Overview

• The NEAT Project
• Disclaimer
• The API of the Library
• Examples
NEAT Project

• A New, Evolutive API and Transport-Layer Architecture for the Internet
• Partners:
  – Simula Research Laboratory
  – Celerway
  – EMC
  – Mozilla
  – Karlstad University
  – Münster University of Applied Sciences
  – University of Aberdeen
  – University of Oslo
  – Cisco
• https://www.neat-project.org
• https://github.com/NEAT-project/neat
Disclaimer

• This is work in progress
• The API can change anytime
• The code has not been tested substantially
• Comments welcome
• This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 644334 (NEAT). The views expressed are solely those of the author(s).
The Library

• BSD Licensed (3 clause)
• Implemented in C
• Portable (supports currently FreeBSD, Linux, Mac OS X, and NetBSD)
• Currently only supports the kernel versions of SCTP and TCP
• Callback based
• Uses libuv
• Uses flows
• A peer is addressed by names and ports
Infrastructure Functions

- `struct neat_ctx * neat_init_ctx(void)`
- `void neat_free_ctx(struct neat_ctx *)`
- `void neat_start_event_loop(struct neat_ctx *, neat_run_mode)`
- `void neat_stop_event_loop(struct neat_ctx *)`
Callbacks

• neat_error_code
  on_connected(struct neat_flow_operations *)
• neat_error_code
  on_error(struct neat_flow_operations *)
• neat_error_code
  on_readable(struct neat_flow_operations *)
• neat_error_code
  on_writable(struct neat_flow_operations *)
• neat_error_code
  on_all_written(struct neat_flow_operations *)
Callbacks (Cont.)

- `struct neat_flow_operations` contains pointers to the context, the flow, the status and all the callbacks.
- `neat_error_code`
  ```c
  neat_set_operations(struct neat_ctx *,
                     struct neat_flow *,
                     struct neat_flow_operations *)
  ```
Flow Management

- struct neat_flow *
  neat_new_flow(struct neat_ctx *)
- void
  neat_free_flow(struct neat_flow *)
Flow Management (Cont.)

- neat_error_code
  neat_open(struct neat_ctx *,
            struct neat_flow *,
            const char *,
            const char *)

- neat_error_code
  neat_accept(struct neat_ctx *,
              struct neat_flow *,
              const char *,
              const char *port)

- neat_error_code
  neat_shutdown(struct neat_ctx *,
                struct neat_flow *);
Flow Properties

- Require or ban
  - IPv4, IPv6
  - SCTP, TCP, UDP, UDPLite
  - Congestion control
  - Retransmissions

- Can request
  - Security
  - Message boundary preservation

```
neat_error_code
neat_get_property(struct neat_ctx *,
                   struct neat_flow *,
                   uint64_t *);
```

```
neat_error_code
neat_set_property(struct neat_ctx *,
                   struct neat_flow *,
                   uint64_t);
```
Flow Input/Output

- neat_error_code
  neat_open(
    struct neat_ctx *,
    struct neat_flow *,
    const char *,
    const char *)

- neat_error_code
  neat_read(
    struct neat_ctx *,
    struct neat_flow *,
    unsigned char *,
    uint32_t,
    uint32_t *);
int main(void) {
    ctx = neat_init_ctx();
    flow = neat_new_flow(ctx);
    prop = NEAT_PROPERTY_MESSAGE;
    neat_set_property(ctx, flow, prop);
    ops.on_connected = on_connected;
    ops.on_error = on_error;
    neat_set_operations(ctx, flow, &ops);
    neat_accept(ctx, flow, "*", "8080");
    neat_start_event_loop(ctx, NEAT_RUN_DEFAULT);
}
A Server – Handling Connections

```c
neat_error_code
on_connected(struct neat_flow_operations *opCB){
  opCB->on_all_written = on_all_written;
  opCB->on_readable = on_readable;
  return NEAT_OK;
}
```
A Server – Handling output

```c
neat_error_code
on_all_written(struct neat_flow_operations *opCB) {
    opCB->on_readable = on_readable;
    return NEAT_OK;
}

neat_error_code
on_writable(struct neat_flow_operations *opCB) {
    neat_write(opCB->ctx, opCB->flow, buffer, buffer_filled);
    opCB->on_writable = NULL;
    return NEAT_OK;
}
```
A Server – Handling Input

```c
neat_error_code
on_readable(struct neat_flow_operations *opCB) {
    neat_read(opCB->ctx, opCB->flow,
        buffer, buffer_size,
        &buffer_filled);
    if (buffer_filled > 0) {
        opCB->on_readable = NULL;
        opCB->on_writable = on_writable;
    } else {
        opCB->on_readable = NULL;
        opCB->on_writable = NULL;
        opCB->on_all_written = NULL;
        neat_free_flow(opCB->flow);
    }
    return NEAT_OK;
}
```
```c
int main(void) {
    ctx = neat_init_ctx();
    flow = neat_new_flow(ctx);
    prop = NEAT_PROPERTY_RETRANSMISSIONS_REQUIRED;
    neat_set_property(ctx, flow, prop);
    ops.on_connected = on_connected;
    ops.on_error = on_error;
    neat_set_operations(ctx, flow, &ops);
    neat_open(ctx, flow, name, port);
    neat_start_event_loop(ctx, NEAT_RUN_DEFAULT);
}
```
Happy Eyeballs

- Triggered by calling `neat_open()`
- Candidate selection can be improved by cached results
- Results are used and will be stored
- Several parameters of the connections can also be cached and used by future connection setups
- High-level requirements
Conclusion

• NEAT develops a library for using transport protocols in a service oriented way
• It is open source, so you can try it
• Comments, bug reports, fixes welcome