The MeshLink library

Hamburg, July 28, 2014

Guus Sliepen
guus@meshlink.io

Februari 2014:

- Meeting in Hamburg.
- Discussion of using tinc in Everbase.

March 2014:

- Sliepen Datakonsult created.
- Meeting in Stockholm.
- Discussion how to go ahead, API.
- Decision to make a dual licensed project (GPL + commercial).
- New library (based on tinc) is called MeshLink.

April 2014:

- Saverio Proto starts work on the library.
- Tinc 1.1 codebase is turned into library.
- Ensured all copyright belongs to Sliepen Datakonsult.
- Start implementing the MeshLink API.

May, June 2014:

- More work on MeshLink.

July 2014:

- First "MeshLink conference" in Hamburg.

Goals of the MeshLink library:

- Provide a secure mesh network.
- Application can directly communicate with other nodes via API.
- Close-to-zero configuration required.
- MeshLink runs in a separate thread.
  - No need to use the same event API.
  - No need to worry timers, blocking calls, ...
- Interaction via thread-safe API.

Initialize MeshLink:

```
#include <meshlink.h>

meshlink_handle_t *mesh =
    meshlink_open("/path/to/cfg", "myname");
```

Now you can register callbacks. Nothing really happens yet until you start MeshLink:

```
meshlink_start(mesh);
```

Now you can run the rest of your application. Stop MeshLink with:

```
meshlink_stop(mesh);
```

Free all MeshLink resources with:

```
meshlink_close(mesh);
```

Invite other nodes into your mesh:

```
const char *url =
    meshlink_invite(mesh, "afriend");
```

Give the URL to the introducee. To use the URL to join an existing mesh:

```
meshlink_join(mesh, url);
```

Note:

- An invitation URL is good for one use.
- Make sure noone else gets the URL.

Or do a two-way exchange:

```
const char *mydata = meshlink_export(mesh);
meshlink_import(mesh, friendsdata);
```

Finding a single node:

```
meshlink_node_t *node
    = meshlink_get_node(mesh, "afriend");
```

Or get a list of all known nodes:

```
meshlink_node_t *nodes[100];
size_t n =
    meshlink_get_all_nodes(mesh, nodes, 100);
for(int i = 0; i < n && i < 100; i++)
    printf("%s\n", nodes[i]->name);
```

Or register a callback to get notified about nodes:

```
void status(meshlink_handle_t *mesh,
            meshlink_node_t *node,
            bool reachable)
{
    printf("%s %s reachable\n",
           node->name,
           reachable ? "is" : "is not");
}

meshlink_set_node_status_cb(mesh, status);
```

Sending data:

```
meshlink_node_t *dest =
    meshlink_get_node(mesh, "afriend");

meshlink_send(mesh, dest, "Hello!", 6);
```

Receiving data:

```
void receive(meshlink_handle_t *mesh,
             meshlink_node_t *src,
             const void *data, size_t len)
{
    printf("%s said: ", src->name);
    fwrite(stdout, data, 1, len);
    fputc('\n', stdout);
}

meshlink_set_receive_cb(mesh, receive);
```

Datagrams and streams

- Normally, data is sent as datagrams.
- Datagrams have UDP semantics:
  - Unreliable
  - Unordered
  - Packets are never split or merged

If one wants reliable, ordered communication, use channels:

- Channels have TCP semantics:
  - Reliable
  - Ordered
  - Boundary between packets is not preserved
- Multiple channels between two nodes possible

Sending data:

```
meshlink_node_t *dest =
    meshlink_lookup_node(mesh, "afriend");
meshlink_channel_t *chan =
    meshlink_channel_open(mesh, dest, 80, ...);
meshlink_channel_send(mesh, chan, "Hello!", 6);
meshlink_channel_close(mesh, chan);
```

Listening for incoming channels:

```
bool accept(meshlink_handle_t *mesh,
            meshlink_channel_t *chan,
            uint16_t port, ...)
{
    if (port != 80)
        return false;

    meshlink_set_channel_receive_cb(...);
    return true;
}

meshlink_set_channel_accept_cb(mesh, accept);
```

Receiving data:

```
void receive(meshlink_handle_t *mesh,
             meshlink_channel_t chan,
             void *data,
             size_t len)
{
    printf("%s said: ", chan->node->name);
    fwrite(stdout, data, 1, len);
    fputc('\n', stdout);
}

meshlink_set_channel_receive_cb(mesh,
    chan, receive);
```

C++ API:

```
#include <meshlink++.h>

auto mesh =
    meshlink::open("/path/to/cfg", "myname");

mesh->start();
auto node = mesh->get_node("afriend");
mesh->send(node, "Hello!", 6);
mesh->stop();

meshlink::close(mesh);
```

Just classes that wrap the C API for now.

Thread safety

- All MeshLink functions will be thread-safe.
- Callbacks are run in MeshLink thread.
- Application must provide callback thread-safety. Recommended way:
  - Put data in a queue.
  - Signal the application's event loop.

Current status:

- Port of tinc to library is done.
- API is documented using Doxygen.
- Simple demo code available.
- UDP-like communication is available.
- Channels are being worked on right now.

TODO:

- Integrate with a real application (Everbase)
- Finish channels API
- Even better documentation
- Fix bugs
- Release version 1.0

That's it!

- Website: `http://meshlink.io/`
- Git: `http://git.meshlink.io/`
  `git clone git://meshlink.io/meshlink`

Questions?