olsonet Communications

# Seawolfie

demonstration

Technical Note

Version 0.4
July 2008

TOC

## Introduction

This technical note outlines the *Seawolfie*[1] demonstration prepared as a 'variation on the theme' of the requirements from Combat Networks. The requirements and the software sketch that implements them should be viewed only as a rough articulation of relevant fragments of the *Seawolf* project. Hopefully, a posteriori they'll be seen as seminal works leading to a successful application. Therefore, this note is intended only at audience active within the *Seawolf* project.

Although we're scratching the overlapping areas of ad-hoc communities, mobile groups and anonymous security enforcement, we refrain from digressions from our focus. Appearance of the *noombuzz* site anchors the functionality to the specific setup and should help to envision crucial details and inspire additional functionality. That's the main goal of the *Seawolfie* demonstration: aid in articulation and implementation of hardware and software details of the first version of the functional *seawolfie* device in particular and the *Seawolf* application in general.

The *Requirements* section has come from Combat Networks.

The *Functionality* section is a dry outline, likely difficult to read prior to the actual demonstration.

The *Demonstration* section studies a functional demonstration. As a side effect, we present our *Virtual Underlay Emulation Engine (VUEE)* in action, touching on its tremendous potential as a development and maintenance tool.

The *Comments & thoughts* section has a good title; since nothing is firm yet, we share a few points that may be not that evident at first sight.

This document is evolving and its form is somewhat inconsistent; chronological rather than based on merits. *Noombuzz* is a case in point: this pivotal functionality is a mere add-on in this document.

## Requirements

a) Identify other devices in the area (30-50 ft)
b) Read other devices memory to determine commonalities in each other's databases
c) Database information (database includes First name, picture (in future versions), and 3-5 user selectable "passions")
d) Signal both hardware devices (vibrate, light, sound) that a commonality has been found
e) Allow users to accept or reject the offer to share information
f) Upon Acceptance allow users to view some database information (first name, picture, area of commonality)
g) Upon Rejection, send sorry I'm busy message
h) Allow users to grant other information in the database to be transmitted (future version)
i) Indicate direction and approximate distance between devices and display as a compass or directional arrow on screen (future version)

[1]Previous version (v. 0.3) of this note was titled "*Matchmaker demonstration*".

## Functionality

**Although** implemented functionality approximates the requirements, it truly is a variation on the theme. We don't strictly follow the requirements for these important reasons:

- Somewhat *reversed* approach highlights the process of forming ad-hoc communities, where members remain anonymous until they decide to reveal their identities and lift privacy to a selectable degree, within a selected group. Note that noombuzz-based provisioning moves part of this functionality elsewhere, but the ultimate choice is still in the user's hands.
- Although the main functionality requires only proximity communication, we do believe multi-hopping networking sooner or later will stand out as an attractive differentiator.

**The setup**: a large multi-threaded conference. At the registration, participants receive badge-like devices, in shape and weight comparable to secure ids... This would be ideal, but likely not that feasible. The devices have a rustic user interface (UI): 2-3 -line LCD, RGB LED, a couple of buttons. The device has a mini-USB interface. All public PCs at the conference run an applet that is able to communicate with the badge via USB, temporarily extending the UI. Also, USB Bluetooth dongles are available for rent. They enable the extended UI not only on the PCs, but also on PDAs, cell handsets, mobile tablets, etc. Free applets are available for popular mobile platforms.

Graphical LCD became a hard requirement for the envisioned functionality. It is not clear yet if it'll be the only LCD, not if it can be accommodated on devices derived from our experimental hardware (the EMSPCC11 node).

**Each device** has a unique and constant identifier. The devices are flashed with data provided by participants prior to the conference, directly on the *noombuzz* site, or indirectly via the conference organizers. All is settable through the extended interface at any time, but the organizers may want to set desired defaults. This is important, as clever and unobtrusive advertising of local businesses may be a good incentive to install and maintain such a system.

The attributes involved in matchmaking:

- **Nickname**: a brief and mnemonic identifier (e.g. first name).
- **Description**: Ideally, it depicts appearance, a prominent feature, etc.
- **Business description**: Business card content, main research interest, employer's profile.
- P**rivate description**: Hobby, spontaneous desire, private contact data.
- **Alarm description**: Unlikely to be set a priori, it enables multi-hopping communication within already established communities.
- **Profile**: List of several well defined interests, features, topics, local venues. For the demo the attributes are binary (interested / applicable, or not).
- **Include**: A match is attempted between the receiver's *Include* and sender's *Profile*, so at any time *Include* should describe current preferences for follow-up interactions.
- **Exclude**: All incoming *Profiles* are discarded if matched against *Exclude*.
- **Noombuzz list**: The list of ids, dispositions, and additional information downloaded from the *noombuzz* site. The ids on the list are handled according to the associated disposition without further consultations.

- **Monitor list**: The list of ids the user wants to communicate with, not necessarily when in the range. In other words, this is the list of members of established communities the user is a part of. Note that these are ad hoc communities formed at the event, not necessarily related to *noombuzz* communities.
- **Ignore list**: The list of ids to be ignored as senders.

The demonstration will make all the details more palatable, here we can offer only dry descriptions. For the demo we use an ASCII terminal, pretending to be the extended user interface. For the rustic i/f on a final device, display likely will be scrolling on LCD, triggered by a user request. Event notifications will be signaled via LEDs, buzzers, etc.

**Let's** assume that the matchmaking data is *seawolfies* reflect users' preferences.

When switched on, the device sends periodic messages with its identifier, the owner's nickname and profile. The receiver and beacon are ON, unless switched off on the user's request.

- Received '*profile msg*' is checked against the '*Noombuzz list*'. If the sender is on the list, the message is handled as prescribed in the associated disposition. Otherwise:

- Received '*profile msg*' is checked against the *Ignore* list. If the sender is on the list, the message is dropped. Otherwise:

- Incoming *Profile* is checked against *Exclude* properties. If any matches, the message is dropped. Otherwise:

- Incoming *Profile* is checked against *Include* properties. If none matches, the message is dropped. Otherwise:

- Sender is checked against the '*tag list*'. This is a highly dynamic list with all current potential contacts, and with all persons the owner is interacting with. If the sender is not on the list, it is added there. Otherwise, some of the attributes are updated. (More is happening behind the scenes, as this list's maintenance is pivotal for a healthy application. De-bouncing of nodes flickering at the reception range is a good example of an important but not that obvious feature.)

There are commands to maintain all the lists, to display pertinent info, and to handle the intended flow of interactions. Let's go with a common chain of events:

**The user** displays the '*tag list'*, and picks an entry. The '**Y**' command accepts the sender, and replies with the user's *Description*. The '**N**' command moves the sender to the *Ignore* list.

After the '*profile msg'* is accepted, at least one side should be able to physically locate the other, and start conversation. Private or business data exchange may follow. The actual flow of events and exchanged information will vary, as all depends on spontaneous user decisions.

Alarms, although they may be used within any established community or for multi-hopping point to point communication, are primarily meant to organize conference participants into groups according

to the conference threads, workshops, social events, etc. -- they provide efficient (group-targeted) event notifications. Incoming alarms are checked against the *Ignore* and *Monitor* lists first. If the sender is not there, the usual profile matching occurs.

The *noombuzz* site is designated as the main (possibly: exclusive) interface for the *Seawolfie* provisioning. Rich web-based functionality, abundance of relevant data and flexible selection methods make it the ideal tool to prepare most of the configuration data loadable via a PC-USB interface. However, in this version we keep the '*event data*' and '*noombuzz data*' complementary, in hope to uncover still hidden but attractive functionality. We'll consolidate into a consistent approach in the next iteration. Let's take a look at the demonstration and more details:

## Demonstration

This section is not meant to be a comprehensive guide to the implemented functionality. Instead, we aim at giving the reader an orientation tour only.

Screen snapshots were taken from the demo running under VUEE, but the functionality and her implementing code are identical on the network of the EMSPCC11 nodes. The actual demonstration should include this presentation within VUEE, followed by a similar setup with physical nodes. We will deal with four nodes only, but larger configurations are likely to be studied before good requirements and competent hardware design decisions can be articulated.

### *Equipment*

Four Olsonet's general experimental board EMSPCC11 and their hand-assembled variants are used for the demonstration:

Essential components of the EMSPCC11:

- MSP430F1611 microcontroller: <http://focus.ti.com/docs/prod/folders/print/msp430f1611.html>
- CC1100 RF module: <http://focus.ti.com/docs/prod/folders/print/cc1100.html>
- On board data storage: <http://www.atmel.com/dyn/resources/prod_documents/doc3443.pdf>

Hand-assembled variants with two buttons, joystick, graphical LCD and buzzer are used to build skeleton PicOS drivers and verify the resource requirements. The pilot version of the application is likely to run on those boards, possibly after some further modifications.

### *Matchmaking*

The designers will be surprised how much innovative thoughts may be implemented for organizing this seemingly trivial functionality. True profiles will be much more elaborate, and the matching likely will employ fuzzy algorithms. For the demo, we deal with simple bitmaps and bitwise AND operations.

It is yet unclear how much of the selection and profiling can be performed on the *noombuzz* site, either prior or during the event the *Seawolfie* devices are deployed for. If this is the only required provisioning interface, most of the structures described below disappear, delegated to the *noombuzz* site. The '*Noombuzz list'* is prepared and downloaded to the device, and the matching is reduced to id comparisons. However, we keep the functionality independent from *noombuzz* for now, calling this part of the setup '*event data*' as opposed to '*noombuzz data*' from the '*Noombuzz list*'. One can easily nullify '*event data*' by clearing *Include* vector or setting *Exclude* to 0xFFFF. Similarly, empty (not downloaded) '*Noombuzz list*' leaves only '*event data*' operational.

Even if *Noombuzz* is the only provisioning interface, the functionality below should be contemplated, as that way or another id classification  via profile matching or other criteria must be performed somewhere.

Let's study 'event data' first:

Each *Profile* (same goes for *Include* and *Exclude* lists) is a 16-bit map of binary choices. We arbitrary divide them into four categories four choices each, as depicted below:

**Profile**

| | | | |
|---|---|---|---|
| **personal** | **professional** | **volatile** | **solid** |

**Personal**

| | | | |
|---|---|---|---|
| **visiting** | **local** | **female** | **male** |

**Professional**

| | | | |
|---|---|---|---|
| **thread 1** | **thread 2** | **workshop A** | **workshop B** |

**Volatile**

| | | | |
|---|---|---|---|
| **squash** | **bridge** | **pm trip** | **bar** |

**Solid**

| | | | |
|---|---|---|---|
| **cinema** | **outdoors** | **piano** | **music** |

Let's look at the data pretending to come from questionnaires, set for the experiment in the *simm.xml* configuration file (simm stands for *simple matchmaker*):

```
<node number="0">
   <location>5.0 5.0</location>
   <preinit tag="HID" type="lword">0xBACA000A</preinit>
   <preinit tag="NICK" type "string">Alice</preinit>
   <preinit tag="DESC" type "string">white blouse</preinit>
   <preinit tag="DBIZ" type "string">Ala Inc.</preinit>
   <preinit tag="DPRIV" type "string">613-111-2222</preinit>
   <preinit tag="PROFI" type="word">0xAF26</preinit>
   <preinit tag="PINC" type="word">0xFFFF</preinit>
   <preinit tag="PEXC" type="word">0</preinit>
</node>
```

This node is visible in the simulator as '***Node 0***', located at **(*5, 5*)** on the Cartesian plane. Its hardware

id is ***0xBACA000A***, of which two least significant bytes form the logical identifier **10** used for all communications. The user nicknamed ***Alice*** chose "***white blouse***" as her initial description, "***Ala Inc.***" as business data, and "***613-111-2222***" as her bold private cue. She describes herself as a *visiting female*, professionally interested in *all* choices,. She is interested in the *afternoon trip* organized for the conference participants, and if she wants to talk about her true interests, she'll talk about *piano* music or Himalayan trekking (*outdoors*). She want to talk to *all*, and excludes *nobody*.

Now one should be able to decode Bob's (11), Cat's (102) and Dog's (103) settings as well.

```
<node number="1">
   <location>15.0 5.0</location>
      <preinit tag="HID" type="lword">0xBACA000B</preinit>
      <preinit tag="NICK" type "string">Bob</preinit>
      <preinit tag="DESC" type "string">tall, red tie</preinit>
      <preinit tag="DBIZ" type "string">Bombardier</preinit>
      <preinit tag="DPRIV" type "string">room 666</preinit>
      <preinit tag="PROFI" type="word">0x55F0</preinit>
      <preinit tag="PINC" type="word">0xFFFF</preinit>
      <preinit tag="PEXC" type="word">0</preinit>
   </node>

   <node number="2" start="off">
      <location>5.0 15.0</location>
      <preinit tag="HID" type="lword">0xBACA0066</preinit>
      <preinit tag="NICK" type "string">Cat</preinit>
      <preinit tag="DESC" type "string">nice paws</preinit>
      <preinit tag="DBIZ" type "string">Mice Exterme</preinit>
      <preinit tag="DPRIV" type "string">on March break</preinit>
      <preinit tag="PROFI" type="word">0xE0FF</preinit>
      <preinit tag="PINC" type="word">0xFFFF</preinit>
      <preinit tag="PEXC" type="word">0x2000</preinit>
   </node>


   <node number="3" start="off">
      <location>15.0 8.0</location>
      <preinit tag="HID" type="lword">0xBACA0067</preinit>
      <preinit tag="NICK" type "string">Dog</preinit>
      <preinit tag="DESC" type "string">big jaws</preinit>
      <preinit tag="DBIZ" type "string">Hunt Extreme</preinit>
      <preinit tag="DPRIV" type "string">whistle high C</preinit>
      <preinit tag="PROFI" type="word">0xF014</preinit>
      <preinit tag="PINC" type="word">0xFFFF</preinit>
      <preinit tag="PEXC" type="word">0</preinit>
   </node>
```

All this is arbitrary and for illustration only. As already said, a good general frame for the matchmaking profiles won't be that obvious to forge.

### *Networking*

We employ four nodes. When switched on, they broadcast '*profi msg'* messages with frequencies spread between 2 and 4 seconds. All other messages are triggered by users. The messages are relatively long, as they contain strings of characters. However, most of them don't multi-hop.

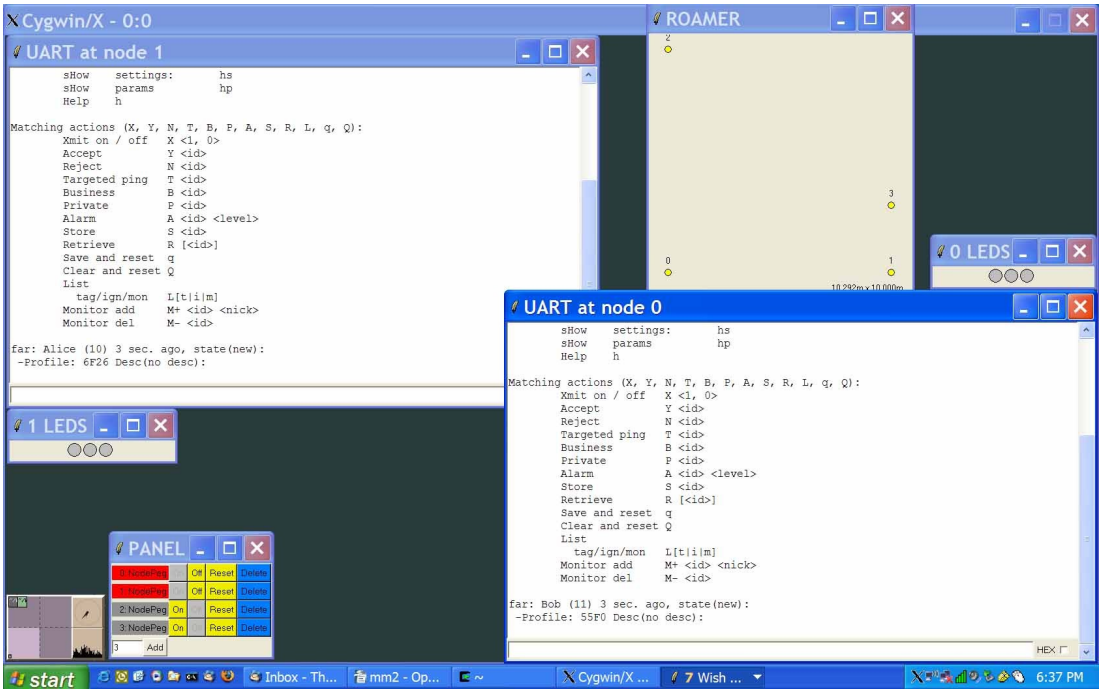
The typical setup will have a high density of nodes, so '*listen before transmit*' (LBT) is essential. Also, transmitters should operate on minimum power levels, to cut down the range and interference. For the demo, we set the data rate to 200K bps, to shorten the range to 5-6 meters.

We believe that multi-hopping functionality (here it is just *alarms*, but much more attractive features can be borrowed from our other blueprints) eventually will become popular, so for a large gathering it may be prudent to switch off forwarding on the user nodes, and set up a quasi-infrastructure with dedicated forwarders, appropriately placed to cover the venue with good and redundant data paths.

The infrastructure-backed setups are discussed in a separate document within the *Seawolf* project.

### *A simple VUEE model*

The following snapshot comes from the VUEE emulation of the network:




The four nodes are visible in the ROAMER window, the PANEL shows that only Alice's and Bob's are ON, and their extended UIs are in UART windows. LEDS are shown as well. We outline basic operations and their results using this setup.

At startup, the menu is displayed (no *noombuzz* here, for clarity):

```
Set / show matching (s, p):
        nickname:       sn <nickname 7>
        desc:           sd <description 15>
        priv:           sp <priv desc 15>
        biz:            sb <biz desc 15>
        alrm:           sa <alrm desc 15>
        profile:        pp <ABCD hex>
        exclude:        pe <ABCD hex>
        include:        pi <ABCD hex>

Help / bulk shows:
        sHow    settings:       hs
        sHow    params          hp
        Help    h

Matching actions (U, X, Y, N, T, B, P, A, S, R, L, q, Q):
        AUto on / off   U [1|0]
        Xmit on / off   X [1|0]
        Accept          Y <id>
        Reject          N <id>
        Targeted ping   T <id>
        Business        B <id>
        Private         P <id>
        Alarm           A <id> <level>
        Store           S <id>
        Retrieve        R [<id>]
        Save and reset q
        Clear and reset Q
        List
          tag/ign/mon   L[t|i|m]
        Monitor add     M+ <id> <nick>
        Monitor del     M- <id>
```

The first section sets the attributes we've already discussed. If a command appears without an argument, it is interpreted as a *show* request, e.g. *sn* is '*show nickname*'.

The next menu section is about bulk shows, divided into two parts. The actual output is the best introduction:

```
hs
Nick: Alice, Desc: white blouse
Biz: Ala Inc., Priv: 613-111-2222, Alrm:
Profile: 6F26, Exc: 0000, Inc: FFFF

hp
Stats for hostId - localHost (BACA000A - 10):
 Freq audit (4) events (4) PLev (1) Time (898)
 Mem free (788, 788) min (0, 0)
```

In the *hs* output we should recognize the data used for matchmaking, *hp* shows a few of the basic node's parameters we'll ignore for now.

Since Bob and Alice are in the range, they immediately see each other's pings. Alice sees Bob's data:

```
far: Bob (11) 3 sec. ago, state(new):
 -Profile: 55F0 Desc(no desc):
```

Bob (id 11) is 'far' (2.5 – 5 meters away), he appeared about 3 seconds ago with the 0x55F0 profile, and no description. Should Bob switch off the beacon (or the badge), or walk away, after a short while Alice would see:

```
far: Bob (11) 0 sec. ago, state(gone):
 -Profile: 55F0 Desc(no desc):
```

Two of the features not announced yet have become visible: rough distance estimation, and maintenance of the nodes in the immediate neighborhood reflected in the '*tag list*'.

As already mentioned, we set the nodes to have the RF range of about 5 meters. Rough but useful distance estimations based on the RSSI, are *far* (2.5 – 5), *near* (1.2 – 2.5), and *proxy* (0 – 1.2). How good this can be on the final hardware and how it may be extended to show directions, remain open issues.

More complex algorithms are applied for a reasonable maintenance of the neighborhoods. Several states are introduced (*new, reported, confirmed, matched, gone*), separated by corresponding '*fading*' states, all for a balance between two extremes: incessant reporting of all events, and a silence broken only on explicit user requests. The former is not practical in majority of realistic setups, the latter can't maintain the context achievable with automatic periodic messages. We believe that the functionality enabled via periodic pings makes a significant difference, therefore this more complex approach is justified.

We've already seen the *new* and *gone* states. If Alice replies with '*Y 11*', Bob will see:

```
far: Alice (10) 4750 sec. ago, state(reported):
 -Profile: 6F26 Desc(intro): white blouse
```

Now, if Bob sends back '*B 10*', Alice's terminal will show:

```
far: Bob (11) 0 sec. ago, state(matched):
 -Profile: 55F0 Desc(business): Bombardier
```

Alice would see '*confirmed*' state between her '*Y*' and Bob's '*B*'. Note that the transition to the *matched* state is considered important enough to reset the clock of the Bob's state. Of course, all these arbitrary choices can be adjusted for a given application.

The *autoack* feature helps with a meaningful demo with just one terminal. If set, automatic replies are sent for acceptable matchmaking messages. The reply is of the same type as the incoming message and mimics the user responding with the *Y, B*, or *P* command. In general, *autoacs* are useful for end to end reliable messaging.

If we switch Cat and Dog ON and start playing with the nodes in the ROAMER window, we'll see the results, imagine variety of scenarios at a real conference, and be able to check how the system deals with them. There is a wide gap between what is implemented for this demonstration and what will be needed for a real system. However, VUEE enables a comfortable environment for these activities.

The notion of '*targeted ping*' is introduced, to signal an explicit desire to communicate. If the user issues the 'T <id>' command, an extra beacon message is sent addressed to the target id. This explicit interest is signaled to the recipient with the "*intim*" (intimate) label, as below on the Dog's interface:

```
near: Cat (102) 664 sec. ago, *intim* state(matched):
 -Profile: E0FF Desc(private): on March break
```

As mentioned, alarms signal multi-hopping extensions, and in the demo their output looks like that:

```
Alrm from Cat(102) profile(E0FF) lev(7) hops(2) for 0:
happy hours at 5
```

Note the number of hops (2), and 0 as the receiver id (the latter means broadcast).

LEDs may be one of the very few ways to signal events on the actual wearable badge. For the demo, we have the following '*protocol*' implemented:

**Solid green** is on when the node is switched on. If there is at least one entry matched, the green led **blinks**. If the beacon is switched off, it sets the **blue** mode, blinking if there is a match on the '*tag list*'. **Red** replaces blue or green for about 4 seconds after an alarm is received and displayed. As so many other details, this is an arbitrary functionality.

*Save* and *Restore* move data to and from non-volatile memory, enabling post-collection processing, data archives, or event logs.

### *Noombuzz provisioning*

As said, this is not just an add-on. It may easily replace the *Include* and *Exclude* vectors, and the *Ignore* list. All other data may be downloaded from *Noombuzz*, obsoleting most of the functionality above, or rather moving data manipulation to the *Noombuzz* site. Let's look at the current implementation again, this time with the *Noombuzz* hooks in **bold**. The startup screen:

```
***Seawolfie 0.4***
Set / show matching (s, p):
       nickname:      sn <nickname 7>
       desc:          sd <description 15>
       priv:          sp <priv desc 15>
       biz:           sb <biz desc 15>
       alrm:          sa <alrm desc 15>
       profile:       pp <ABCD hex>
       exclude:       pe <ABCD hex>
       include:       pi <ABCD hex>

Help / bulk shows:
       sHow   settings:      hs
       sHow   params         hp
       sHow   event desc     he [ABCD hex [ABCD hex]]
       sHow   nbuZZ desc     hz [AB hex {AB hex]]
       Help   h

Matching actions (U, X, Y, N, T, B, P, A, S, R, E, L, q, Q):
       AUto on / off  U [1|0]
       Xmit on / off  X [1|0]
       Accept         Y <id>
       Reject         N <id>
       Targeted ping  T <id>
       Business       B <id>
       Private        P <id>
```

```
Alarm                 A <id> <level>
Store                 S <id>
Retrieve              R [<id>]
Erase                 E <id>
Save and reset        q
Clear and reset       Q
List
  nbuzz/tag/ign/mon   L[z|t|i|m]
nbuZZ add       Z+ <id> <what: 0|1> <why: AB hex> <dhook> <memo>
nbuZZ del       Z- <id>
Monitor add     M+ <id> <nick>
Monitor del     M- <id>
```

'*Show event desc*' is not really *noombuzz*-related, but added for the event symmetry. One may recall the arbitrary demo settings:

```
he
 event 15       visiting
 event 14       local
 event 13       female
 event 12       male
 event 11       thread 1
 event 10       thread 2
 event 9        workshop A
 event 8        workshop B
 event 7        squash
 event 6        bridge
 event 5        pm trip
 event 4        bar
 event 3        cinema
 event 2        outdoors
 event 1        piano
 event 0        music
```

Bob's profile is 0x55F0. What it means decoded? Alice can check easily:

```
he 55f0
 event 14       local
 event 12       male
 event 10       thread 2
 event 8        workshop B
 event 7        squash
 event 6        bridge
 event 5        pm trip
 event 4        bar
```

She can also see the match that brought Bob to her device. We recall that Alice's *Include* is a generous 0xFFFF, but say she is pickier with 0x5010. If she does:

```
he 55f0 5010
 event 14       local
 event 12       male
 event 4        bar
```

she'll see maybe not that impressive but true (selected from provisioned data) bonding with Bob.

Similarly with the *noombuzz* data:

```
hz
```

```
noombuzz 7    night life
noombuzz 6    career
noombuzz 5    blogs
noombuzz 4    looks great
noombuzz 3    last year
noombuzz 2    latin
noombuzz 1    combat
noombuzz 0    olsonet
```

The data pretend to be downloaded from the *noombuzz* site. Note the freedom in user's selection, or rather in the classification moved to the web site. We've made the vector 8-bit only, but we expect this width to be good enough for any comfortable setup. Note that an attempt at rigorous and general classification with the 16-bit vector for 'e*vent data*' is far from satisfactory. The superiority of the private selection on the *noombuzz* site over the classification uniform for all participants is clearly visible.

**Z+** fakes a single entry download from noombuzz. Let Alice do:

```
Z+ 11 0 8 0 $50
NBuzz add 11
```

She says: Bob (id **11**) should be ignored (**0**) because of the noombuzz category 3 (**8** = 2^3) '*last yea*r', with memo '**$50**' attached. If Bob was in the neighborhood, he'll be gone soon despite matching profile, as *noombuzz* takes priority over everything else:

```
far: Bob (11) 399 sec. ago, state(gone):
 -Profile: 55F0 Desc(no desc):
```

Let's see the noombuzz list:

```
Lz
NBuzz imports:
 - 11 dh(0) ....|... $50
```

Note the '|' and '.' showing encoded criteria. This convention is likely more readable than hex numbers for most users, and seems suitable for LCD.

If Alice didn't borrow from, Bob but lent him $50, her *noombuzz* entry would need this minute correction:

```
Z+ 11 1 8 0 $50
NBuzz upd 11
```

that would bring Bob back to Alice's *seawolfie*:

```
far: Bob (11) 6 sec. ago, state(new):
 -Profile: 55F0 Desc(no desc noombuzz):
```

Note the noombuzz reference, as easy to cross-check as before:

```
Lz
NBuzz imports:
 + 11 dh(0) ....|... $50
```

'+' and '-' are the only dispositions now. '*dh*' stands for '*data hook*' and represents associated data of possibly large volume, including downloaded or RF-transmitted pictures.

Voluminous data will be the main addition in version 0.5 of this document. If we're lucky, it'll be the *Seawolfie* 1.0, on well-defined hardware.

## Comments and thoughts

No particular order in these thoughts and reflections.

It may be that the requested functionality can be achieved with a simpler approach. However, we believe that the presented design has the advantage of being applicable to truly diverse application spaces. Also, we find most of the ad hoc networking applications deceivingly simple to sketch, and disappointingly complex in crucial but initially hidden details.

The actual '*leading customer'* or '*target application*' must be firmly established very early in the development cycle, as the device's physical characteristics and networking aspects will impact or even dictate fundamental decisions.

Special care must be taken to articulate *Seawolfie* interactions with existing devices and classical networking. It may be that the best approach is a software add-on to an existing device (Android, Blackberry). Consumer electronics space may be a good candidate to such extensions.

Unless there is a need for interoperability with a priori unknown systems, standards like ZigBee may be harmful. If interoperability seems important, the previous point should be considered.

As physical proximity is an important functional constraint, RF links should be rather short. For the demo, we set transmitters' power level and data rate such that good communication breaks at about 5 meters. Only slightly wider range is advisable for the true application, as short links also help with high density of the nodes.

RSSI is good only for rough distance estimations. Directional antennas or sensors may be used to both distance and directional approximations. However, this is pure and difficult R&D, with unpredictable outcome.

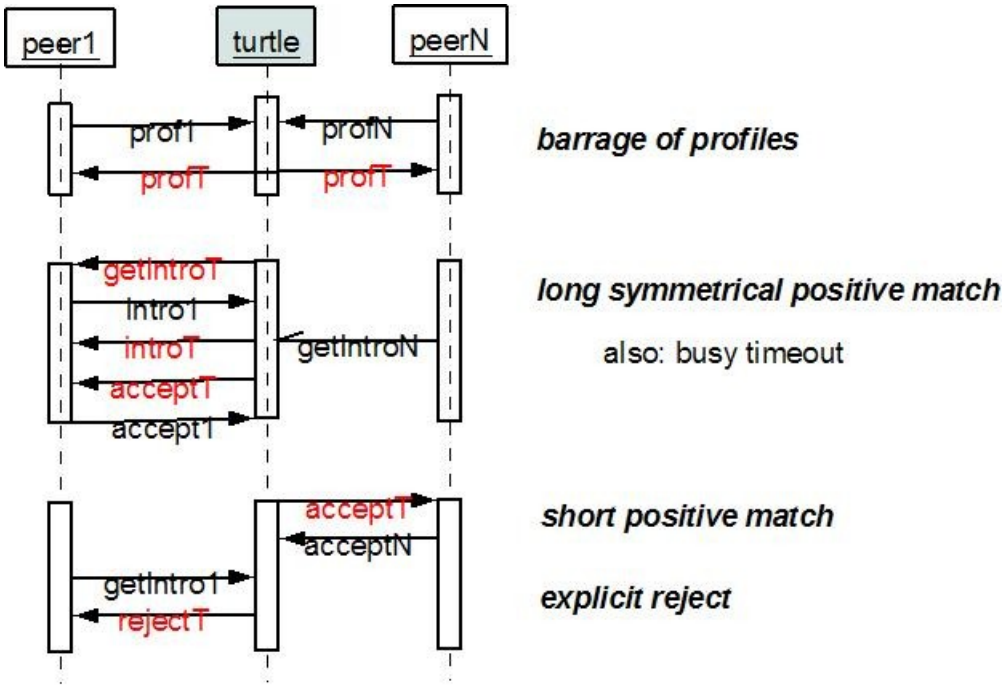
GUI is very important, and novel approaches may be needed. For large gatherings, works on a good OSS, distributed processing, or swarm intelligence may bring forward functionality that will eclipse the original goals.

The device should be designed in a way that signaling within the communities is very convenient through the embedded '*rustic interface*', but all verbose info, configuration parameters, and functional links to post-collection data processing should use the '*extended interface*'.

Non-volatile memory (*NVM*) plays an important functional role of the repository of data selected for further processing off the node. Diverse technologies can be employed, each with intricate constraints. Therefore, the *NVM* should be carefully researched and designed for the target application.

*Noombuzz* role (e.g. as the exclusive provisioning interface) should be decided as soon as possible, but not hastily; retractions may be expensive.

The RF traffic as well as UI 'load' may be significant in dense deployments. We may want to experiment with large number of nodes with matching profiles, to keep UI functional under the stress. To illustrate:




Note that the '*barrage of profiles'* is incessant. Network engineering for a given deployment should partition RF channels for picture exchange, site surveys may be needed to optimize (mostly: shorten) the RF range and set pertinent parameters (e.g. LBT). Of course, these considerations are valid for mass deployments, not for pilot runs.