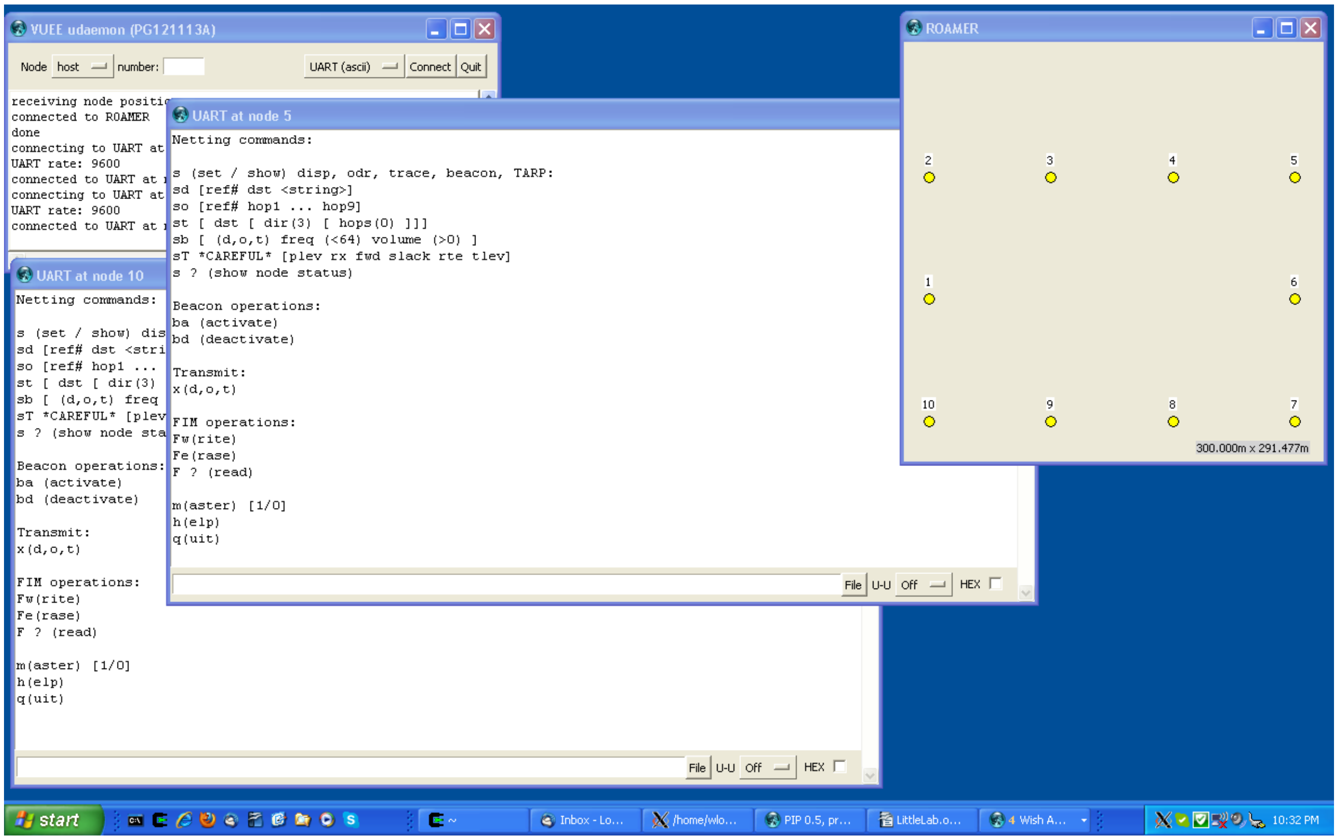
The 'Netting' praxis is designed to explore our ad-hoc networking, including operator-directed routing (ODR) which defies ad-hoc'ness but can serve as a good comparison and measurement tool.

No remote configurations. Although very limiting, it simplifies operations, especially for novice users. The mode of operations here is “set & go”, meaning that certain structures can be set and repeatedly executed, with or without operator involvement. This should be helpful for long experiments, comparisons, performance assessments, etc.

This version has very simple serial interface (9600bps, 8, N, 1, N) for *piter* or any serial terminal. Output formatted for human eyes.

For those running VUEE:
Most of the numbers in the examples below come from a VUEE run (of the rec10.xml model), some differ from real hardware. Commands were issued on node 10.



```
receiving node positio
connected to ROAMER
done
connecting to UART at
UART rate: 9600
connected to UART at n
connecting to UART at
UART rate: 9600
connected to UART at n
```

UART at node 5

```
Netting commands:

s (set / show) disp, odr, trace, beacon, TARP:
sd [ref# dst <string>]
so [ref# hop1 ... hop9]
st [ dst [ dir(3) [ hops(0) ]]]
sb [ (d,o,t) freq (<64) volume (>0) ]
sT *CAREFUL* [plev rx fwd slack rte tlev]
s ? (show node status)

Beacon operations:
ba (activate)
bd (deactivate)

Transmit:
x(d,o,t)

FIM operations:
Fw(rite)
Fe(rase)
F ? (read)

m(aster) [1/0]
h(elp)
q(uit)
```

UART at node 10

```
Netting commands:

s (set / show) dis
sd [ref# dst <stri
so [ref# hop1 ...
st [ dst [ dir(3)
sb [ (d,o,t) freq
sT *CAREFUL* [plev
s ? (show node sta

Beacon operations:
ba (activate)
bd (deactivate)

Transmit:
x(d,o,t)

FIM operations:
Fw(rite)
Fe(rase)
F ? (read)

m(aster) [1/0]
h(elp)
q(uit)
```

At boot:

```
PicOS v3.6/PG121115A-WARSAW, (C) Olsonet Communications, 2002-2012
Leftover RAM: 9222 bytes
CC1100: 1, 842.3MHz, 0/200kHz=842.3MHz
Netting commands:
```

```
s (set / show) disp, odr, trace, beacon, TARP:
sd [ref# dst <string>]
so [ref# hop1 ... hop9]
st [ dst [ dir(3) [ hops(0) ]]]
sb [ (d,o,t) freq (<64) volume (>0) ]
sT *CAREFUL* [plev rx fwd slack rte tlev]
s (show node status)

Beacon operations:
ba (activate)
bd (deactivate)

Transmit:
x(d,o,t)

FIM operations:
Fw(rite)
Fe(rase)
F (read)

m(aster) [1/0]
h(elp)
q(uit)
```

Note that the RF is configured for European ISM. Let's explore the commands via examples:

s
```
10 at 310: M 0 at 310 mem 4292.4050.78 bat 2600
```
Status / statistics: Node **10** is at **310** uptime, no master (**0**), there is **4292** words RAM available, so far minimum has been **4050**, the min. stack was **78** words. Battery level is **2600** (it should be replace around 2200).

sd
```
Disp #0 to 0 ret 0 len 0 <>
```
Show *disp* msg: not set.

sd 77 5 1 abracadabra no spaces
```
Disp #77 to 5 ret 1 len 11 <abracadabra>
```
Set disp: ref# 77, to be sent to node 5, we want an ack (ret 1), the string follows, up to the 1st space. Note that in all commands the to-node id =0 means broadcast.

so
```
Odr(0) #0 0 0 0 0 0 0 0 0 0
```
show *ODR*: not set

so 66 1 2 3 4 5 6
```
Odr(6) #66 1 2 3 4 5 6 0 0 0
```
Set ODR: ref#**66** followed by the list of 6 node ids (**1 2 3 4 5 6**). Note that (**6**) is useful if we use zero as the last element of the list.

st
```
Trace to 0 dir 77 hco 0
```
Show trace: not set. Note that 77 show that (sorry about it), as
st 0 0 0
```
Trace to 0 dir 0 hco 0
```
is a valid set: flood the network (0 as dst) with no hop limit (hco 0) to request trace back from all nodes.

st 5 3 0
```
Trace to 5 dir 3 hco 0
```
This is more reasonable: bi-directional trace to node 5, no hop limit.
Dir 0 – back-trace; 1 – proxy fwd (displayed at dst); 2 – forward-trace (displayed at the issuing source), 3- bi-directional.

sb
```
Beac 0(0) freq 0 0 of 0
```
Show beacon: not set.

sb d 10 3
```
Beac d(1) freq 10 0 of 3
```
Set beacon: **d**isp msg (*msg_type **1***) in **10**s intervals **3** times. Visible *msg_type* helps with traces, as their quite arcane details involve multiple msg types. In most cases, we ignore it. Note that *sb* to an unset message (d,o,t) results in the '*not set*' reply.

sT
```
Plev 7 rx 1 fwd 1 slack 1 rte 2 tlev 2
```
sT (show/set TARP) includes power level, RF receiver on/off and four of TARP parameters. We strongly discourage non-expert users from using this command. For example, power level may be useful to shorten RF range, but if misconfigured, it may generate asymmetrical 'links' that are difficult to spot and debug.

sT 0
```
Plev 0 rx 1 fwd 1 slack 1 rte 2 tlev 2
```

If the changed sT is to be preserved, FIM operations can be used:
F
```
FIM(128): plev 7 rx 1 fwd 1 slack 1 rte 2 tlev 2
```
Show FIM: virgin. The **128** (or 256) is the size of FIM. (We write all over FIM to minimize erases, and show the slot# as yet another rarely relevant implementation detail.)

Fw
```
FIM(0): plev 0 rx 1 fwd 1 slack 1 rte 2 tlev 2
```
sT was written and the new values will be used also after reboot (power cycle).

sT 7
```
Plev 7 rx 1 fwd 1 slack 1 rte 2 tlev 2
```

Fw
```
FIM(1): plev 7 rx 1 fwd 1 slack 1 rte 2 tlev 2
```

Fe
```
FIM(128): plev 7 rx 1 fwd 1 slack 1 rte 2 tlev 2
```
Forces 'factory defaults' after:
q
Reboot.

h
Help shows the commands, as at boot time.

xd
```
Disp #77 to 5 ret 1 len 11 <abracadabra>
5052(60): disp fr 5.4 #77 ret 0 11<abracadabra>
```

and, on node 5:
```
5052(60): disp fr 10.4 #77 ret 1 11<abracadabra>
```
Note the ret change: on node 5, it is still 1, back it is 0. The timestamp 5052 is also shown mod 128 (60) to easy calculations in some advanced scenarios, e.g. round trip time with fuzzy acks (and possible retries at every hop) or LBT in heavy congestion. When ref# is set to 0 and the message is on beacon, it is replaced with uptime mod 128. The ODR example at the end of this document illustrates the intentions.

xo
```
Odr(6) #66 1 2 3 4 5 6 0 0 0
5689(57): odr(66) [1.0.6]:
 0:  (10 0 105)<
 1:  (1 103 105)
 2:  (2 103 105)
 3:  (3 103 102)
 4:  (4 104 105)
 5:  (5 104 105)
 6:  (6 98 0)
```
ODR line, as in ***so*** command, appears if there is no active ODR beacon. Otherwise ***sb*** line is shown, see ***ba*** below.

on node 5:
```
5689(57): odr(66) [0.5.6]:
 0:  (10 0 0)
 1:  (1 103 0)
 2:  (2 103 0)
 3:  (3 103 0)
 4:  (4 104 0)
 5: >(5 104 0)
 6:  (6 0 0)
```
This is on the way from node 10 to 6, note **>** (that's *I Node 5*). So, hop# **5**: node id **5**, forward RSSI **104**, and all other available data travelling in the packet. Very soon (the same second), I'm hit again on the way back:
```
5689(57): odr(66) [1.5.6]:
 0:  (10 0 0)
 1:  (1 103 0)
 2:  (2 103 0)
 3:  (3 103 0)
 4:  (4 104 0)
 5:  (5 104 105)<
 6:  (6 98 0)
```

On **every** intermediate node,similar reports are displayed. (It mutilates VUEE, so we're likely to add a flag to switch these off.) It is a list of all node ids with forward and backward RSSIs. In [] brackets are directions (ret=0 means forward, **1** backward) hop counter (up or down) =**5**, and hop 'range' **6**.

ODR and *trace* have hidden values, think about site surveys or deployment checks.

```
xt
Trace to 5 dir 3 hco 0
6388: tr(5) 6 4 5:
 1 105
 3 100
 4 105
 5 103
 6 103
 7 103
 8 104
 9 104
 10 102
```

After ***xt***, the ***st*** line appears. After the packet is back at source, the rest is displayed: **6388** timestamp (note that there is no ref# in trace), **(5)** is the dst node that send the trace back (not that obvious in broadcast traces) *msg_type* **6** happens to correspond to reply to bi-directional request (**dir 3**). There were **4** hops forward and **5** back... note that this would not be possible if TARP's slack were 0 and the packet overshot from **1** to **3** as above. Another trace was symmetrical:

```
xt
Trace to 5 dir 3 hco 0
6402: tr(5) 6 5 5:
 9 105
 8 104
 7 102
 6 105
 5 105
 6 105
 7 103
 8 105
 9 104
 10 103
```

Another one shows true ad-hoc'ness:

```
xt
Trace to 5 dir 3 hco 0
140: tr(5) 6 5 4:
 1 105
 2 104
 3 103
 4 104
 5 104
 6 104
 8 98
 9 104
 10 104
```

Unidirectional traces have their moments, too, but we're stretching a novice reader's patience here.

Master functionality is meant to experiment with clusters with sink nodes. Master node broadcasts the master beacon (independent from the settable '*message beacon'*) every 30s randomized up to +/- 1s. All nodes set their master node id (which is not directly used in *netting* functionality) and prepare their caches for preferential routing. Any node can become master at any time, sometimes triggering short races for domination. After each m command, the node status is displayed:

```
m
10 at 27: M 0 at 0 mem 4292.4052.256 bat 2600
m1
10 at 32: M 10 at 32 mem 4280.4052.256 bat 2600
m 0
10 at 42: M 0 at 42 mem 4292.4052.256 bat 2600
m1
```

from node 5 perspective:

```
m
5 at 45: M 10 at 41 mem 4292.4052.256 bat 2600
m1
5 at 75: M 5 at 75 mem 4280.4052.256 bat 2600
```

Node 10 peacefully abdicated:

```
m
10 at 92: M 5 at 75 mem 4292.4052.256 bat 2600
```

When master beacon is absent for 3 times the prescribed frequency (about 90s here), the node becomes master-less (*ronin*) again. Note that the *ronin* status can be at any time forced by *m0* but it will work only up to the next master beacon arriving to the node.

Message beacon automates longer runs, e.g. to gather data for PDF, average delays, etc.

```
sd 0 5 1 pls_help!!!_hilfe!!!
Disp #0 to 5 ret 1 len 20 <pls_help!!!_hilfe!!!>
```
We did ret 1 to ensure the receipt (round trip).

```
sb d 20 5
```

```
Beac d(1) freq 20 0 of 5
```
ba
```
Beac d(1) freq 20 2 of 5
2299(123): disp fr 5.4 #123 ret 0 20<pls_help!!!_hilfe!!!>
2319(15): disp fr 5.5 #15 ret 0 20<pls_help!!!_hilfe!!!>
```
Note the ref#0 replaced with timestamps mod 128, directly comparable with the arrival time.

xd
```
2336(32): disp fr 5.4 #32 ret 0 20<pls_help!!!_hilfe!!!>
Beac d(1) freq 20 4 of 5
```
This xd (xmit display) issued while disp was on an active beacon cuts into the beacon interval, triggering immediate transmission. Here, it was at 'time 32' instead of 'regular 35'. It was the 4th out of 5 requested transmissions.

bd
```
Beac d(1) freq 20 0 of 5
```
Deactivate the beacon.

As already mentioned, the message beacon allows unattended (logged) operations and diverse stress-tests. All becomes clear with a bit of practise; here we'd like to underline the mechanism of ref# 0 morphing to timestamps on the beacon. For example, consider this ping-pong, this time from real hardware (2 nodes next to each other):

ba
```
Odr(9) #0 10 1 10 1 10 1 10 1 10
Beac o(2) freq 5 0 of 10

597(85): odr(84) [0.2.9]:        (* happened to be on a second boarder *)
 0:  (1 0 0)
 1:  (10 235 0)
 2: >(1 236 0)
 3:  (10 0 0)
 4:  (1 0 0)
 5:  (10 0 0)
 6:  (1 0 0)
 7:  (10 0 0)
 8:  (1 0 0)
 9:  (10 0 0)
597(85): odr(84) [0.4.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4: >(1 236 0)
 5:  (10 0 0)
 6:  (1 0 0)
 7:  (10 0 0)
 8:  (1 0 0)
 9:  (10 0 0)
```

```
597(85): odr(84) [0.6.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4:  (1 236 0)
 5:  (10 235 0)
 6: >(1 235 0)
 7:  (10 0 0)
 8:  (1 0 0)
 9:  (10 0 0)
597(85): odr(84) [0.8.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4:  (1 236 0)
 5:  (10 235 0)
 6:  (1 235 0)
 7:  (10 235 0)
 8: >(1 236 0)
 9:  (10 0 0)
598(86): odr(84) [1.8.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4:  (1 236 0)
 5:  (10 235 0)
 6:  (1 235 0)
 7:  (10 235 0)
 8:  (1 236 235)<       (* from the 2nd hop fwd to the 1st back (8 hops) it took 1s *)
 9:  (10 235 0)
598(86): odr(84) [1.6.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4:  (1 236 0)
 5:  (10 235 0)
 6:  (1 235 236)<
 7:  (10 235 235)
 8:  (1 236 235)
 9:  (10 235 0)
598(86): odr(84) [1.4.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 0)
 3:  (10 235 0)
 4:  (1 236 236)<
 5:  (10 235 235)
 6:  (1 235 236)
 7:  (10 235 235)
 8:  (1 236 235)
 9:  (10 235 0)
598(86): odr(84) [1.2.9]:
 0:  (1 0 0)
 1:  (10 235 0)
 2:  (1 236 236)<
 3:  (10 235 235)
 4:  (1 236 236)
 5:  (10 235 235)
 6:  (1 235 236)
 7:  (10 235 235)
 8:  (1 236 235)
 9:  (10 235 0)
599(87): odr(84) [1.0.9]:        (* another 8 hops in 1s, very consistent *)
 0:  (1 0 235)<
 1:  (10 235 235)
 2:  (1 236 236)
 3:  (10 235 235)
 4:  (1 236 236)
 5:  (10 235 235)
 6:  (1 235 236)
 7:  (10 235 235)
 8:  (1 236 235)
 9:  (10 235 0)
```