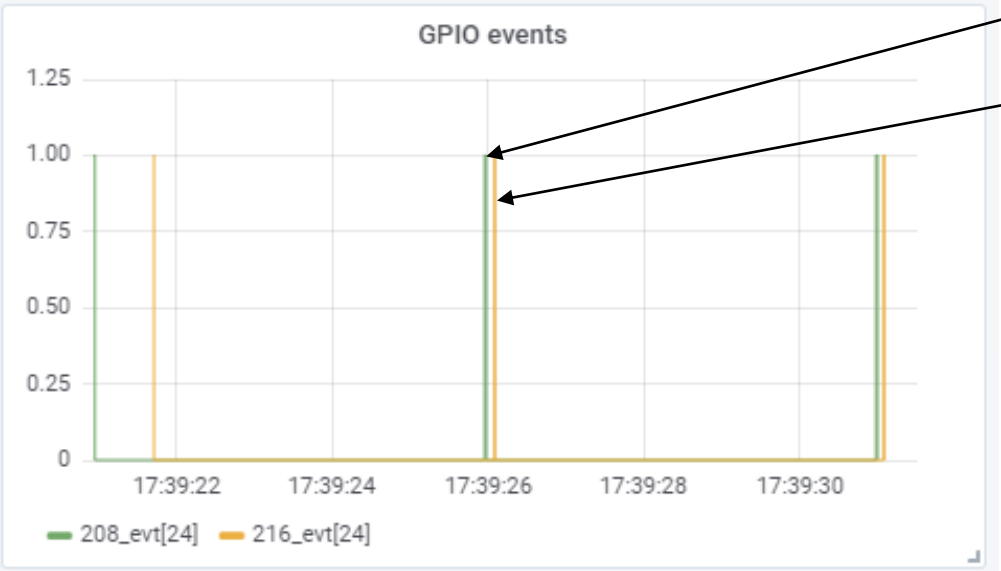
my2c_read(arg)
reads one byte, arg indicates last byte
(`arg =FALSE` → last byte)

my2c_stop()
releases the bus and signals stop

```
126       //i2c start
127       my2c_start();
128       //write address on the bus
129       my2c_write(0x50<<1);
130       //write memory address (2 bytes)
131       my2c_write(cfg.p[0].msg_offsetH);
132       my2c_write(cfg.p[0].msg_offsetL);
133       //disable the bus and wait a bit
134       my2c_stop();
135       clock_delay(100);
136       my2c_start();
137       //write address on the bus and set read bit
138       my2c_write((0x50<<1)|1);
139       //read back all the data, on the last byte indicate end
140       for(i=0;i<(cfg.p[0].msg_length);i++){
141         if(i==((cfg.p[0].msg_length)-1))
142       buffer[i]=my2c_read(false);
143     else
144           buffer[i]=my2c_read(true);
145       }
146       //i2c stop
147       my2c_stop();
```

EEPROM Communication

- Once a falling edge is detected on the source, the data can be read and transmitted to the destination
- Once the destination receives the message, it actuates the GPIO to high, reads the data, and lowers the GPIO



Source can read new data from EEPROM

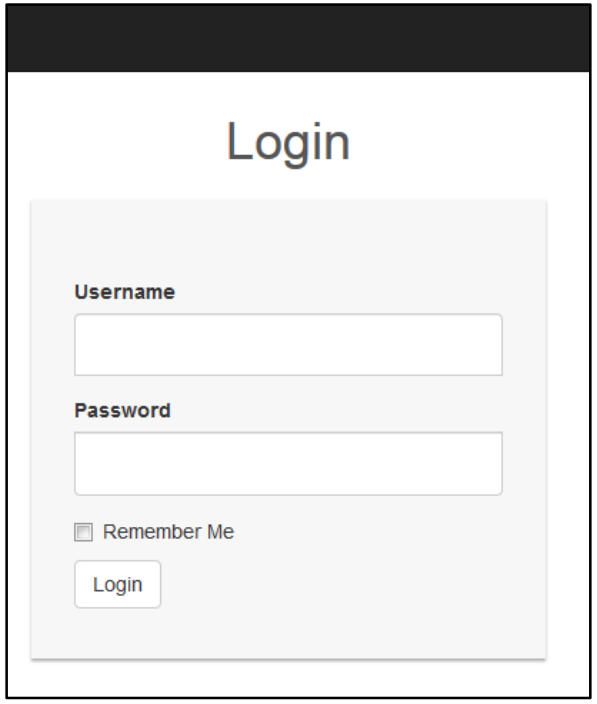
Destination has written data to EEPROM

Note that, the rising edge on the GPIO of a source node is not necessarily constant, as the EEPROM may skew the clock

Competition's Testbed Facility

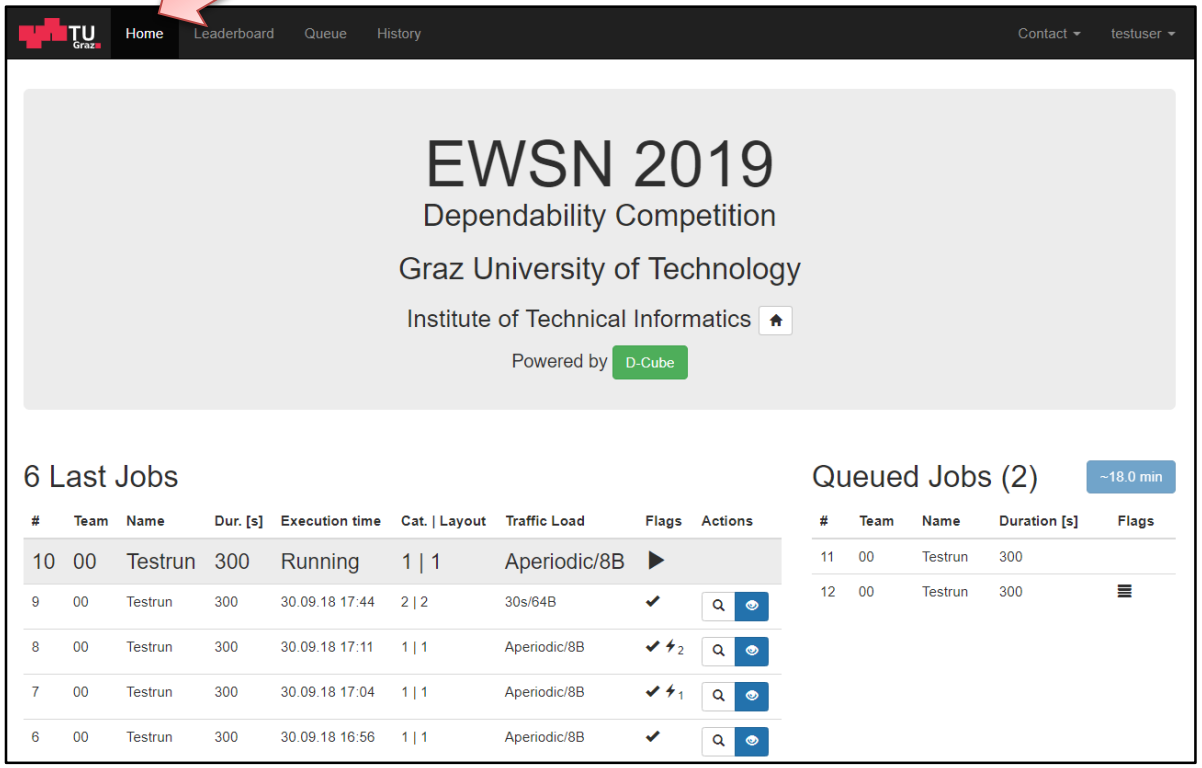
Competition's Testbed Facility

- The testbed facility is available at: https://iti-testbed.tugraz.at/
- Login credentials
  - Each team will receive the login credentials to access the testbed facility via e-mail as soon as:
    - → At least one team member has registered to EWSN 2019
    - → A signed scanned copy of the terms and conditions for the use of the competition's testbed has been sent to the organizers
    - → One username and password shared for the whole team



Competition's Testbed Facility

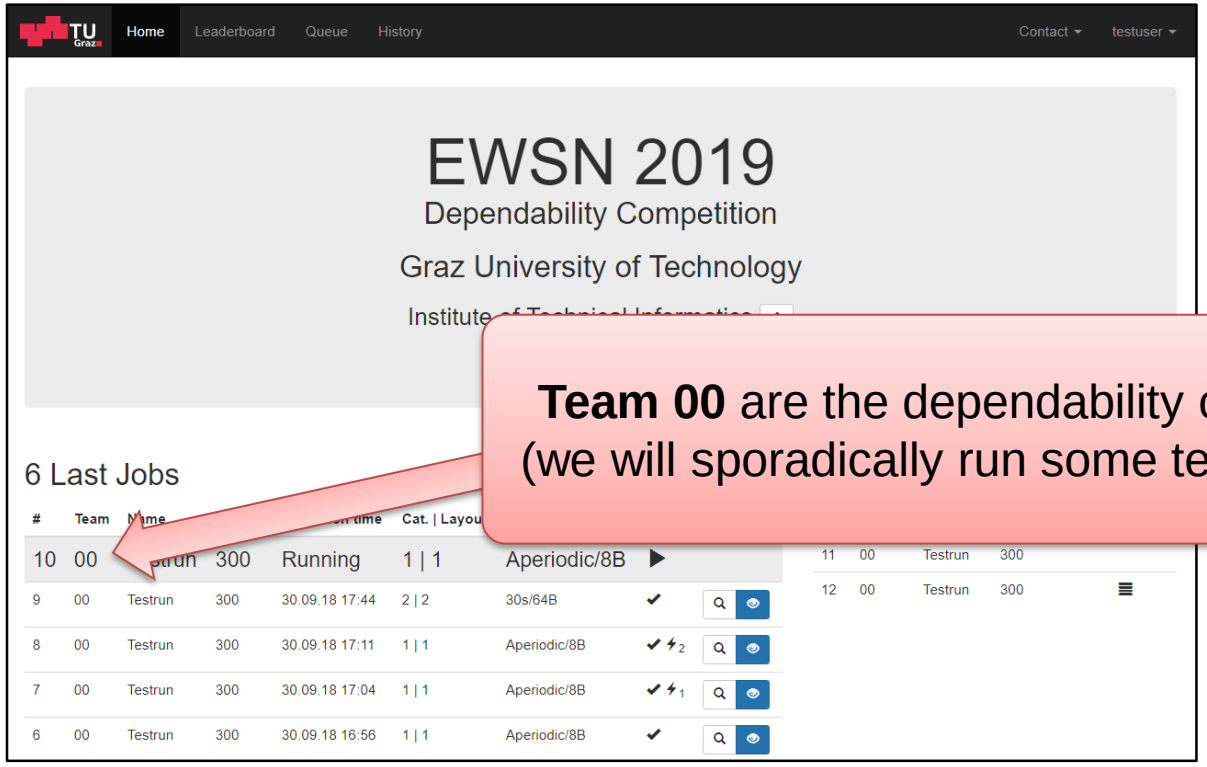
- At a glance
  - Home tab shows the list of all experiments of all teams (completed, running, or queued for execution)



- Currently running
- Successfully completed
- Aborted or failed
- Higher priority job (organizers only)
- Log output enabled (traces only seen by team)
- Visualize results (anyone can see those!)

Competition's Testbed Facility

- At a glance
  - Home tab shows the list of all experiments of all teams (completed, running, or queued for execution)



Team 00 are the dependability competition organizers! ☺
(we will sporadically run some test runs and sanity checks)

▶ Currently running

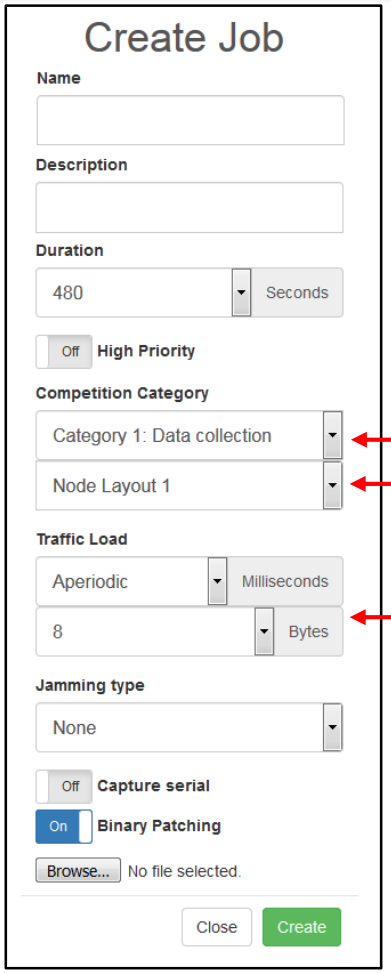
✔ Successfully completed

✖ Aborted or failed

(traces only seen by team)

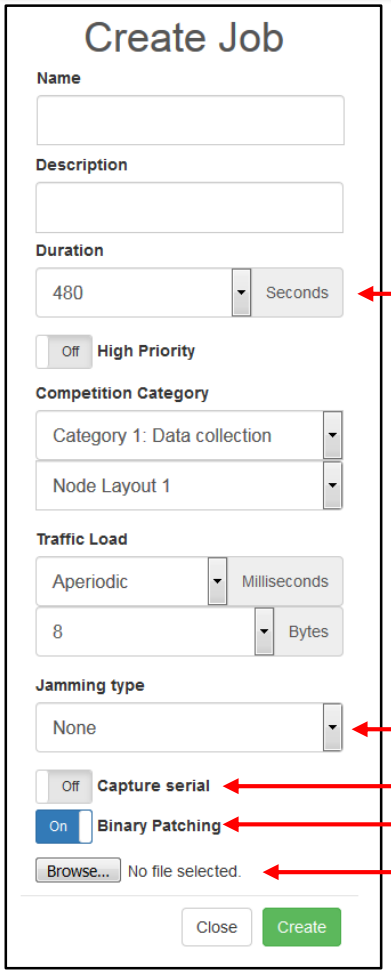
Visualize results
(anyone can see those!)

Firmware Upload



- Contestants can choose in which category the submitted firmware will be competing
- Contestants select one of multiple available node layouts (i.e., different configurations or sets of source and destination nodes)
- Contestants choose the characteristics of the traffic generated by the testbed facility
  - → Aperiodic or periodic traffic
  - → Length in bytes of the data to be transmitted provided in the EEPROM (in this example, 8 bytes)

Firmware Upload



- Contestants can select an experiment duration
  Note: during the preparation phase, it will be limited to max. **480** sec
- Contestants can enable interference in the surroundings of the nodes and specify its level
  Note: this feature will be disabled during the very first days of the preparation phase
- Contestants can capture serial output
  Note: turning FTDI on/off severely affects the energy consumption of the nodes and the accuracy of timing info!
- Binary patching is enabled by default
  (but can be disabled for testing purposes)
- Contestants can upload a single binary `ihex` file: this will be uploaded to all nodes in the network using a common MSP430 Bootstrap Loader

Testbed's Scheduler

- Jobs are typically executed between 18:00 and 8:00 CEST only!

+16% compared to previous edition!

  - On the 28th of October, daylight saving time changes! (testbed will then run between 18:00 and 08:00 CET)
  - During weekends and (Austrian) holidays, experiments can run anytime along the 24 hours

The scheduler is currently active and processing jobs

The scheduler activity will be resumed at 18:00

Amount of time required by the jobs currently queued

Experiments can be queued anytime!

Testbed's Scheduler

- Jobs are typically executed between 18:00 and 8:00 CEST only!

+16% compared to previous edition!

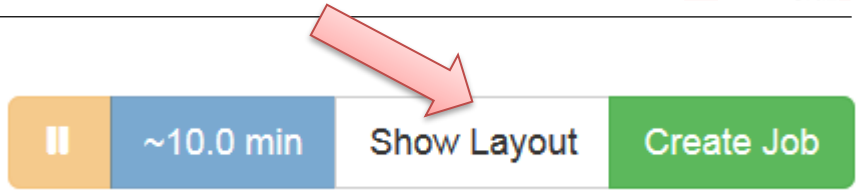
  - On the 28th of October, daylight saving time changes! (testbed will then run between 18:00 and 08:00 CET)
  - During weekends and (Austrian) holidays, experiments can run anytime along the 24 hours

- Why this limitation?
  - During the experiments, a harsh RF environment is created by making use of (among others) Raspberry Pi3 nodes to generate a significant amount of Wi-Fi traffic
  - When heavy Wi-Fi traffic is generated, the University's Wi-Fi infrastructure is severely affected any can be disrupted
  - Therefore, we have agreed with TU Graz to carry out the competition only outside the official working hours

Testbed's Scheduler

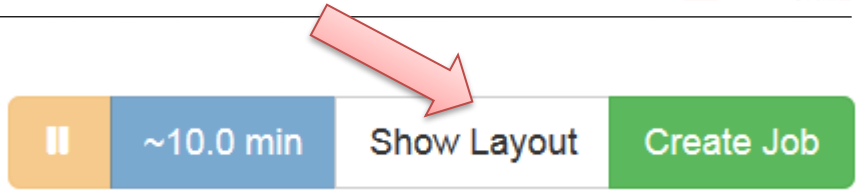
- Jobs execution policy: round-robin
  - Compared to FIFO scheduling, round-robin increases fairness in the number of experiments executed per team in a given time
  - The testbed executes jobs on a per-team basis
    - → Scheduler iterates through the list of teams based on the team number
    - → In case a team is competing in both categories, up to two experiments will be executed before iterating to the next team (at most one for each category)
    - → Once the job(s) of a team complete(s), the testbed executes the next pending job for the next team number (if any)
    - → If there is no job pending for a given team, the scheduler will first iterate through all other teams before scheduling a newly-pending job for that specific team

Layout of Nodes

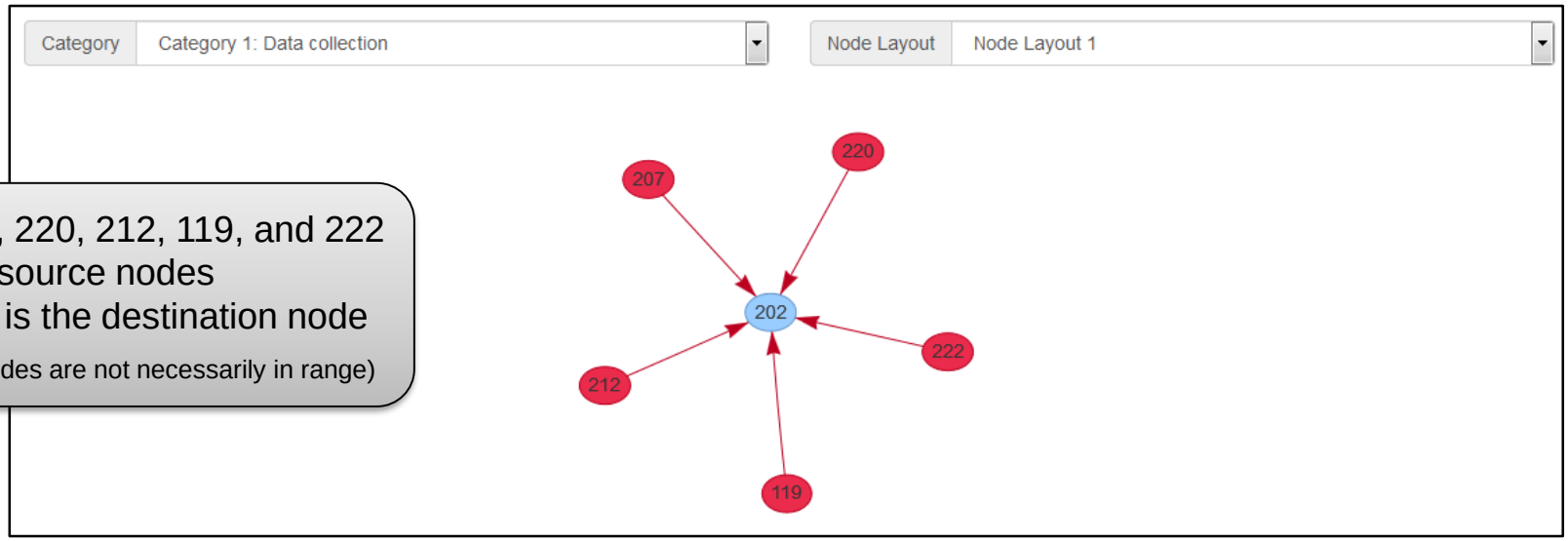


- For each competition category, different node layouts are available
  - Different configurations or sets of source and destination nodes
  - Binary patched by the testbed when running a job
  - Node layout can be specified when creating a job
    - → Layout 1 is representative of a layout for the final evaluation
    - → Layout 2 is a simple node layout with nodes reachable within a single hop for initial tests
    - → Layout "Empty Configuration" is intended to use with binary patching disabled for testing purposes
      - ❖ No data is generated on the EEPROM

Layout of Nodes

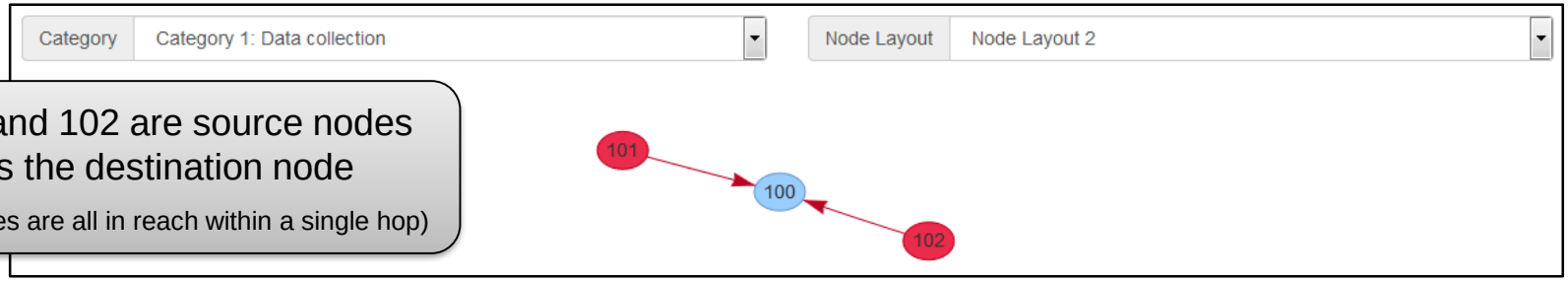


- Category 1: Data collection for condition monitoring



- 207, 220, 212, 119, and 222 are source nodes
- 202 is the destination node

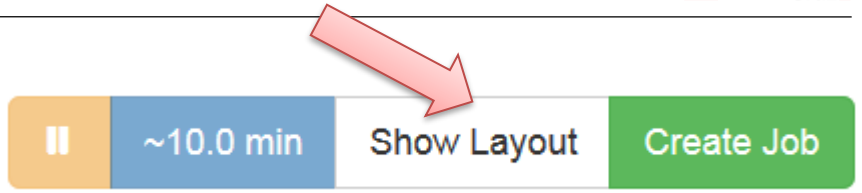
(These nodes are not necessarily in range)



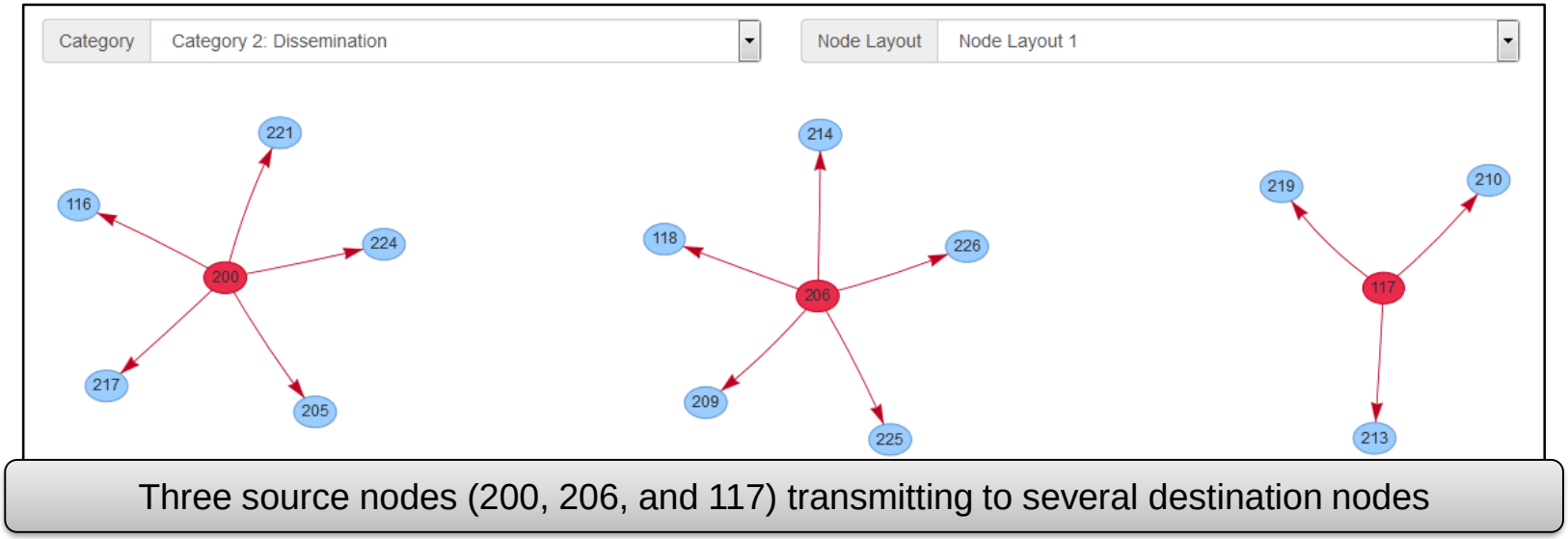
- 101 and 102 are source nodes
- 100 is the destination node

(These nodes are all in reach within a single hop)

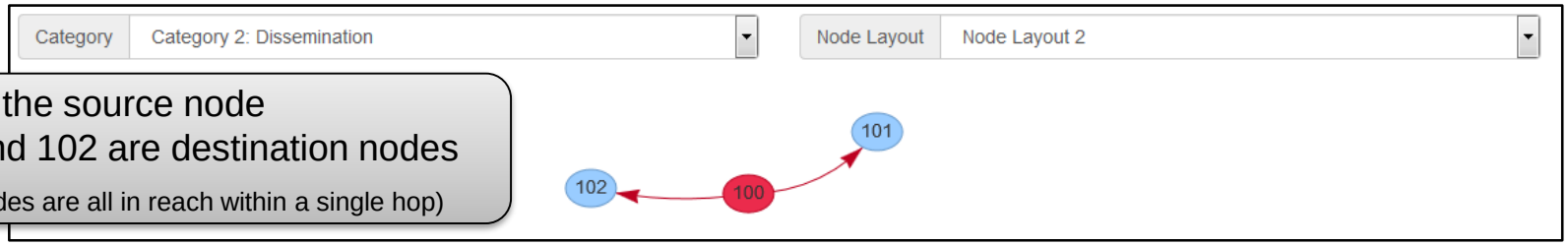
Layout of Nodes



- Category 2: Dissemination of actuation commands



Three source nodes (200, 206, and 117) transmitting to several destination nodes

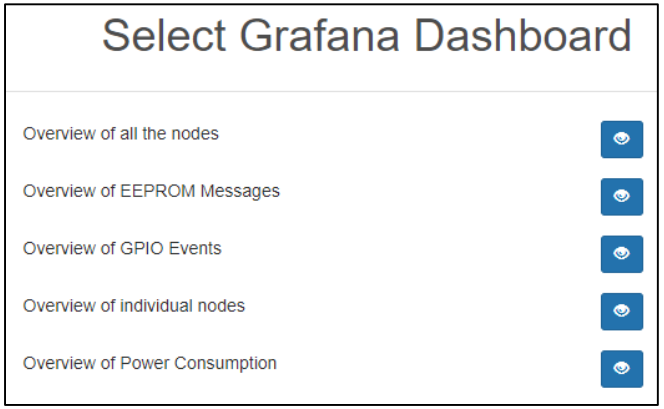


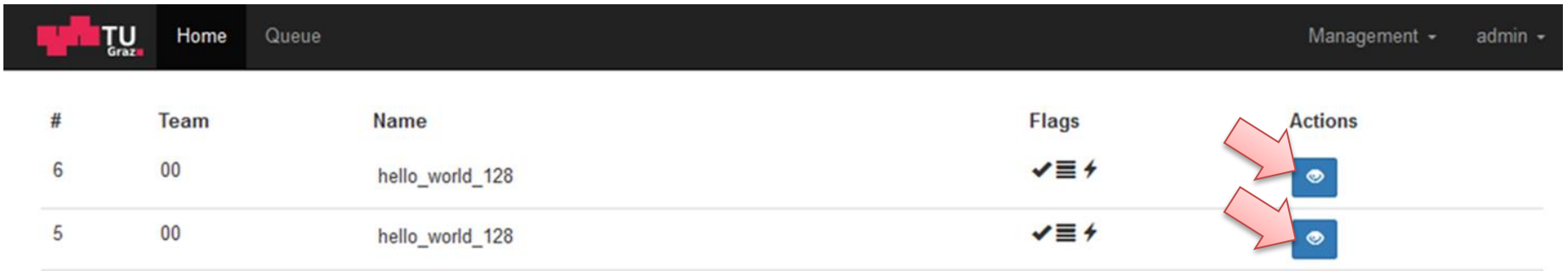
- 100 is the source node
- 101 and 102 are destination nodes

(These nodes are all in reach within a single hop)

Results of an Experiment

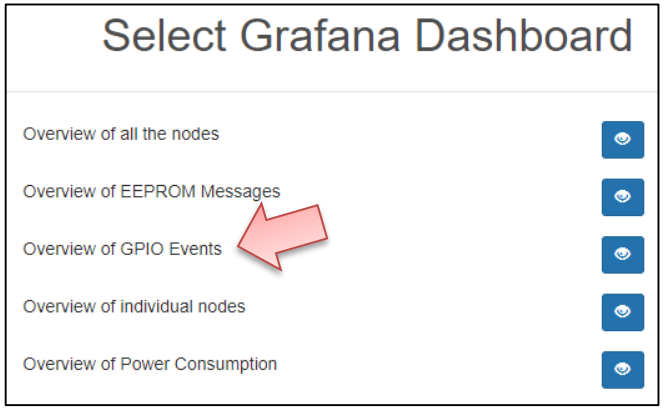
- After the execution of an experiment, graphical results can be checked (by anyone) by clicking on the blue button on the right side
  - Results displayed using Grafana
  - Power consumption and GPIO status is tracked for each node
  - EEPROM messages sent and received for each node
- The team owning a job can also see the program log

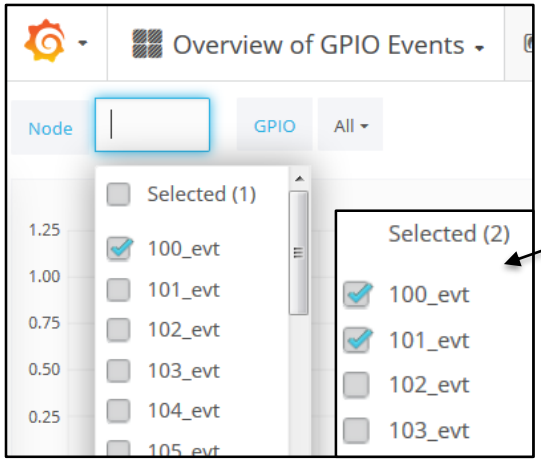




Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption

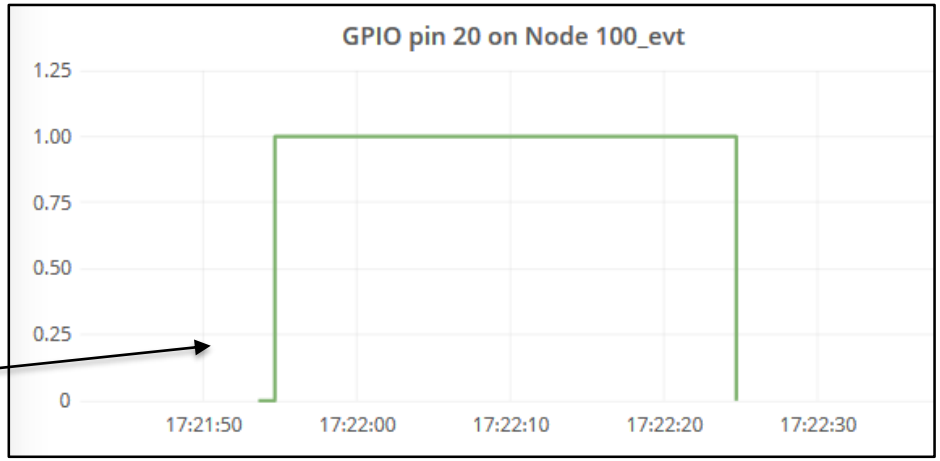




Monitoring the GPIO of multiple nodes at the same time

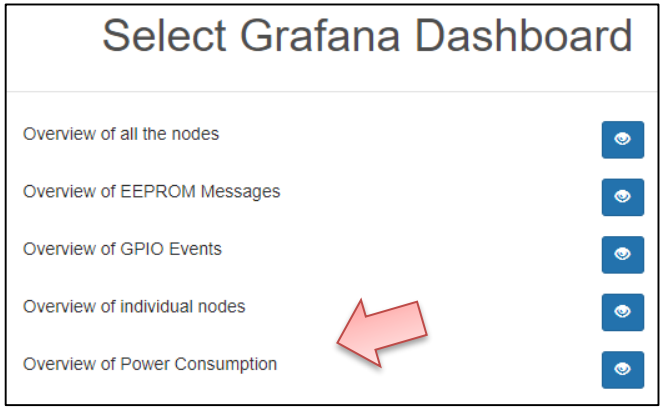
Monitoring individual GPIO pins

(for the numbering explanation check the "GPIO pins" section)



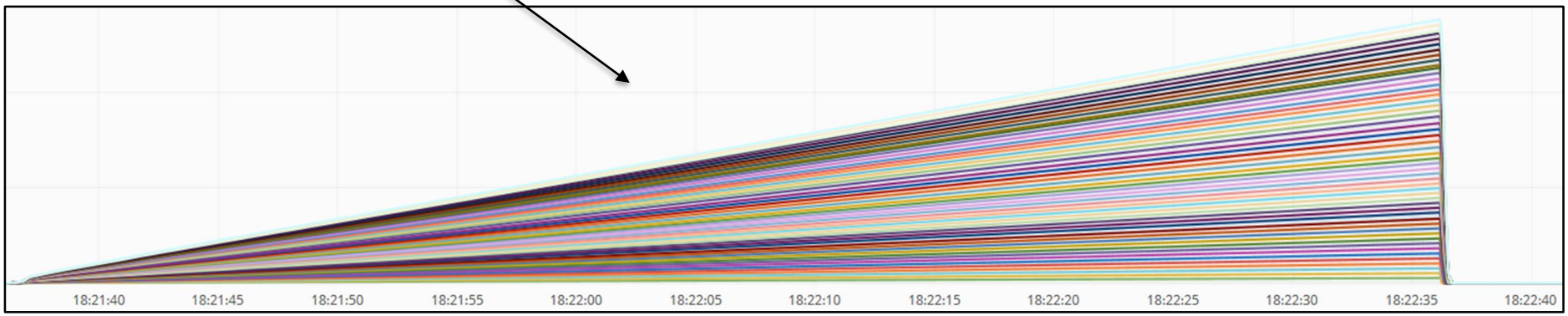
Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



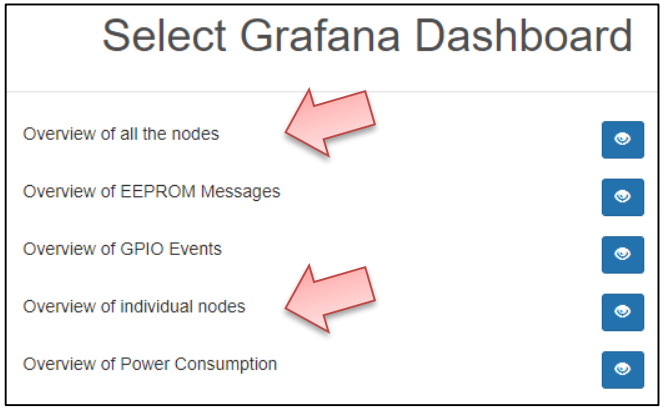
Stacked energy consumption:
Shows the total energy consumed by all nodes in the testbed

Experiment state:
Shows if a sensor node is active (1) or not (0)

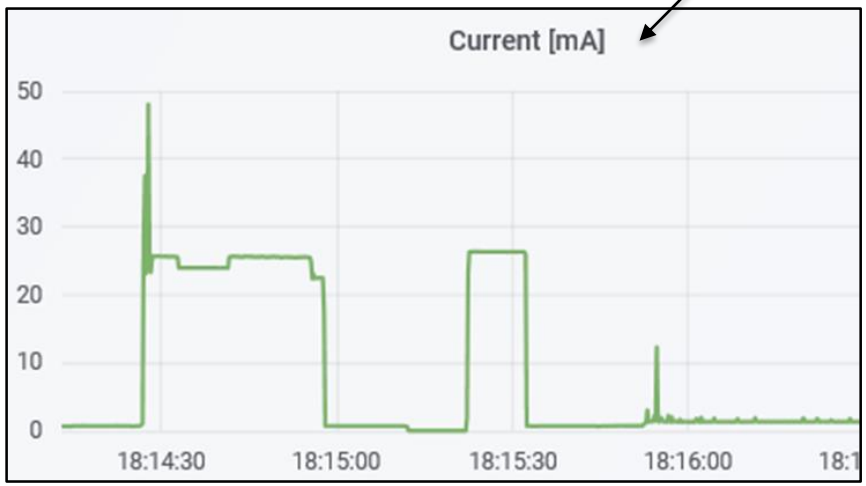


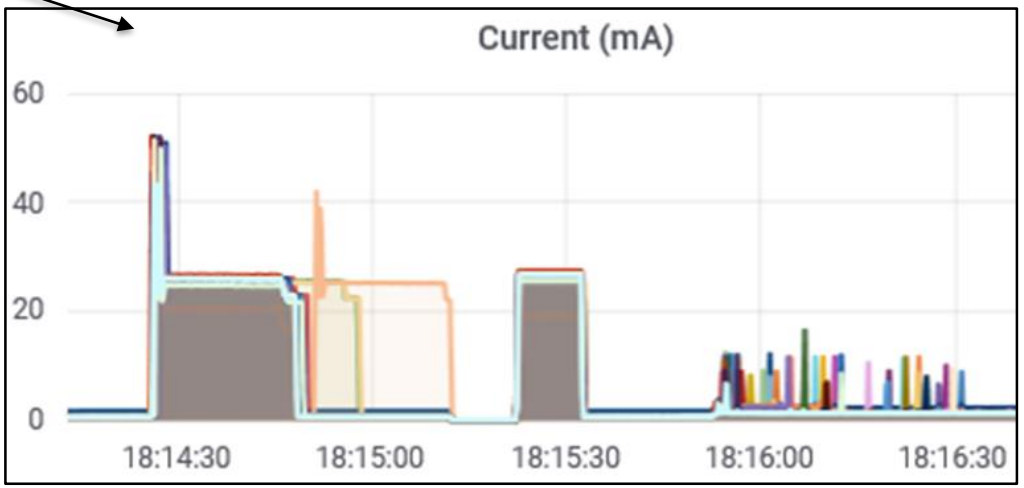
Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



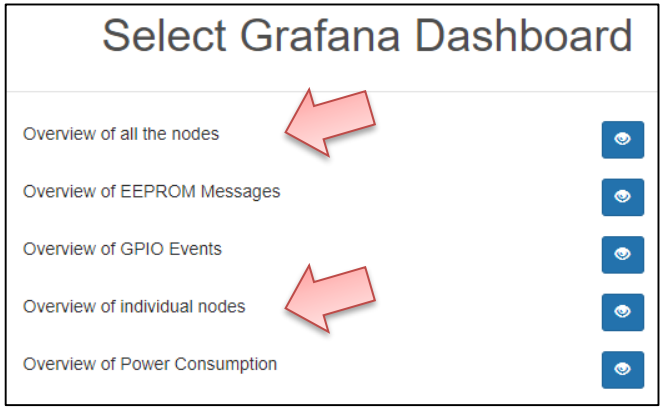
Individual statistics on **voltage, current, power,** and **cumulative energy** for all or specific nodes





Results of an Experiment

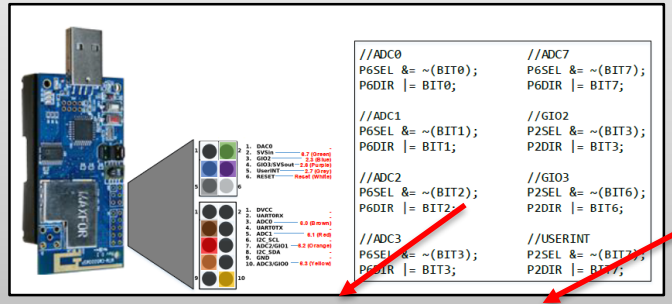
- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



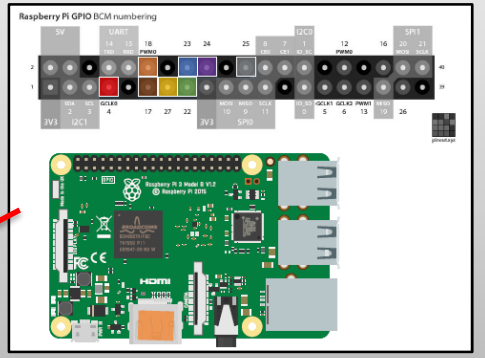
GPIO pins (Information is encoded in a special way – for individual values, use "Overview of GPIO events")

The value is computed as follows:

```
gpio=0;
gpio=gpioRead(17);
gpio=(gpio<<1) | gpioRead(4);
gpio=(gpio<<1) | gpioRead(18);
gpio=(gpio<<1) | gpioRead(27);
gpio=(gpio<<1) | gpioRead(22);
gpio=(gpio<<1) | gpioRead(23);
gpio=(gpio<<1) | gpioRead(24);
gpio=(gpio<<1) | gpioRead(25);
```

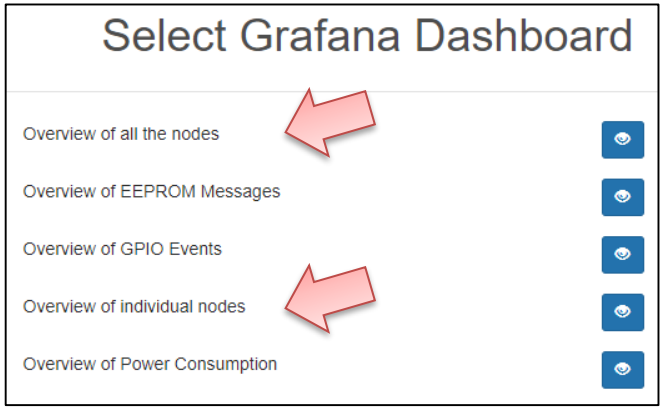


(See "GPIO pins" section for details)



Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



Node status information (serves as a sanity check for both contestants and organizers)

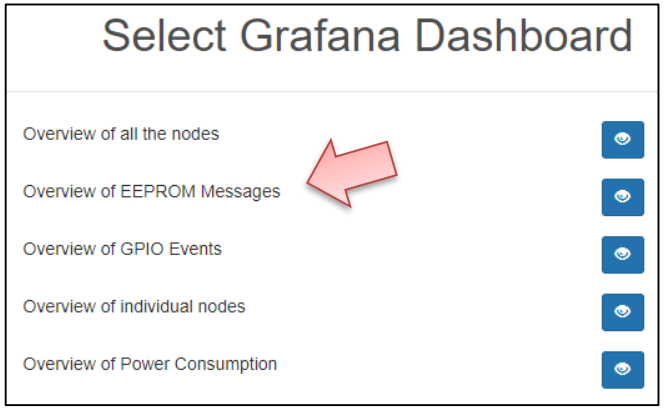
The value is computed as follows:

```
control=0;
control=gpioRead(21);                 // GPIO 21 = TelosB has power? (1 = yes, 0 = no)
control=(control<<1) | gpioRead(20); // GPIO 20 = reset pin of TelosB node (1 = running, 0 = not running)
control=(control<<1) | gpioRead(16); // GPIO 16 = The GPIOs ADC0, ADC1, ADC2, and ADC3 are all configured
                                      as input (0) or as output (1)
control=(control<<1) | gpioRead(12); // GPIO 12 = The GPIOs ADC7, GIO2, GIO3, and USERINT are all
                                      configured as input (0) or as output (1)
```

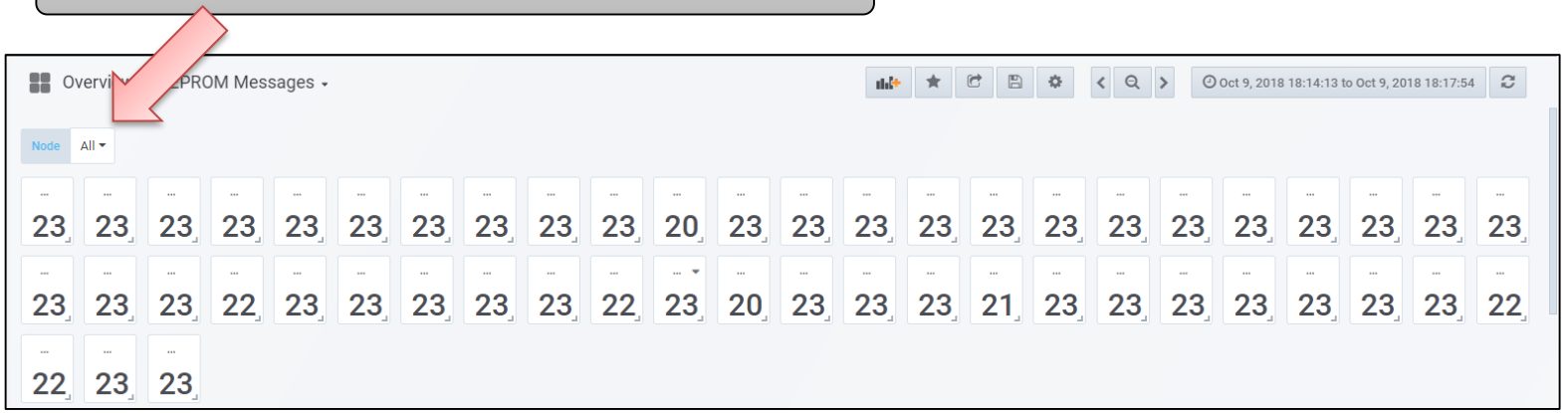
See "GPIO pins" section for details

Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption

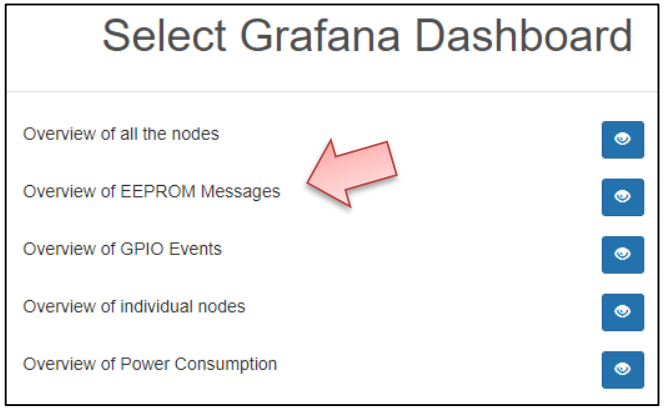


Count of the messages on each node
(displayed for each node selected here)



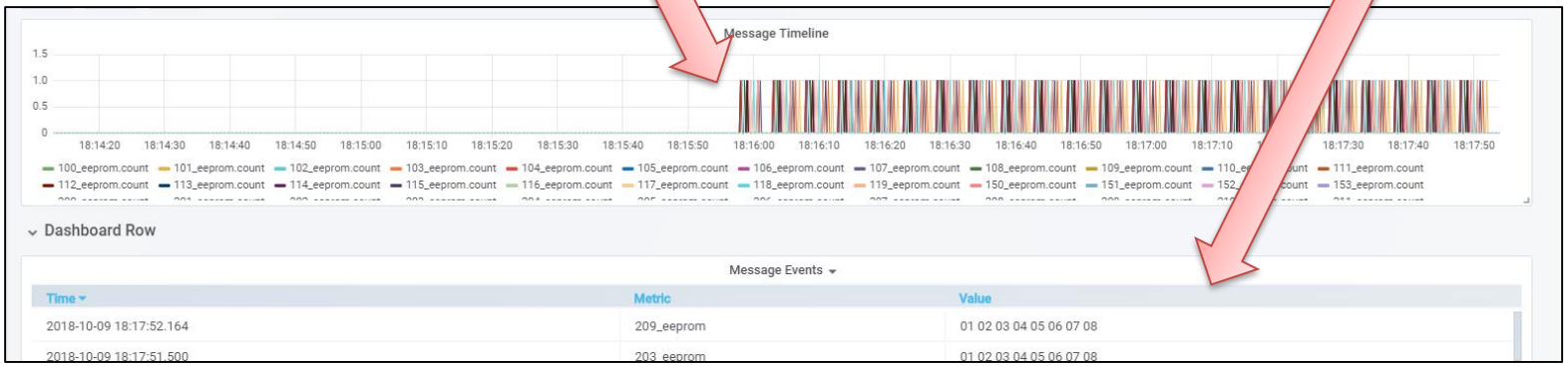
Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



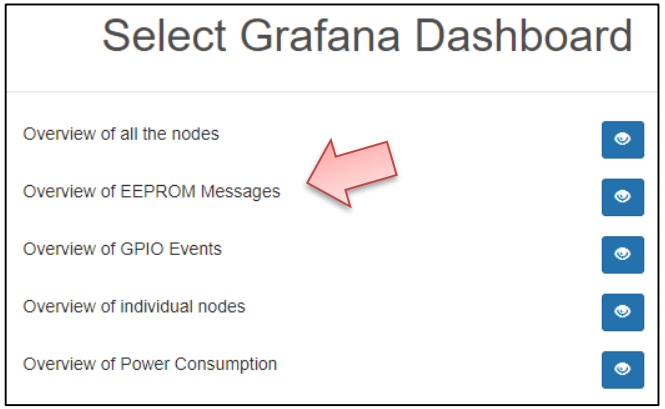
Histogram of the messages on each node
(displayed for each selected node)

Individual messages sent and received by each node

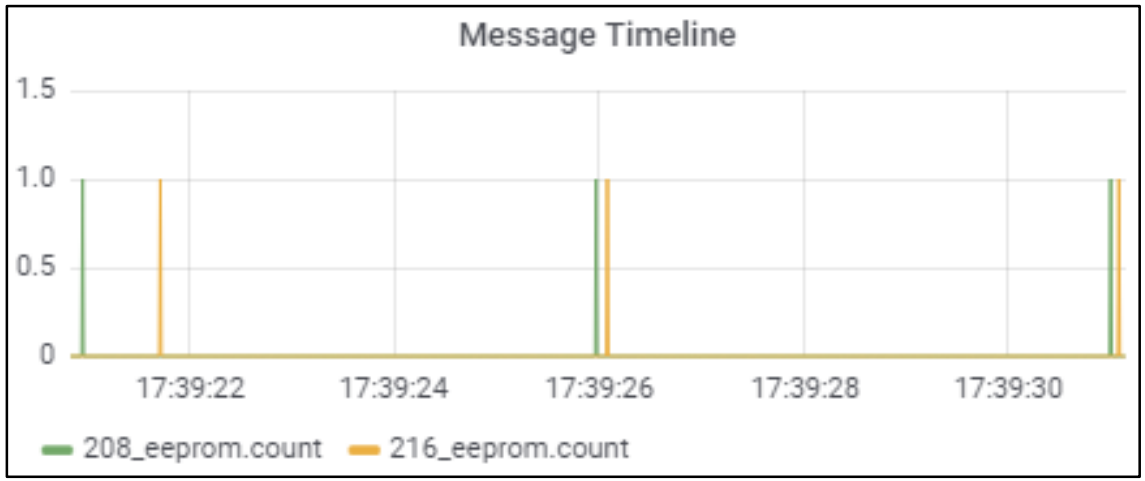


Results of an Experiment

- Grafana dashboards
  - Overview of all the nodes
  - Overview of EEPROM messages
  - Overview of GPIO events
  - Overview of individual nodes
  - Overview of power consumption



Example: how to visualize latency of individual messages
(208 is the source, 216 is the destination)



Message Events

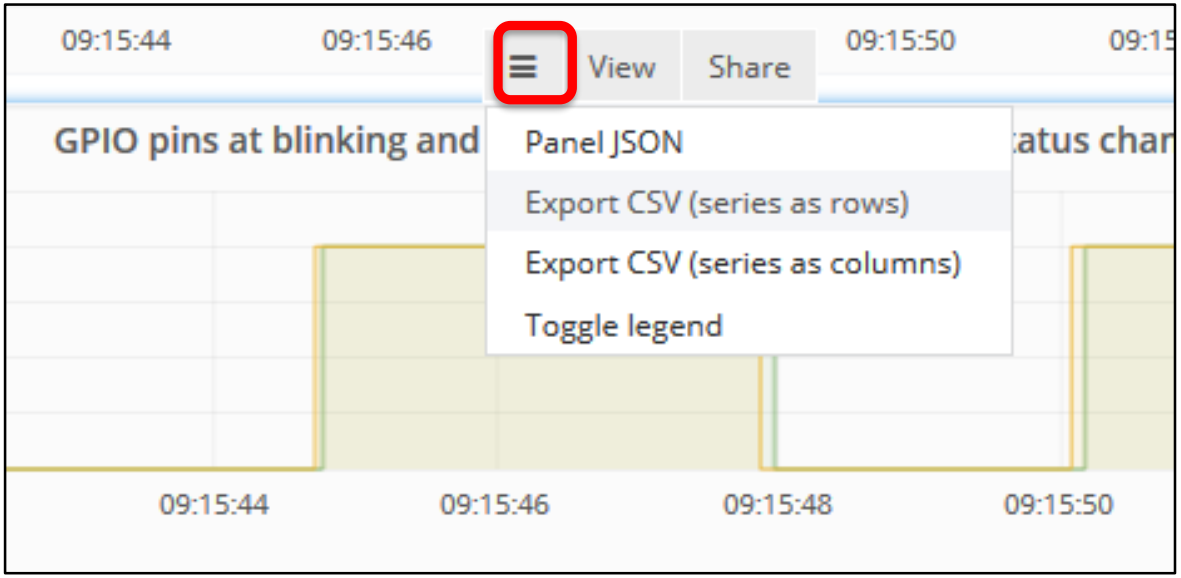
Time	Metric	Value
2018-10-01 17:39:31.085	216_eeprom	fb 29 d1 e6
2018-10-01 17:39:31.003	208_eeprom	fb 29 d1 e6
2018-10-01 17:39:26.088	216_eeprom	fb fa aa 3a
2018-10-01 17:39:25.979	208_eeprom	fb fa aa 3a
2018-10-01 17:39:21.713	216_eeprom	02 1a fe 43
2018-10-01 17:39:20.955	208_eeprom	02 1a fe 43

Visualization in Grafana – FAQ

- Why is Grafana not displaying any point when I zoom in?
  - Grafana uses second resolution for the zoom
  - When zooming too much, the averaging may lead to a situation in which Grafana uses the same timestamp as startpoint and endpoint and cannot hence visualize a line

Visualization in Grafana – FAQ

- Can we export the data seen in Grafana?
  - Yes, CSV files can be exported by clicking on the title of the plot
  - Click on the menu icon and select "Export CSV"

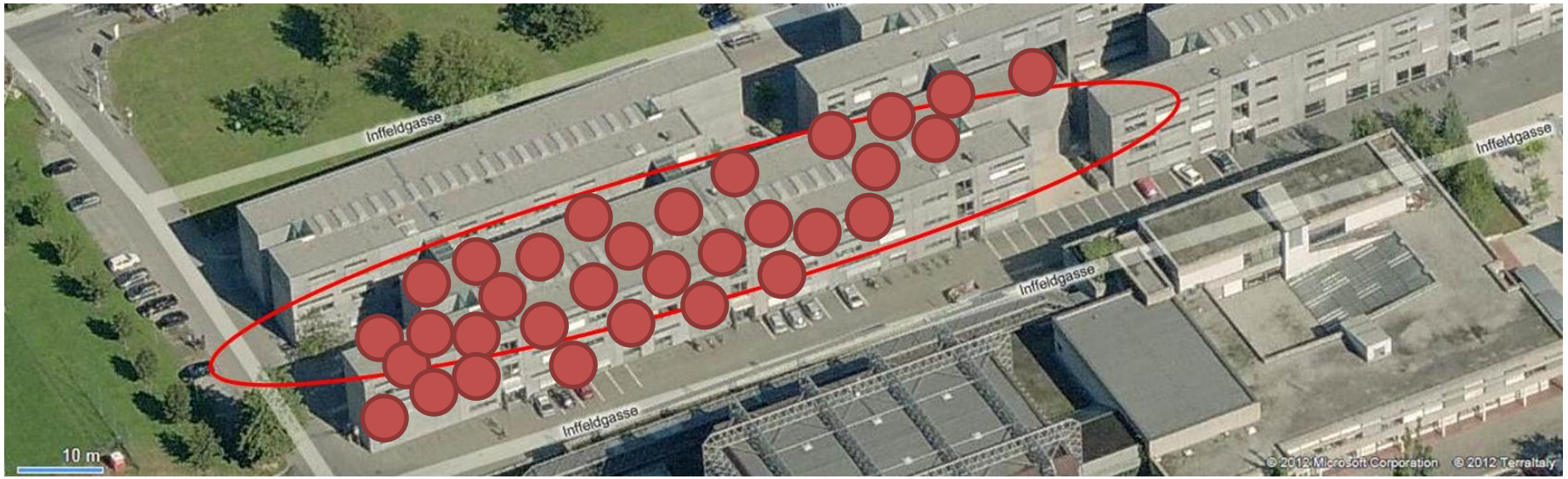


	A	B	C
1	Time	1	2
2	2017-02-16T09:43:46.876Z	0.0840805771962	0.1951102
3	2017-02-16T09:43:47.501Z	0.152616695366	0.2566677
4	2017-02-16T09:43:48.126Z	0.221115444991	0.2613602
5	2017-02-16T09:43:48.751Z	0.289725498238	0.2663699
6	2017-02-16T09:43:49.376Z	0.336447792086	0.2709752

	A	B	C
1	Series	Time	Value
2	Sink node	2017-02-16T09:49:06.669Z	1
3	Sink node	2017-02-16T09:49:08.868Z	0
4	Sink node	2017-02-16T09:49:13.570Z	1
5	Sink node	2017-02-16T09:49:16.571Z	0
6	Sink node	2017-02-16T09:49:25.068Z	1
7	Sink node	2017-02-16T09:49:28.674Z	0

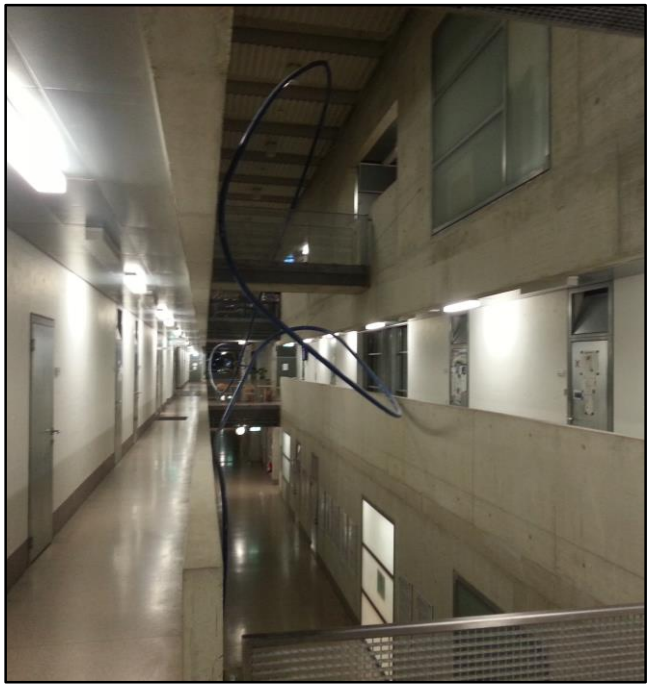
Testbed Location

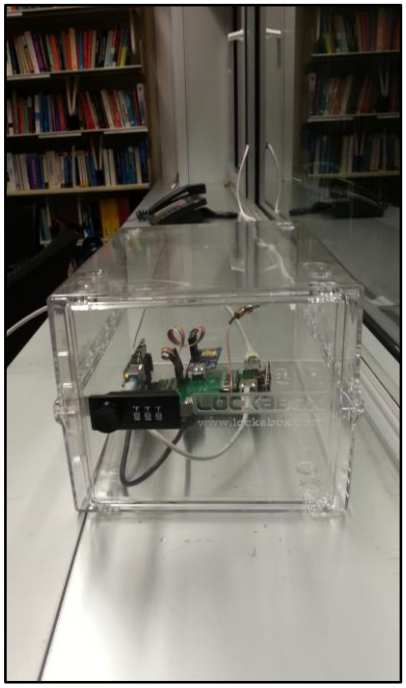
- Nodes are deployed in Inffeldgasse 16 (Graz, Austria)
  - University offices, seminar rooms, and laboratories (belonging to the Institute for Technical Informatics of TU Graz)
  - 51 testbed nodes currently active over multiple floors
  - Density of nodes varies across the building



Testbed Location

- Nodes are deployed in Inffeldgasse 16 (Graz, Austria)
  - University offices, seminar rooms, and laboratories (belonging to the Institute for Technical Informatics of TU Graz)
  - 51 testbed nodes currently active over multiple floors
  - Density of nodes varies across the building





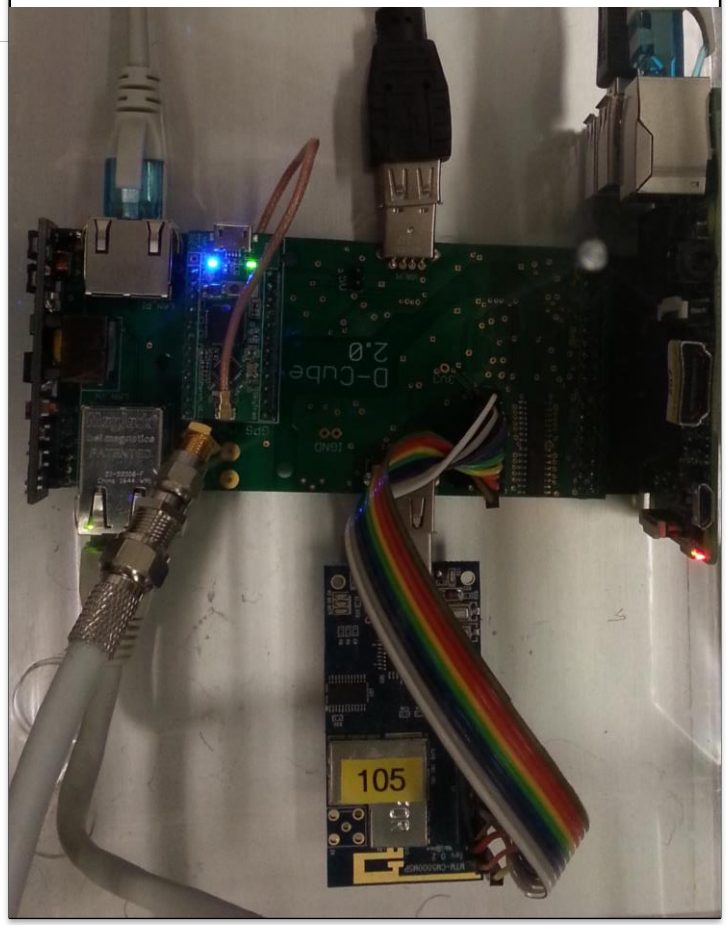
Testbed Hardware

- The testbed allows contestants to program several Maxfor/Advanticsys MTM-CM5000-MSP nodes (replicas of TelosB/Tmote Sky nodes)
  - With and without SMA antenna
  - All powered via USB
  - 10 kB of RAM
  - Attached to D-Cube (http://www.iti.tugraz.at/D-Cube)

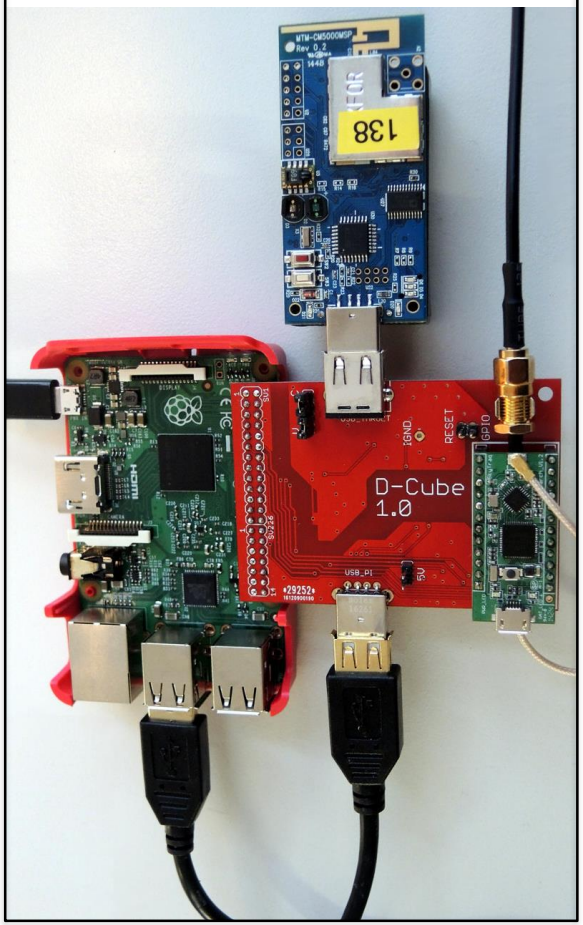
Testbed Hardware: D-Cube

- More info: http://iti.tugraz.at/d-cube

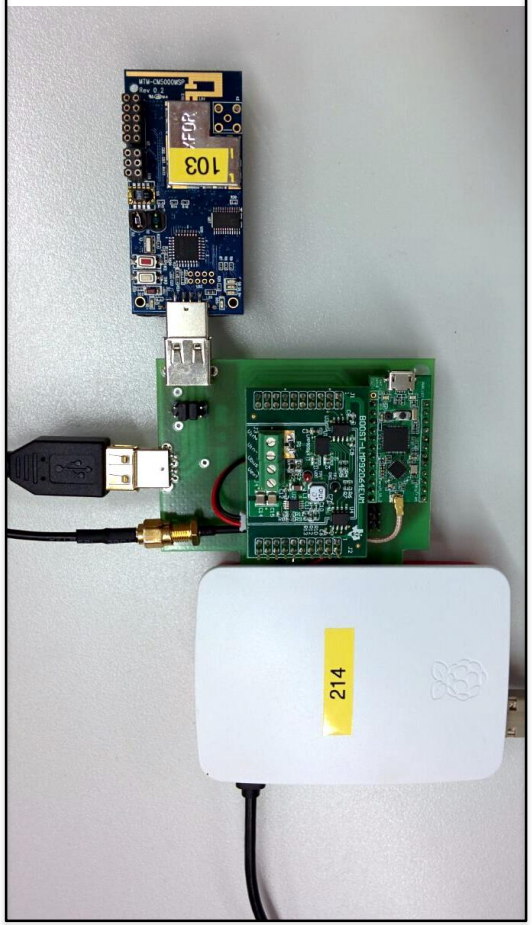
EWSN'18 version



EWSN'17 version



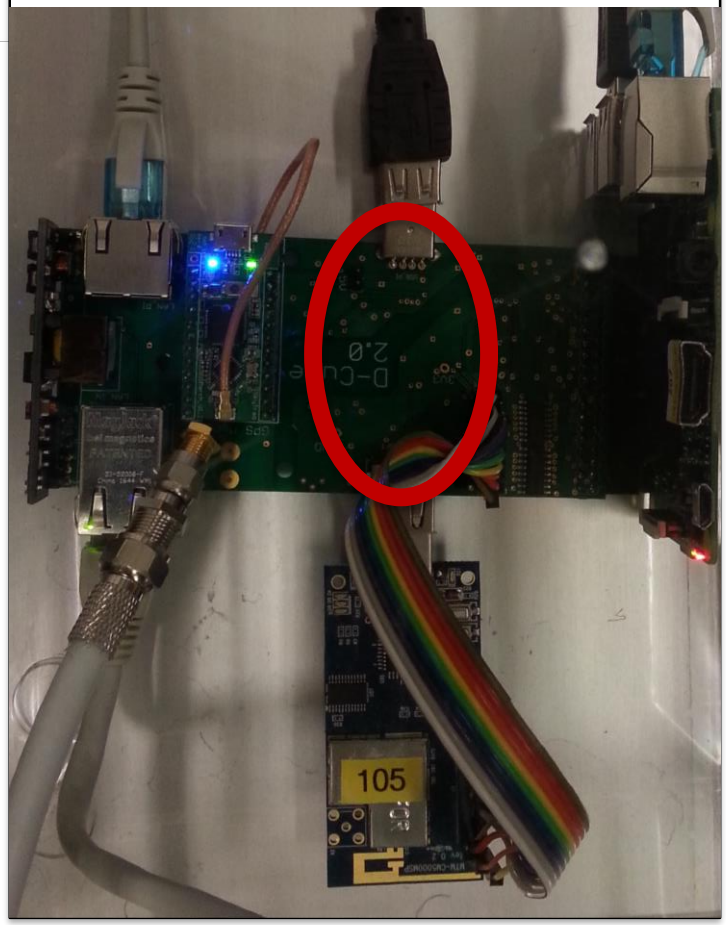
EWSN'16 version



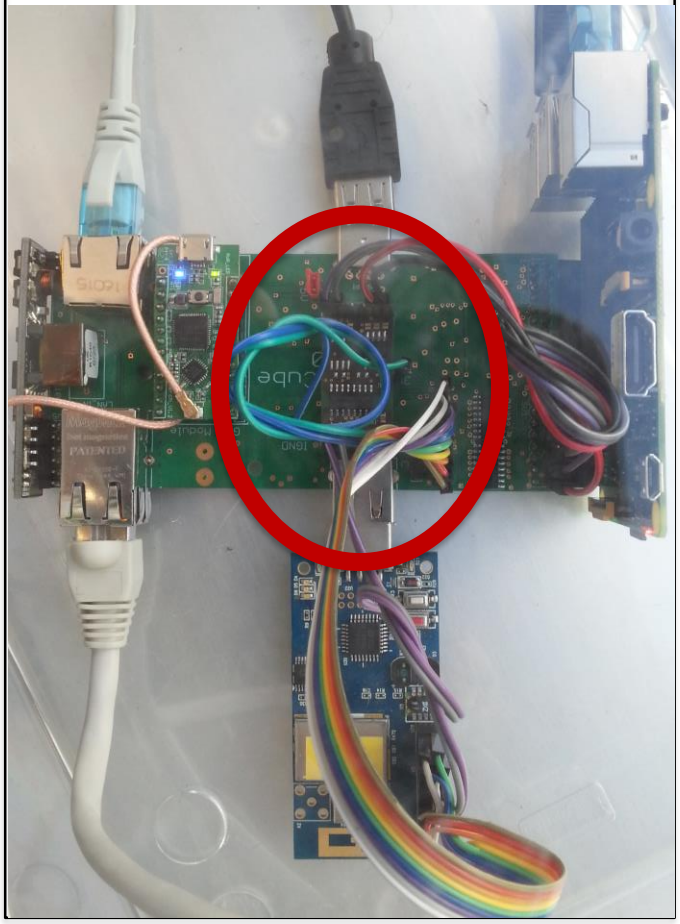
Testbed Hardware: D-Cube

- More info: http://iti.tugraz.at/d-cube

EWSN'18 version

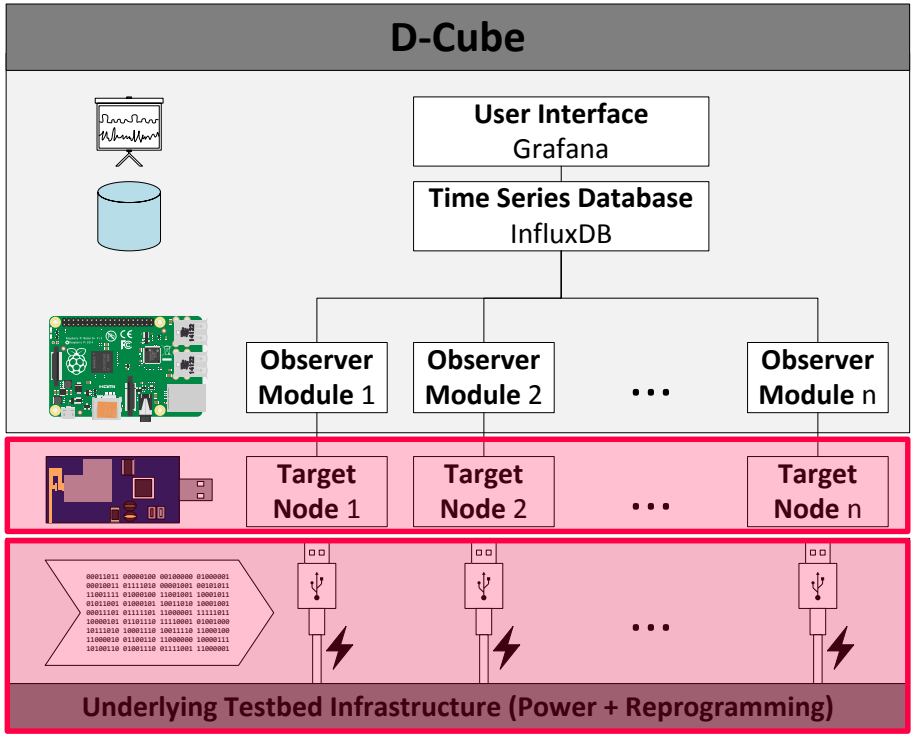


This year's prototype (EWSN'19)



- Same as the 2018 edition, but with an additional EEPROM
- This allows to read/write variable size payloads

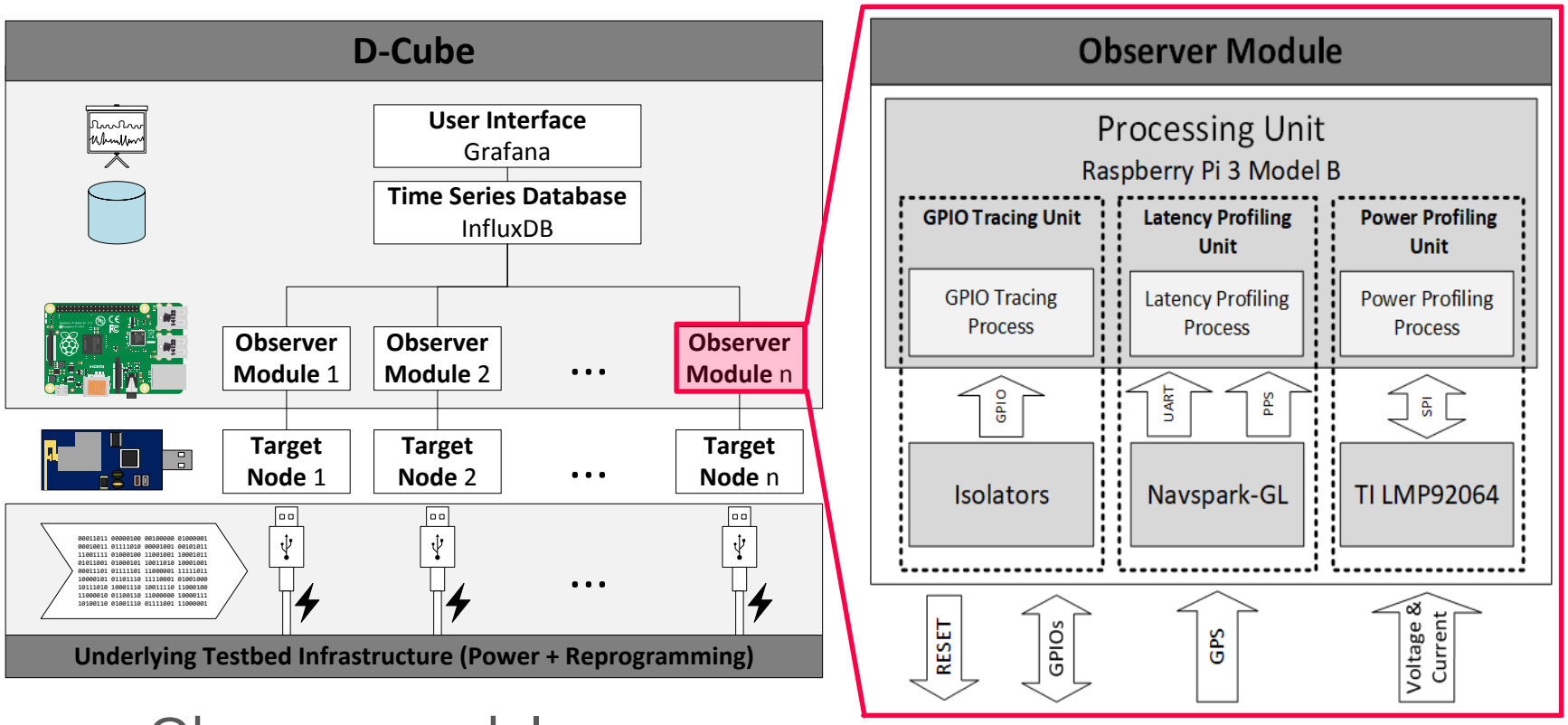
Testbed Hardware: D-Cube



- Target nodes
  - → Devices running the code/system under test
  - → D-Cube agnostic to HW platform chosen as target
  - → MTM-CM5000-MSP node (TelosB replica - 10 kB RAM)

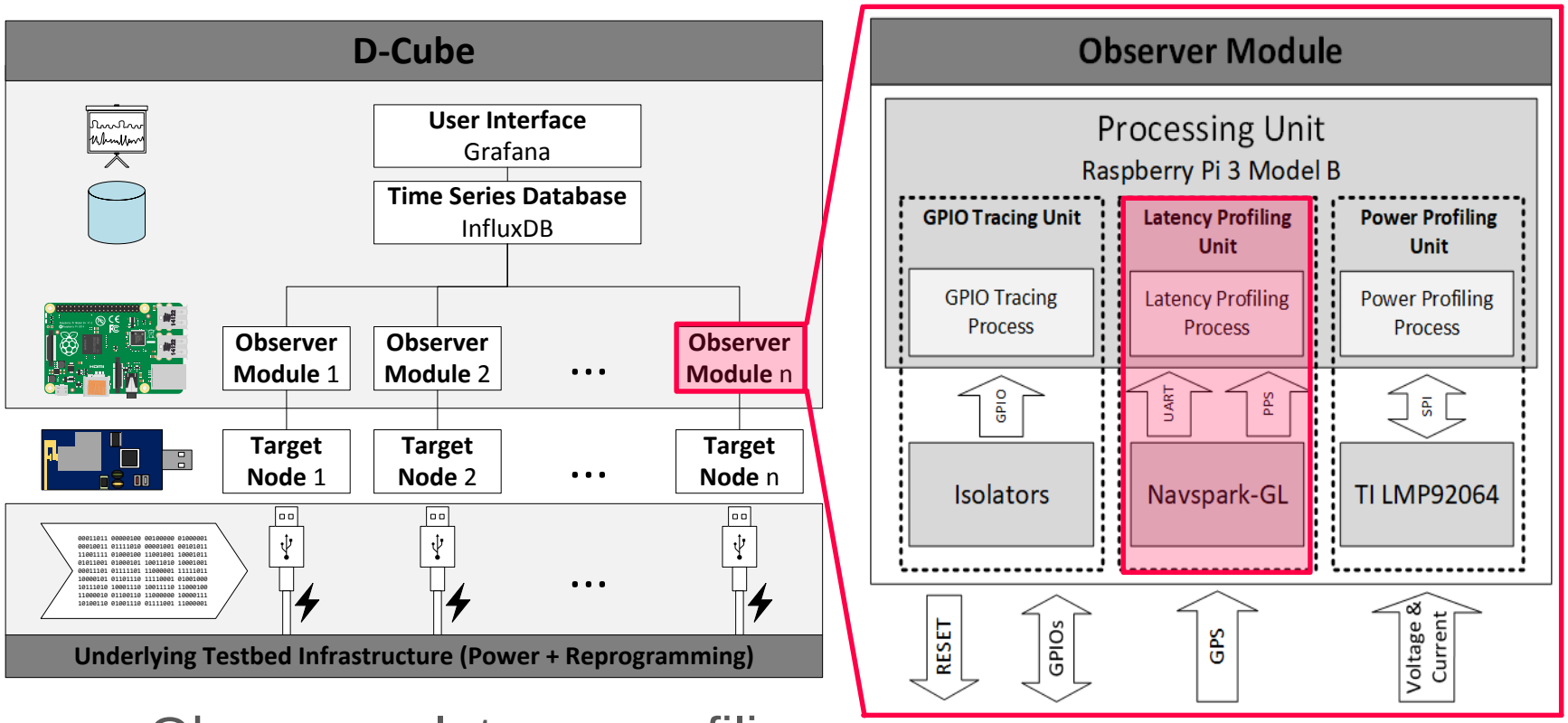
- Underlying infrastructure
  - → Power + reprogramming of the target nodes
  - → Allows to disable the UART interface

Testbed Hardware: D-Cube



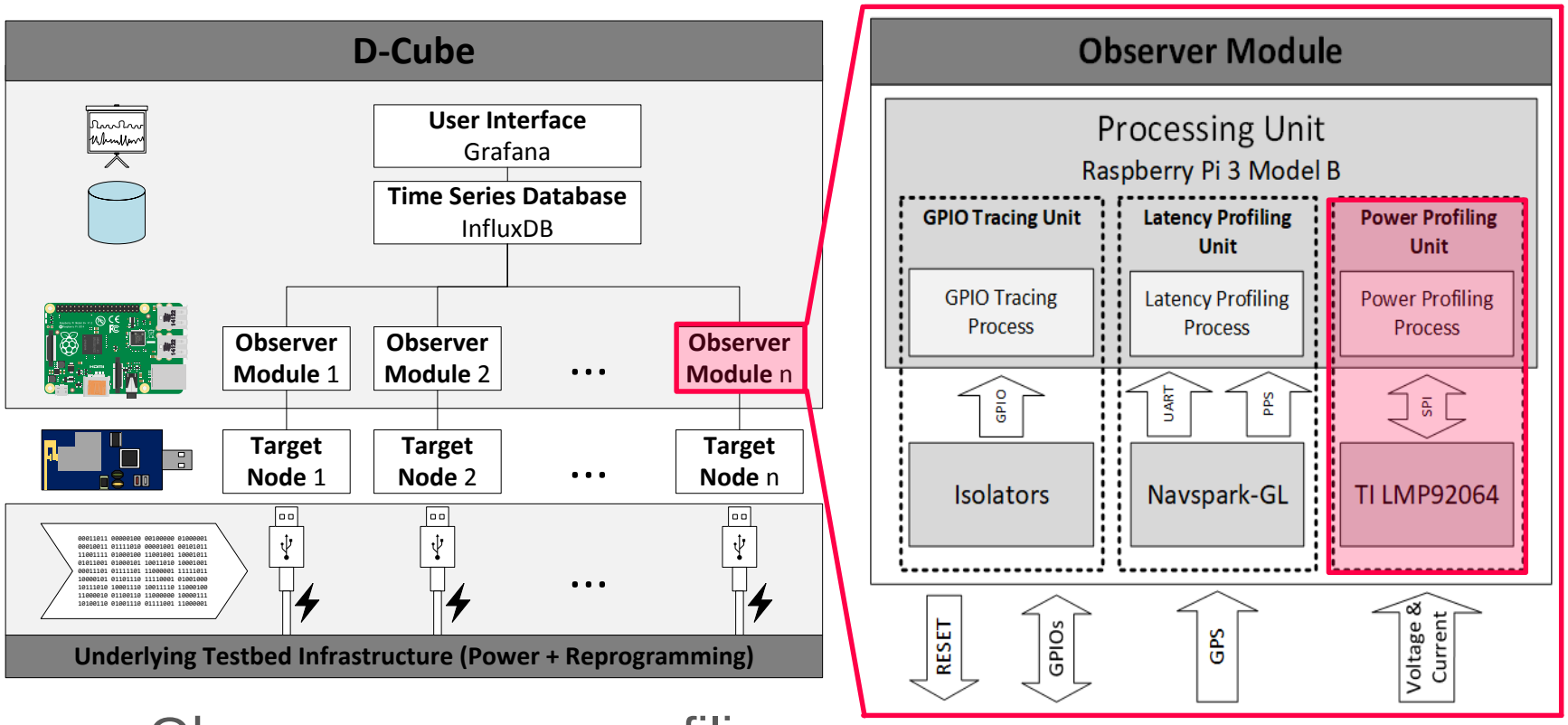
- Observer modules
  - → Each module monitors exactly one target node
  - → Raspberry Pi 3 + custom-made add-on card (ADC+GPS)

Testbed Hardware: D-Cube



- Observers: latency profiling
  - → GPS module to synchronize system clock (NavSpark-GL: Arduino DevBoard with GPS/GLONASS)
    http://navspark.mybigcommerce.com/navspark-gl-arduino-compatible-development-board-with-gps-glonass/
  - → Ensures accurate time measurements across the nodes in the testbed

Testbed Hardware: D-Cube

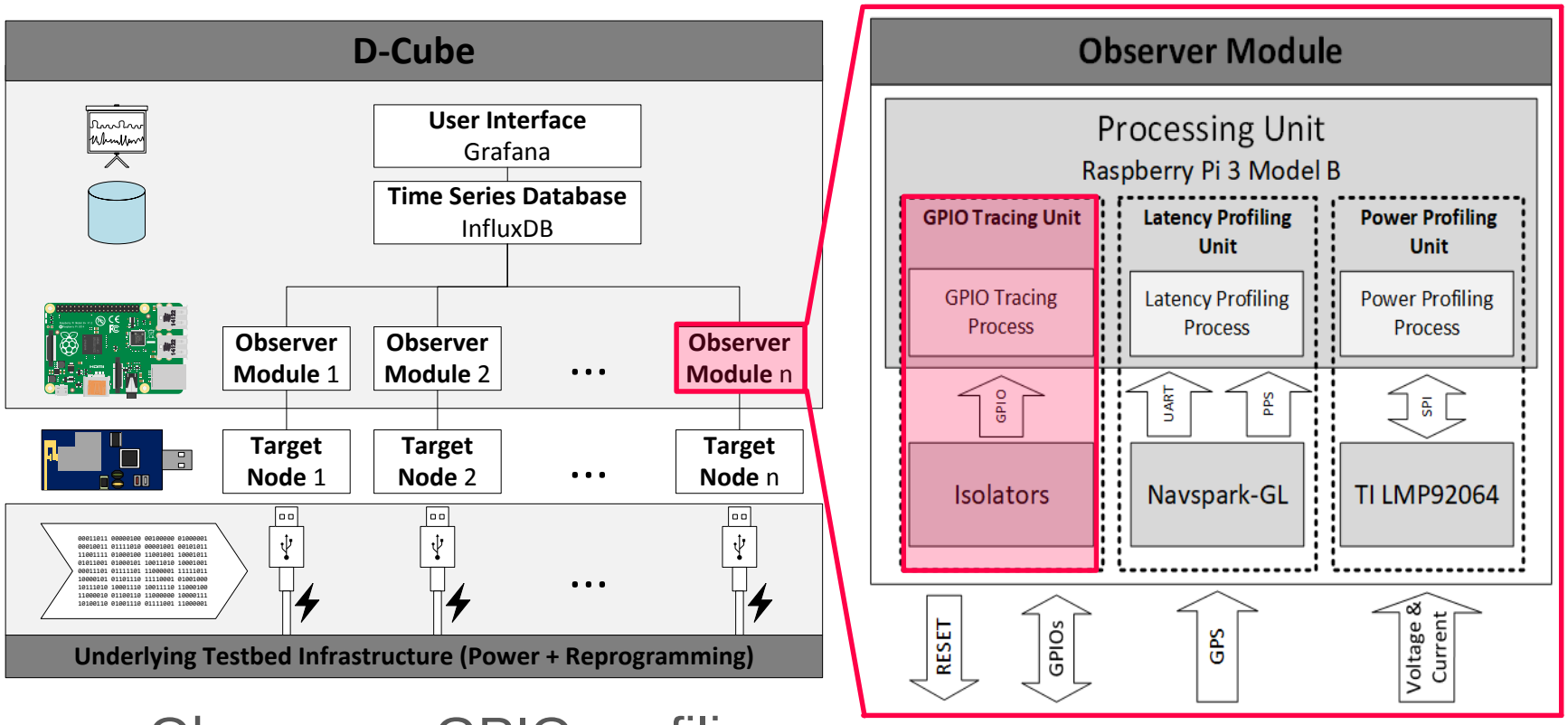


- Observers: power profiling

  → Simultaneous sampling ADC (TI LMP92064) read via SPI @ 62.5 kHz using a real-time process

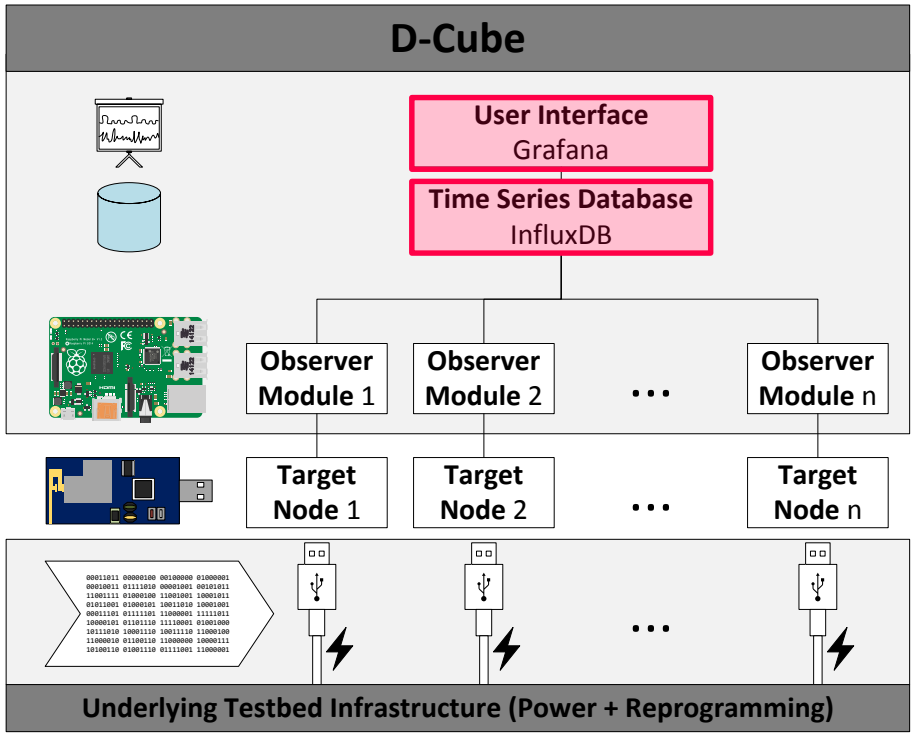
  - Voltage channel: up to 10.82V with 2.82mV resolution
  - Current channel: up to 150.59mA with 39.22µA resolution

Testbed Hardware: D-Cube



- Observers: GPIO profiling
  - → GPIO changes are monitored using the same real-time process sampling the ADC
  - → System clock accuracy is ensured by the GPS module (NTP for nodes where GPS is unavailable)

Testbed Hardware: D-Cube

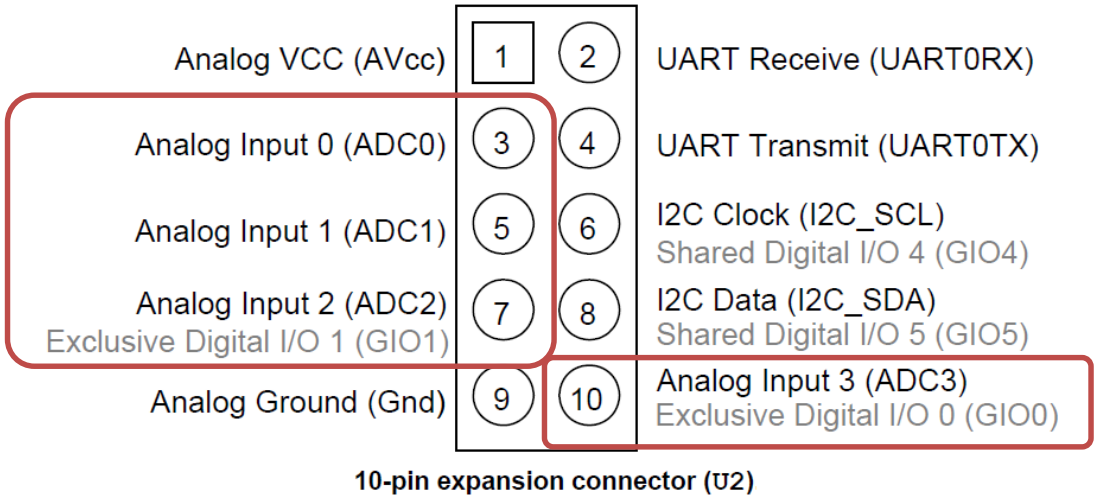


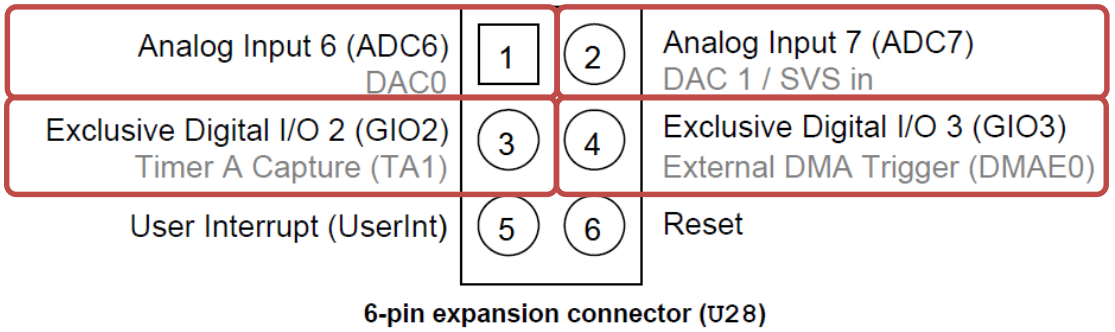
- Time Series database
  - → Collects and persistently stores the data from all observers
  - → InfluxDB (open-source)
  - → Nanosecond precision timestamps
- User Interface
  - → Acts as proxy to the database and gives real-time feedback
  - → Grafana (open-source)

GPIO Pins

GPIO Pins

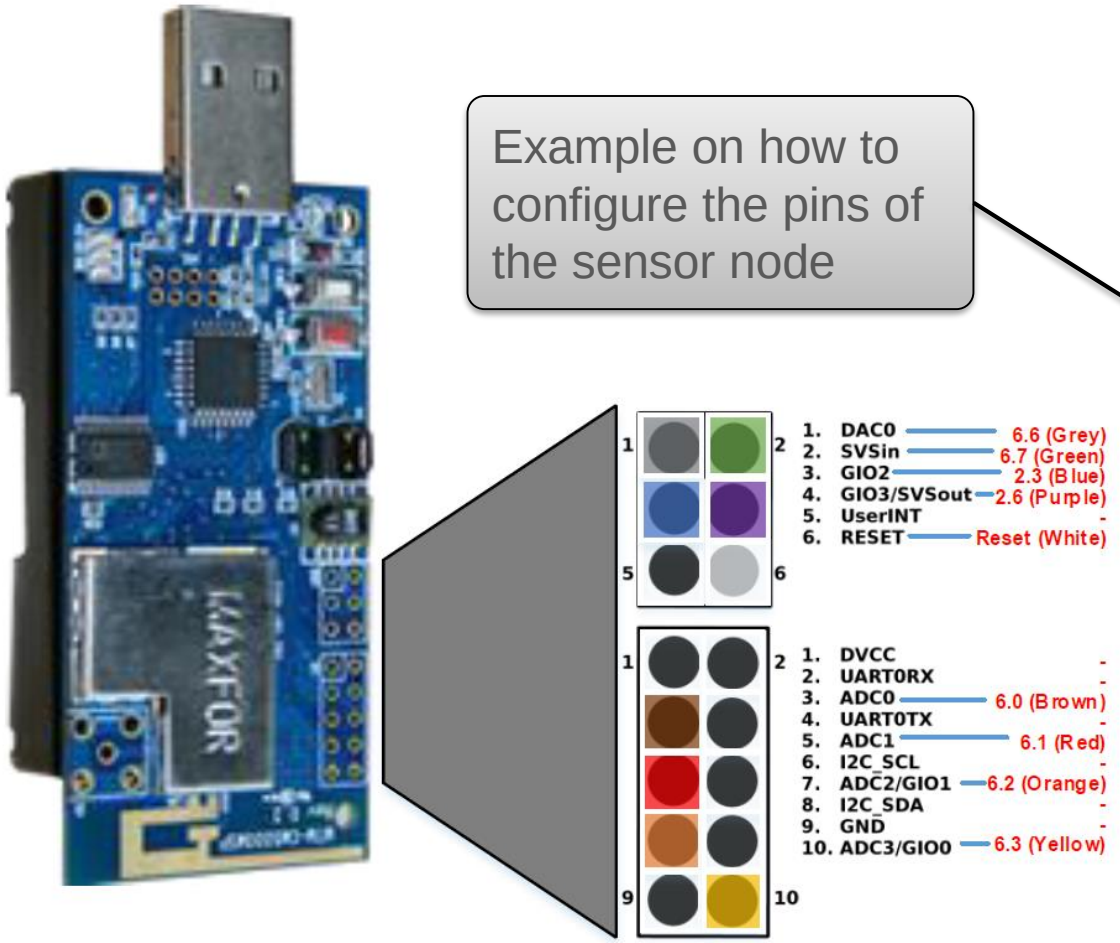
- The testbed facility is connected to eight of the pins available in the 10-pin and 6-pin expansion connector





GPIO Pins

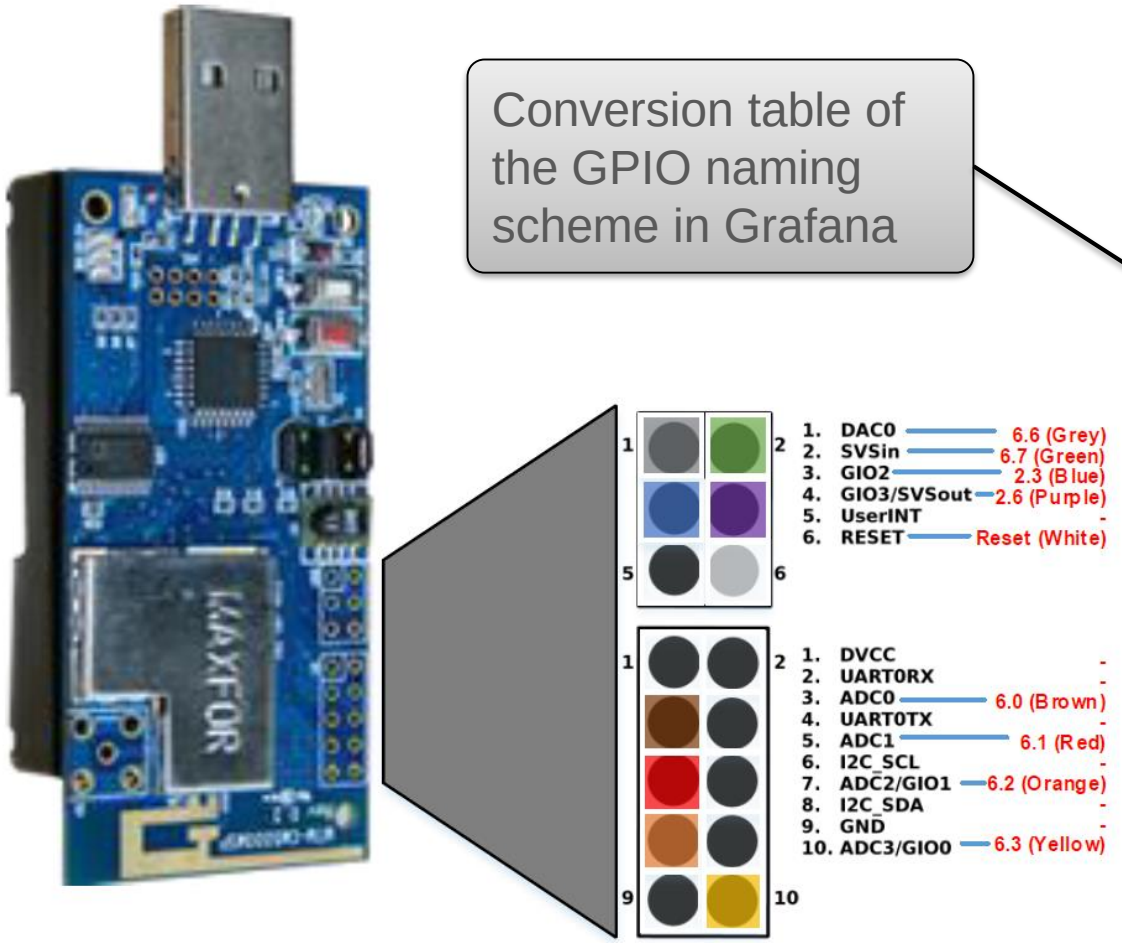
- The testbed facility is connected to eight of the pins available in the 10-pin and 6-pin expansion connector




```
//ADC0                          //ADC7
P6SEL &= ~(BIT0);               P6SEL &= ~(BIT7);
P6DIR |= BIT0;                  P6DIR |= BIT7;

//ADC1                          //GIO2
P6SEL &= ~(BIT1);               P2SEL &= ~(BIT3);
P6DIR |= BIT1;                  P2DIR |= BIT3;

//ADC2                          //GIO3
P6SEL &= ~(BIT2);               P2SEL &= ~(BIT6);
P6DIR |= BIT2;                  P2DIR |= BIT6;

//ADC3                          //ADC6 / DAC0
P6SEL &= ~(BIT3);               P6SEL &= ~(BIT6);
P6DIR |= BIT3;                  P6DIR |= BIT6;
```

GPIO Pins

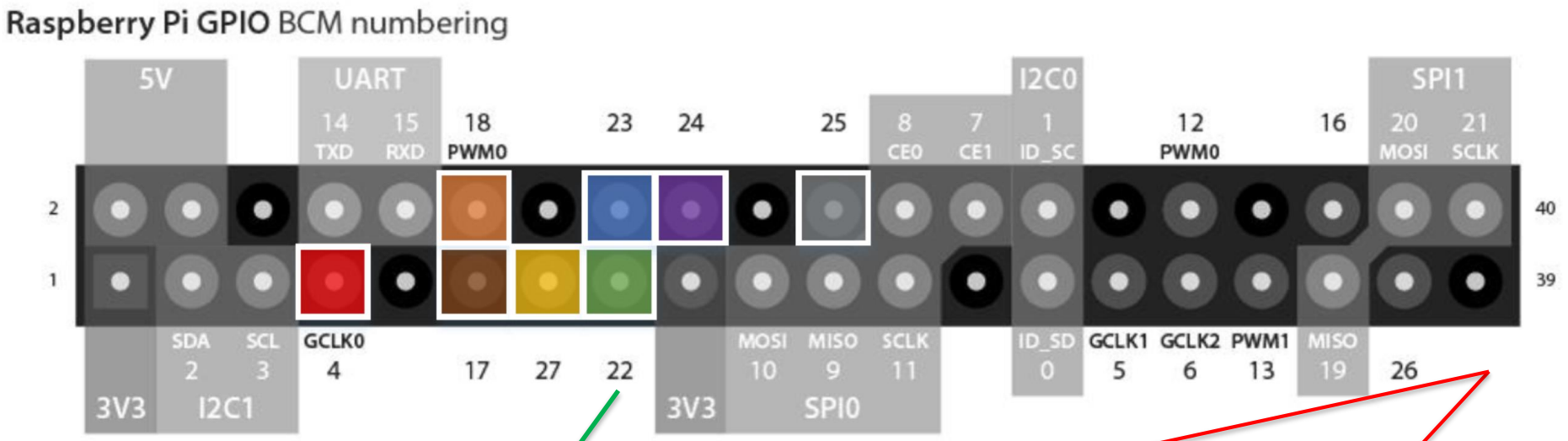
- The testbed facility is connected to eight of the pins available in the 10-pin and 6-pin expansion connector



Sensor Node	Grafana
ADC0	GPIO 17
ADC1	GPIO 4
ADC2/GIO1	GPIO 18
ADC3/GIO0	GPIO 27
ADC7/SVSin	GPIO 22
GIO2	GPIO 23
GIO3/SVSout	GPIO 24
DAC0/ADC6	GPIO 25

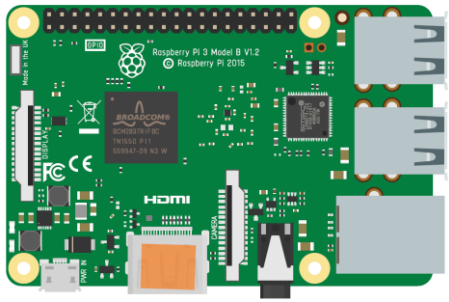
Numbering of GPIO Pins in Grafana

- The GPIO numbers in Grafana correspond to the GPIO pin number to which the sensor node testbed is attached on D-Cube's Observer (Raspberry Pi3)



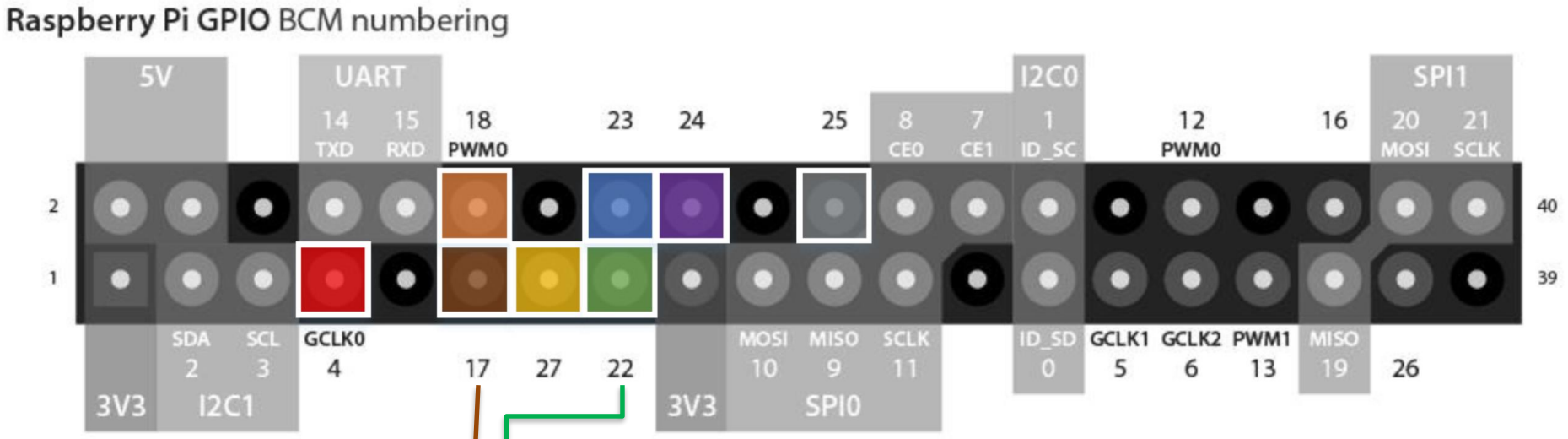
(Pin 22 on Raspberry Pi3)

ADC7/SVSin	GPIO 22



Numbering of GPIO Pins in Grafana

- The GPIO numbers in Grafana correspond to the GPIO pin number to which the sensor node testbed is attached on D-Cube's Observer (Raspberry Pi3)



Sensor Node	Grafana
ADC0	GPIO 17
ADC1	GPIO 4
ADC2/GIO1	GPIO 18

Sensor Node	Grafana
ADC3/GIO0	GPIO 27
ADC7/SVSin	GPIO 22
GIO2	GPIO 23

Sensor Node	Grafana
GIO3/SVSout	GPIO 24
DAC0/ADC6	GPIO 25

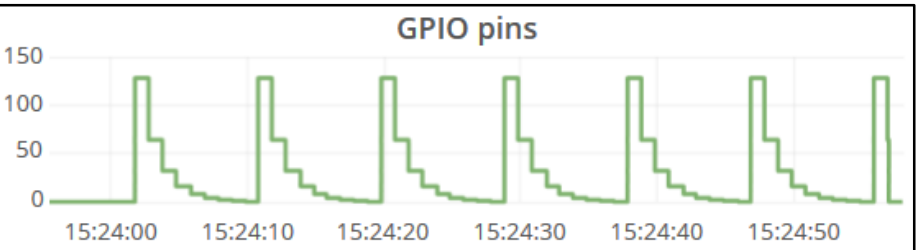
GPIO Pins in Grafana

- In the “Overview of individual nodes” tab, the displayed “GPIO pins” numbers in Grafana is derived with the following mapping:

- Example: “GPIO pins” value of 18
  - 18 = 0001 0010 in binary
  - Using Grafana’s mapping:
  - ADC0=0; ADC1=0; ADC2=0; ADC3=1
  - SVSin=0; GIO2=0; GIO3=1; ADC6=0

```
gpio=0;
gpio=gpioRead(17);
gpio=(gpio<<1) | gpioRead(4);
gpio=(gpio<<1) | gpioRead(18);
gpio=(gpio<<1) | gpioRead(27);
gpio=(gpio<<1) | gpioRead(22);
gpio=(gpio<<1) | gpioRead(23);
gpio=(gpio<<1) | gpioRead(24);
gpio=(gpio<<1) | gpioRead(25);
```

Mapping in Grafana

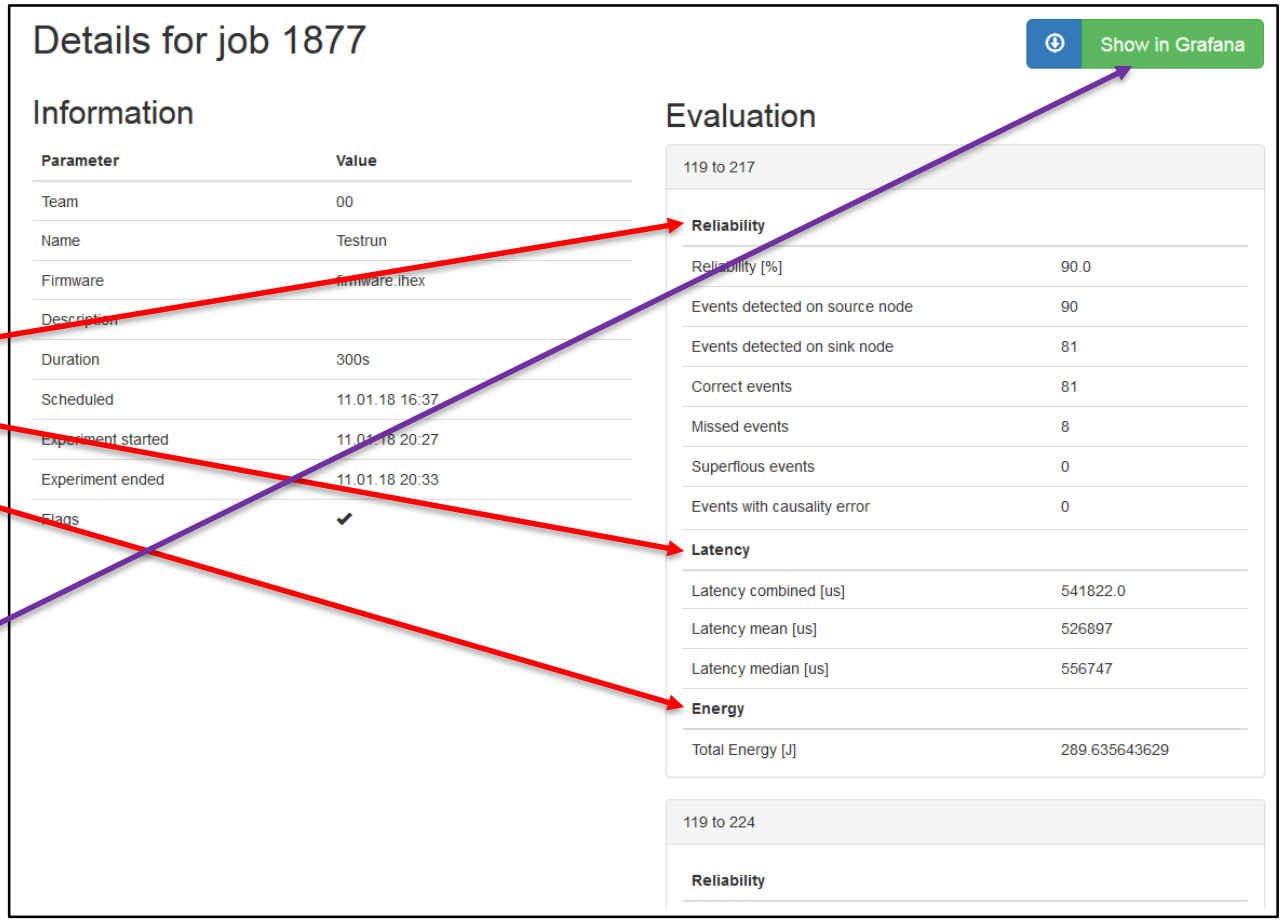


Experiments' Evaluation

Evaluation of Experiments

- In the first days of the preparation phase, this feature will be disabled

- We first let all teams familiarize with the testbed facility
- Detailed evaluation results on reliability, latency, and energy will be displayed after a few days
- Meanwhile, one can see the performance for each run using Grafana



Tentative Agenda

Tentative Agenda

- Preparation phase
  (08.10.2018 – 20.12.2018)

1. Initial preparation phase
   08.10.2018 – 24.10.2018
   - → Teams get access to the competition infrastructure
   - → Teams get acquainted with the new competition settings
     (binary patching and data transmission/reception via EEPROM)
   - → No harsh RF environment yet

NOW!

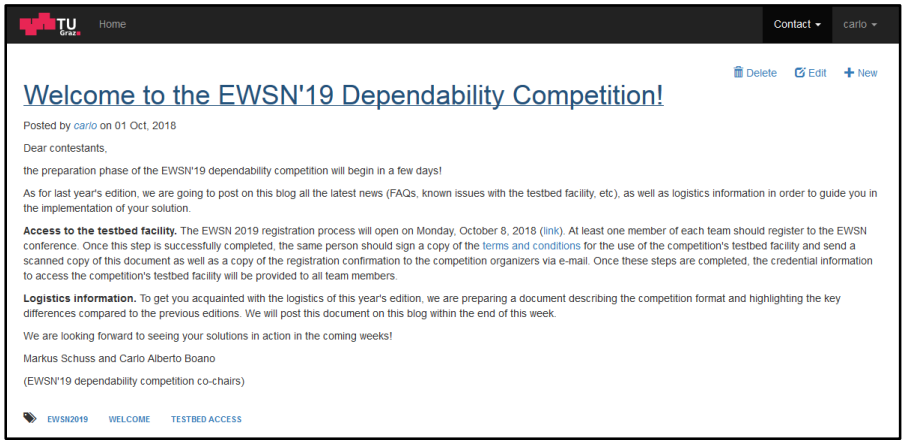
2. Second preparation phase
   25.10.2018 – 11.11.2018
   - → Harsh RF environment can be generated
   - → Leaderboard and detailed evaluations (details follow)

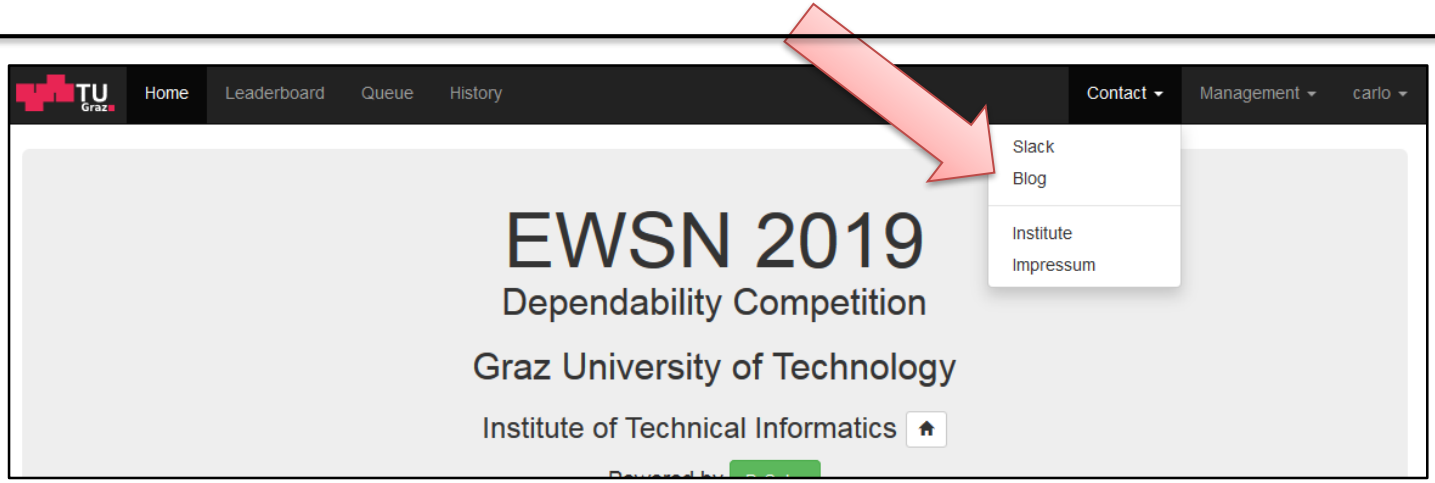
3. Third preparation phase
   12.11.2018 – 20.12.2018
   - → Additional testbed layouts (node placements) available

Communication with the Organizers

Official Blog

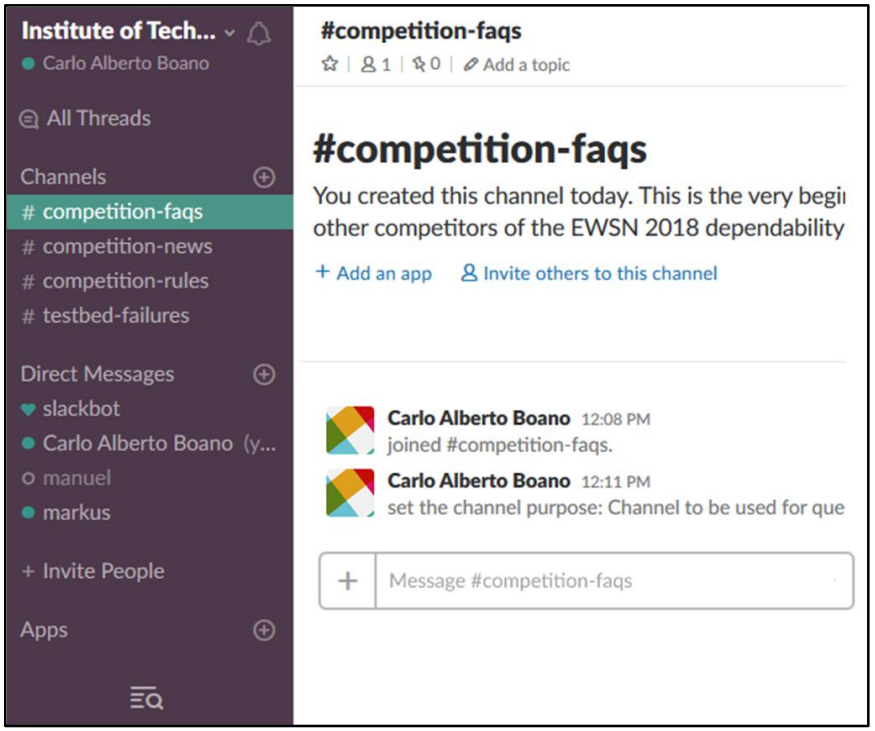
- The organizers have created a blog to keep contestants up to date about the logistics and any important news
  - Please check it regularly!
  - Answers to FAQs will be posted here

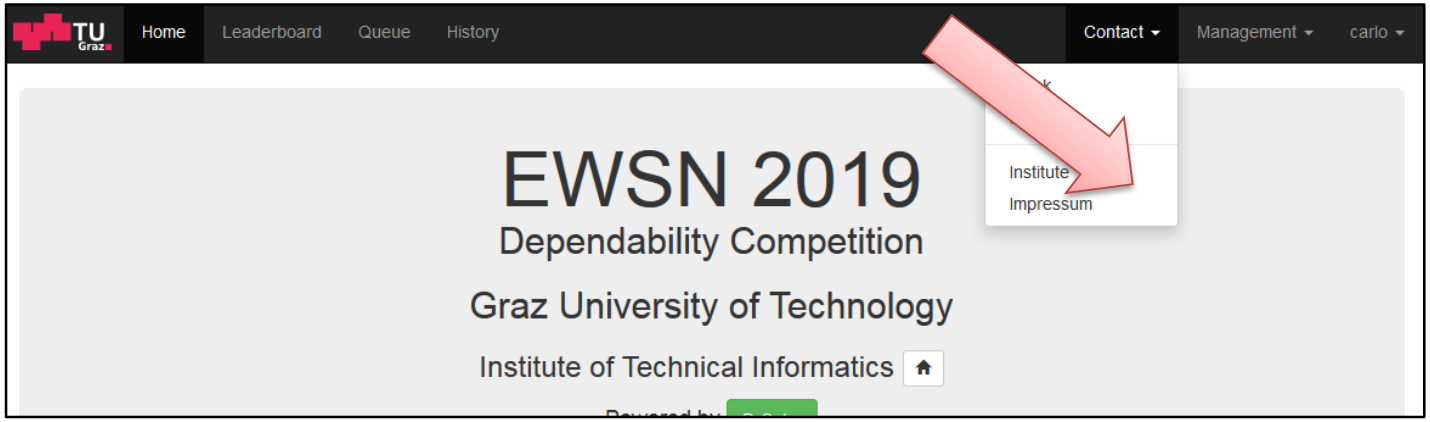




Slack Group

- The organizers have also created a slack group to let contestants easily post questions and interact with the organizers as well as with the other teams
- To join slack, click here





Contacts

- Carlo Alberto Boano
  - E-mail: cboano@tugraz.at
  - Tel.: +43 316 873 6413
- Markus Schuss
  - E-mail: markus.schuss@tugraz.at
  - Tel.: +43 316 873 6403