Specification of DeseNET Protocol

Michael Clausen

Abstract:

The DeseNET protocol is a wireless master-slave oriented real-time data exchange protocol. For the purpose of teaching, the protocol is stripped down to an absolute minimum of functionality in order to be simple to understand and simple to implement by students. DeseNET provides cyclic traffic and event based traffic.

1 DeseNET Concepts

1.1 Sampled Values

Synchronized periodical measurement data

Sampled Values enables measurements on multiple sensors connected to the same DeseNET network. The protocol ensures that all sensors have the time to send their measured values within a DeseNET cycle and measures are synchronized. A DeseNET network supports up to 16 independent Sampled Values streams per Sensor which can be enabled on a cycle base by the DeseNET Gateway. The measurements are synchronized using a Beacon frame send from the Gateway to all Sensors (Broadcast). Every Sensor has to read his inputs related to Sampled Values exactly at the moment the beacon has been received.

The **Gateway** sends a **SV Group Mask** field as part of the **Beacon** frame in order to control which **Sampled Values Groups** have to be acquired immediately after the **Beacon** frame reception. The actual sampled data has to be part of the response send by each **Sensor** .

1.2 Events

A **Sensor** can send a variable amount of **Events** during a DeseNET cycle. Only the actual size left in the wireless frame send by the Sensor limits the number of events possible to send at once. An Event can carry data and up to 16 different events can be defined. Events are part of the sensor's response to the Beacon and broadcasted to all other sensors on the network.

1.3 Static configuration

There is no need for a sensor to explicitly register within the network, the fact that the sensor responds to the beacon adds the sensor automatically to the network. If a sensor does not respond to the beacon for several times, it just can be considered to be offline.

The configuration of the sensors is static and has to be done manually by the network operator. Each sensor gets a unique sensor address and up to 32 sensors can be addressed by one gateway.

1.4 Gateway model

The gateway collects the data of all the individual sensors. Data exchange between sensors or data exchange from the gateway to the sensors is not intended (It would be however possible to add this feature).

2 DeseNET Topology

An DeseNET network consists of the following entities:

- A unique network supervisor called the **Gateway (GW)**
- Multiple (up to 32) Input sensors called **Sensor (SN)**

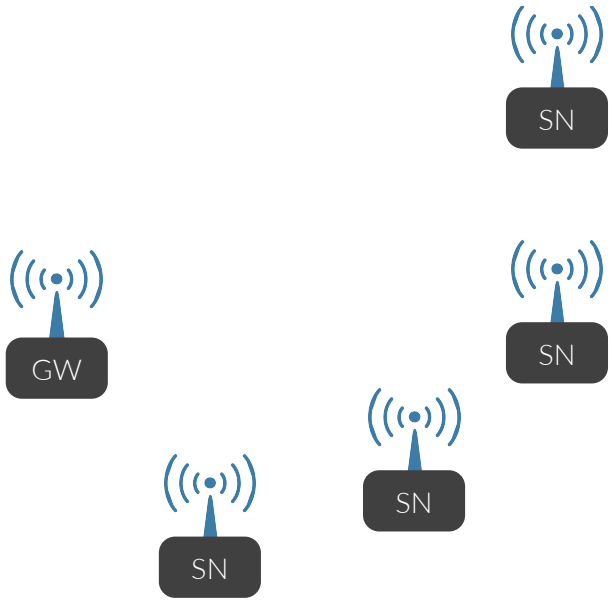


Figure 2.1: DeseNET network topology

The network features a wireless star topology, in order to keep it simple, no mesh networking is possible. So each sensor has to be in the range of the gateway in order to be able to deliver sensor data.

The **Gateway (GW)** coordinates all the wireless traffic on the DeseNET network and synchronizes the local clocks of all connected DeseNET **Sensors (SN)**.

3 DeseNET Mechanisms

A **SN** can run one or more sensor applications on top of the DeseNET stack whereas each application can publish multiple **Sampled Values** streams or publish **Events** . The following figure illustrates this:

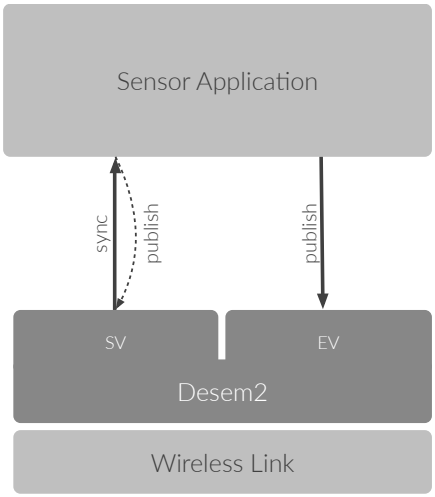


Figure 3.1: DeseNET Layers

3.1 Sampled Values

3.1 Time synchronization and synchronous Input

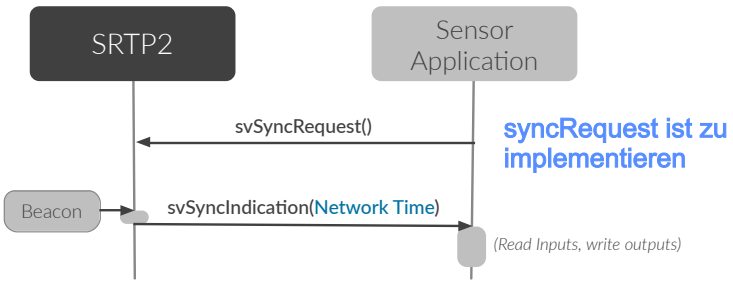


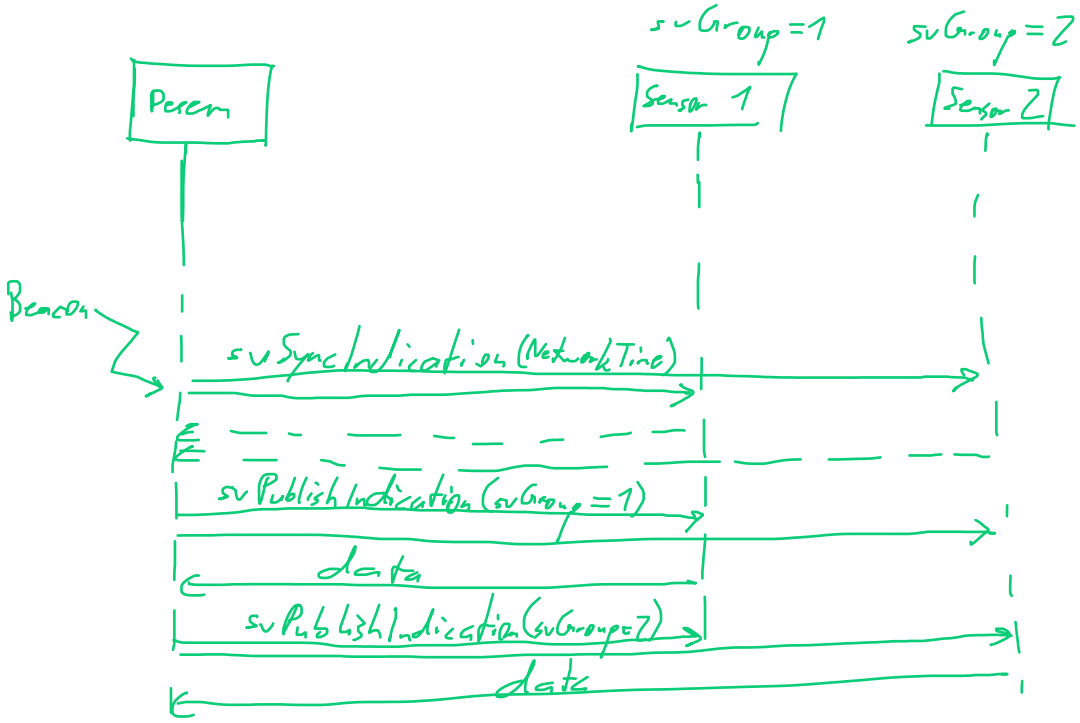
Figure 3.2: Synchronization

An application can register at the DeseNET stack in order to be informed as soon as possible about the arrival of a beacon frame. This enables the system to read all Inputs on the same network at the same moment. As the Gateway sends the actual Network time as payload inside the Beacon, all applications can optionally synchronize their clocks.

in AbstractApplication::svSyncRequest()

Der ptr zum Accelerometer der diese fkt aufruft ist "this"

In onReceive()



3.1 Sampled values publishing

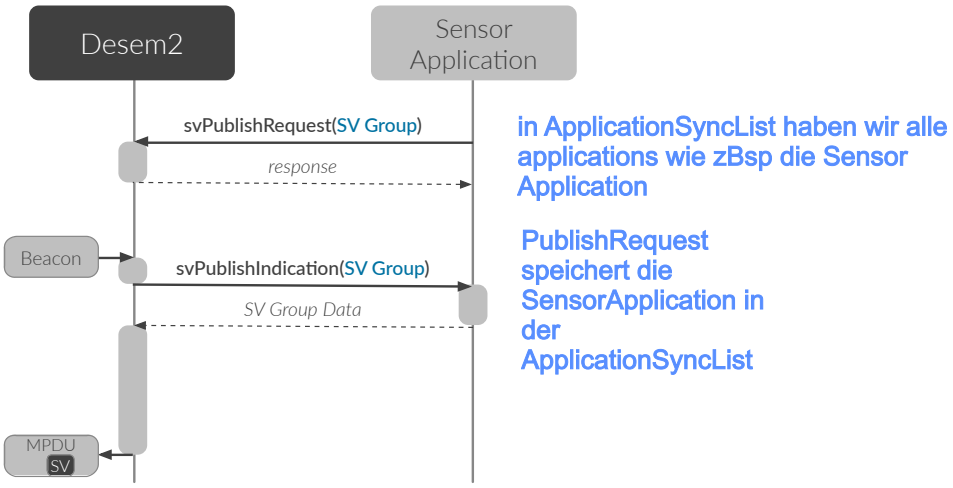


Figure 3.3: SV Publish

Using a **Sampled Values Publish Request** (svPublishRequest) service call, a sensor application can ask the DeseNET stack for the permission to publish a **Sampled Values** stream. Only one sensor application can send a stream to the same **SV Group** at the same time, so if there is already another application registered, the service call should fail.

Once the DeseNET stack receives a Beacon frame, it will notify all registered **Sampled Values publishers** using a **svPublishIndication**. The sensor application has to return the actual data (read during the last **svSyncIndication()**) to send in the **Sampled Values stream**.

3.2 Events

3.2 Event publisher

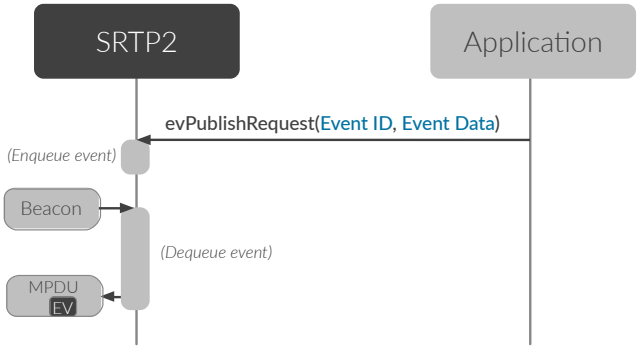


Figure 3.4: Events

A sensor application can publish at any time an event using the **evPublishRequest** service primitive. The sending of the event is not acknowledged.

4 DeseNET Protocol

The **Gateway** sends periodically a so called **Beacon** frame in order to synchronize the Input operations and the time on the network. A **SN** is only allowed to send one **MPDU (Multi-PDU) frame** as a response to the **Beacon** frame. As all **SN** receive the **Beacon** frame at the very same moment, the sequence of the responses from the sensors must be coordinated in order to avoid collisions. This is done by giving each possible sensor on the network a time slot which depends the sensor's ID (0..31), whereas the first sensor can use the first slot and so on.

The following figure shows a typical DeseNET cycle:

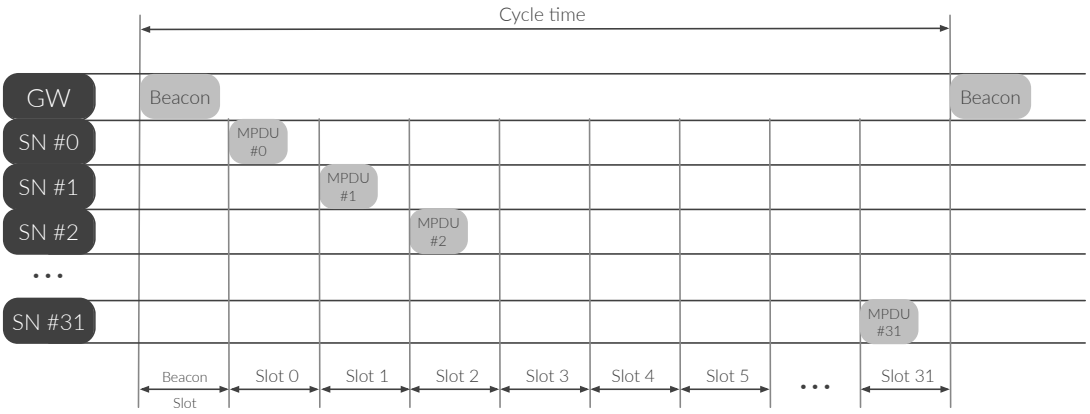


Figure 4.1: DeseNET cycle

Desenet defines only two frame types: **Beacon** and **MPDU (Multi-PDU)** frames. As the name already states, the **MPDU** frame serves as a container for multiple DeseNET embedded PDUs. This minimizes the overhead by allowing the DeseNET stack to group multiple PDUs into one single frame.

All frames are broadcast to all network nodes. This is for Beacon and MPDU frames. If a wireless technology does not support broadcasting, it can not be used as the foundation of theDeseNET protocol. Sensors receiving MPDU frames should just drop them, if their RF receiver is not deactivated anyway, as they can go to deep sleep after responding to the Gateway's Beacon.

4.1 Beacon Frame

The **Beacon** frame is send periodically from the **Gateway** to a all **Sensors** (Broadcast). It can be seen as the Heartbeat of the DeseNET network. **Sensors** are only allowed to send a frame as a response to the **Beacon** frame.

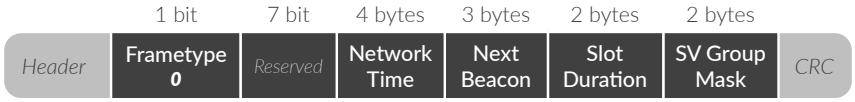


Figure 4.2: Beacon frame

All multibyte data is encoded in the **little endian** format. This is just for performance reasons, as ARM and Intel processors use little endian per default. For the fields smaller than actually 8 bits, the most significant bit is taken first.

Header

Might or might not be added by the actual wireless transmission technology. In any case it will be ignored by the DeseNET protocol.

Frametype

1 Bit unsigned integer, valid values: [0, 1].

Frametype defines the type of the DeseNET frame. There are only two types of frames:

- **0** : Beacon frame
- **1** : MPDU frame.

Network Time

4 bytes unsigned integer, represents Milliseconds since midnight of the current day.

Network Time is the actual time on the **Gateway** in milliseconds since midnight. This time representation has been chosen in order to keep the timestamp fields at a minimum size. The gateway will add the current date's UNIX epoch milliseconds (64 bit value, Epoch milliseconds to midnight) to the value in order to create the correct timestamp. All **Sensors** have to synchronize their local clocks to this value.

Next Beacon

3 bytes unsigned integer, represents Milliseconds.

Next Beacon is the number of milliseconds until the next beacon is send. Note that this time interval is between the start of the two beacons.

Slot duration

2 bytes unsigned integer, represents Milliseconds.

Slot duration is the actual size of a single of the 32 slots in milliseconds. A Sensor with the ID **n** has to wait **(n+1)*Slot_duration** before actually sending his response. Note that the time starts at the beginning of the Beacon frame, that is why you have to add 1 to the Sensor ID.

SV Group

16 bits boolean mask, each bit represents a SV Group.

The **SV Group Mask** field is a bit-mask where each bit represents a **SV Group**. If the corresponding bit is set, the Sensors have to sample the input and include the data in their response to the **Beacon**, otherwise not. As the **SV Group Mask** field is 16 bits in size, 16 SV groups are defined whereas bit 0 of the field represents group 0 and bit 15 group 15.

CRC

The **CRC** (Cyclic Redundancy Check) might be added and checked by the wireless transmission hardware.

4.2 MPDU Frame

The **MPDU** frame is send by all **Sensors** as a response to the **Beacon** frame from the **Gateway**. Such a frame contains multiple embedded PDUs. This ensures that as much payload as possible can be send at once in one frame. This eliminates unnecessary overhead by sending a lot of small frames and increases the overall reactivity of the system.

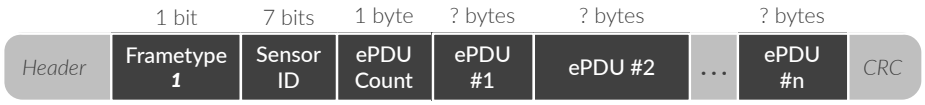


Figure 4.3: MPDU frame

All multibyte data is encoded in **little endian** format. This is just for performance reasons, as ARM and Intel processors use little endian per default. For the fields smaller than actually 8 bits, the most significant bit is taken first.

Header

Might or might not be added by the actual wireless transmission technology. In any case it will be ignored by the DeseNET protocol.

Frametype

1 Bit unsigned integer, valid values: [0, 1].

Frametype defines the type of the DeseNET frame. There are only two types of frames:

- **0** : Beacon frame
- **1** : MPDU frame.

Sensor ID

7 Bit unsigned integer, valid values: [0, 31].

Id of the sensor actually sending the MPDU frame. Each ID should only be present once in the network. The ID is configured statically on the Sensors.

ePDU Count

1 byte unsigned integer [0, 255].

ePDU Count is the number of ePDUs contained in the MPDU. This field is followed by the actual ePDUs.

CRC

The **CRC** (Cyclic Redundancy Check) might be added and checked by the wireless transmission hardware.

4.3 SV ePDU

Sampled Values embedded PDU.

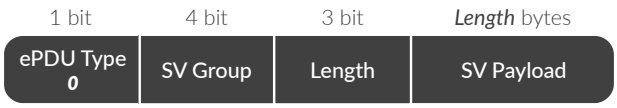


Figure 4.4: SV PDU

All multibyte data is encoded in **little endian** format. This is just for performance reasons, as ARM and Intel processors use little endian per default. For the fields smaller than actually 8 bits, the most significant bit is taken first.

ePDU Type

1 bit unsigned integer [0, 1].

The **ePDU Type** determines the type of the embedded PDU. It is always **0** for the SV ePDU.

SV Group

4 bit unsigned integer, valid values in range [0, 15].

The **SV Group** field determines from which SV group the subsequent payload comes. Note that this is not a mask anymore, it is the number of the SV Group. This value should be unique on a network, so if multiple applications run on the same network, it has to be ensured that there are no such conflicts.

Length

3 bits unsigned integer [0, 7].

Length is the actual length of the Sampled Values payload in bytes. The maximal size of data is 7 bytes.

SV Payload

Length bytes binary data.

The field **SV Payload** contains the actual data of the Sampled Values ePDU. The format of this buffer is not defined at all and completely application specific.

4.4 EV ePDU

Event embedded PDU.

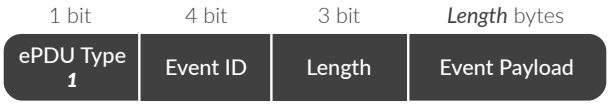


Figure 4.5: EV PDU

All multibyte data is encoded in **little endian** format. This is just for performance reasons, as ARM and Intel processors use little endian per default. For the fields smaller than actually 8 bits, the most significant bit is taken first.

ePDU Type

1 bit unsigned integer [0, 1].

The **ePDU Type** determines the type of the embedded PDU. It is always **1** for the EV ePDU.

Event ID

4 bits unsigned integer [0, 15].

Event ID is the actual identifier of the Event. This value should be unique on a network, so if multiple applications run on the same network, it has to be ensured that there are no conflicts.

Length

3 bits unsigned integer [0, 7].

Length is the actual length of the Event payload in bytes. The maximal size of data is 7 bytes.

Event Payload

Length bytes binary data.

The field **Event Payload** contains the actual data of the Event ePDU. The format of this buffer is not defined at all and application specific.