In this handout we give some idea of how meta-code can be used to build task-specific WSN configurations. All examples are written in Meta-Lang, the assembler-like language of meta-code. We implement a simple data gathering application with the following set of features: spanning-tree-based, unique ID assigned for each node, 1-min1 measurements are collected, delivered to the top of the spanning tree and stored into the buffer. Hence, we cover the following layers of the WSN network stack: routing, data and application processing. In this example we assume that MAC-layer is provided. Time synchronization (e.g. LTS spanning-tree based time-sync algorithm) can be easily added with minimal changes needed to the presented capsules.

The first capsule we use to build a classic spanning tree (see Listing 1). This capsule must periodically (10s) flood the network and be executed on each node to reflect the changes in the tree structure.

Listing 1: Spanning tree construction

```
; SYSTEM segment
nys AUTOUPDATE 0         /* disable autoupdate (capsules of the same version will be accepted, capsules of lower versions will be declined) */
LIFETIME 10s             /* recognized post-fuse: mS (milliseconds), s (sec), ms (millisecond) */
ID 0x                    /* (packets) */
bufc 0                   /* DATA segment (allocated inside the capsule) */
from=0                   /* "S" is some real network address */
hops=0                   /* local variable */
code.init                /* CODE segment "init" (executed once) */
inc hops                 /* (B << 16) + A (32-bit result) */
add                      /* compute A */
comp_coef                 /* must not be zero */
m_z = sense_temp          /* choose initializer for B (or A) */
m_w = mult 18000          /* compute A */
and 65535                 /* 32-bit result */
push BUFS[0]              /* A = 18000 * (A & 65535) + (A >> 16) */
lsh 16                    /* (B << 16) + A (32-bit result) */
add                      /* compute A */
m_w = mult 36969          /* compute A */
and 65535                 /* 32-bit result */
push BUFS[0]              /* A = 36969 * (A & 65535) + (A >> 16) */
```

The second capsule performs automatic node id assignment based on a measured temperature value (can be humidity, or any other available 16-bit sensor, or a mix) and a simple pseudo-random number generator shown in Listing 2 below:

```
/* 32-bit result */
push BUFS[0]              /* "PACK.DST" fixed, "PACK.SRC" and "PACK.DST" change at each hop */
jalp s,14                 /* process packets addressed to S */
exiit                     /* exit point (the capsule stays alive) */
sense_temp                /* clean up the top part */
pop BUFS[0]               /* store the new ID in the capsule */
sense_temp                /* choose initializer for B (or A) */
exiit                     /* exit the spanning tree */
```

Listing 2: "Multiply-With-Carry" random number generator of G. Marsaglia

```
/* 32-bit result */
push BUFS[0]              /* "PACK.DST" fixed, "PACK.SRC" and "PACK.DST" change at each hop */
jalp s,14                 /* process packets addressed to S */
exiit                     /* exit point (the capsule stays alive) */
sense_temp                /* clean up the top part */
pop BUFS[0]               /* store the new ID in the capsule */
sense_temp                /* choose initializer for B (or A) */
exiit                     /* exit the spanning tree */
```

Listing 3: Automatic node id-assignment

```
/* 32-bit result */
push BUFS[0]              /* "PACK.DST" fixed, "PACK.SRC" and "PACK.DST" change at each hop */
jalp s,14                 /* process packets addressed to S */
exiit                     /* exit point (the capsule stays alive) */
sense_temp                /* clean up the top part */
pop BUFS[0]               /* store the new ID in the capsule */
sense_temp                /* choose initializer for B (or A) */
exiit                     /* exit the spanning tree */
```

The following example shows how program size can be reduced by making calls to macro-instructions (this is supposed to replace a part of the original code on lines 7-24 in Listing 3):

```
/* 32-bit result */
push BUFS[0]              /* compute A */
jalp s,14                 /* process packets addressed to S */
exiit                     /* exit point (the capsule stays alive) */
sense_temp                /* clean up the top part */
pop BUFS[0]               /* store the new ID in the capsule */
sense_temp                /* choose initializer for B (or A) */
exiit                     /* exit the spanning tree */
```

The macro-instruction "comp_coef" shown above is stored in the on-board instruction dictionary and defines the following sequence of simpler operations:

```
/* 32-bit result */
push BUFS[0]              /* compute A */
jalp s,14                 /* process packets addressed to S */
exiit                     /* exit point (the capsule stays alive) */
sense_temp                /* clean up the top part */
pop BUFS[0]               /* store the new ID in the capsule */
sense_temp                /* choose initializer for B (or A) */
exiit                     /* exit the spanning tree */
```

This capsule is executed once during the initialization phase on each node (see Listing 3).

Copyright is held by the author/owner(s).
ACM 978-1-60558-955-8/10/04.

1This is just a sensing interval; no time-sync is used; measurements are not timestamped.
The code above lines 3 and 5 show that macro-instructions can be nested.

In the code above, lines 3 and 5 show that macro-instructions can be nested.

The next two capsules are responsible for taking and collecting measurements.

The algorithm assumes that we already have an established tree topology in the network. The capsule periodically initiates a measurement on each node, accumulates a buffer of 10 measurements and sends it back up to the top of the spanning tree (see Listing 4). The sink node does not execute this capsule.

Listing 4: Sense and send measurements to the sink

Line 10 above is an example of using code polymorphism.

The last capsule resides on the sink node, receives measurements from different nodes and stores them into the buffer (see Listing 5).

Listing 5: Collect and store measurements in the sink's buffer

Typically, sensor network applications work based on principles shown above (spanning tree, periodical sampling, etc). But what if we need to make two nodes communicate to each other? In this case the following MANET-like route discovery scheme may be useful:

Listing 6: MANET-like route discovery

The examples above can be further improved and customized to meet the requirements of a specific application.