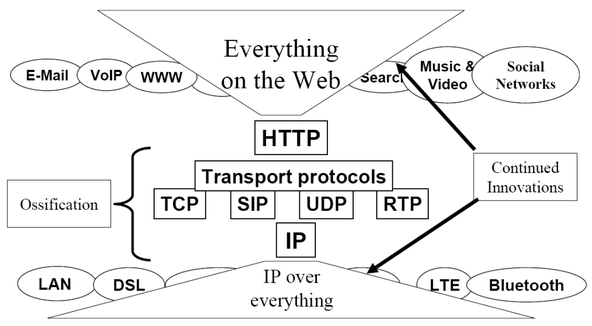
neat

NEAT – A New Evolutive API and Transport-Layer Architecture for the Internet

Karl-Johan Grinnemo[1], Tom Jones[2], Gorry Fairhurst[2], David Ros[3], Anna Brunstrom[1], and Per Hurtig[1]

[1]*Karlstad University, Karlstad, Sweden , Email: {karl-johan.grinnemo,anna.brunstrom,per.hurtig}@kau.se*
[2]*University of Aberdeen, Aberdeen, U.K. , Email: {tom,gorry}@erg.abdn.ac.uk*
[3]*Simula Research Laboratory, Oslo, Norway , Email: dros@simula.no*

Internet Transport is Ossified



Reasons:

- Middleboxes, e.g., NATs
- The Sockets API
- Obsolete IP options
- ...

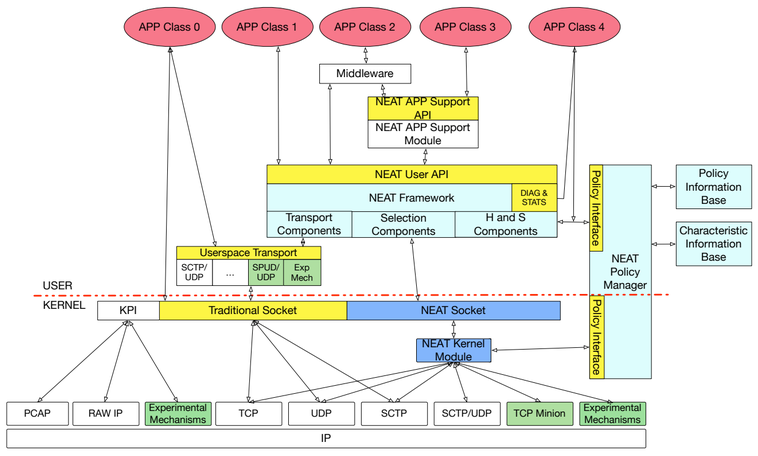
Point Solutions

- Virtual network overlays
  - Hide the underlying network from the transport
- Application-layer transport enhancements (e.g., middleware)
  - Fairly complex and limited scope
- Application-level transport protocols (e.g., QUIC)
  - Target a particular category of applications
- Sockets API extensions
  - Not deployable and evolvable

New Transport

- Deployable
  - Independent of particular software and technologies
- Evolvable
  - Permit parts (e.g., protocols) to be added as needed
  - Loose coupling of parts
- Flexible API
  - Higher abstraction level than the Sockets API
  - Offers a *transport service*

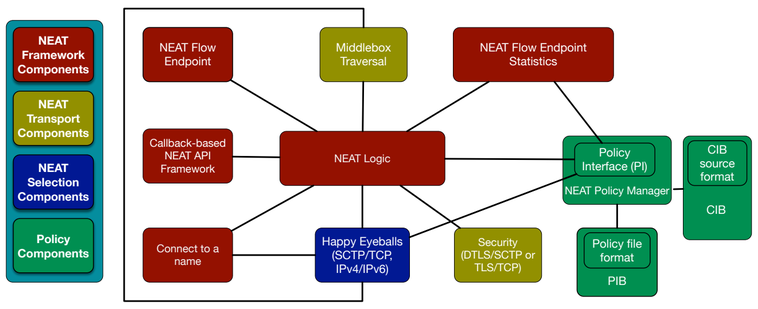
Overview NEAT Architecture



Two parts:

- High-Level Transport System
  - Often uses low-level transport system
- Low-Level Transport System
  - Essential for the creation, utilisation, and evolution of NEAT

Low-Level Transport System



Remarks:

- NEAT Flow Endpoint ≈ TCB
- NEAT Logic orchestrates the transport system
- NEAT Policy components administer application and network policies and comprise:
  - NEAT Policy Manager
  - Policy Information Base (PIB)
  - Characteristics Information Base (CIB)
- *Connect to a name* is the NEAT address resolver and provide the following features:
  - Asynchronous DNS lookup
  - Address monitoring
  - Multi-homing support
  - Private network marking
- *Happy Eyeballs* handles transport selection

NEAT API Framework

- Implements a callback-based API
- The base is an event loop
- Built on top of `libuv`

NEAT API Example

```
static struct neat_flow_operations ops;

static neat_error_code
on_error(struct neat_flow_operations *opCB)
{
    exit(EXIT_FAILURE);
}

static 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;
}

int
main(int argc, char *argv[])
{
    if ((ctx = neat_init_ctx()) == NULL) {
        debug_error("could not initialise context");
        result = EXIT_FAILURE;
        goto cleanup;
    }

    // new neat flow
    if ((flow = neat_new_flow(ctx)) == NULL) {
        debug_error("neat_new_flow");
        result = EXIT_FAILURE;
        goto cleanup;
    }

    // set callbacks
    ops.on_connected = on_connected;
    ops.on_error = on_error;

    if (neat_set_operations(ctx, flow, &ops)) {
        debug_error("neat_set_operations");
        result = EXIT_FAILURE;
        goto cleanup;
    }

    // wait for on_connected or on_error to be invoked
    if (neat_open(ctx, flow, argv[argc - 2],
        argv[argc - 1]) == NEAT_OK) {
        neat_start_event_loop(ctx, NEAT_RUN_DEFAULT);
    } else {
        debug_error("neat_open");
        result = EXIT_FAILURE;
        goto cleanup;
    }

    ...
}
```

Ongoing and Future Work

- Implement `libNEAT` library
- Evaluate performance of `libNEAT`
  - E.g., *Happy Eyeballs*
- Transport Protocol Enhancements
  - E.g., multipath scheduling
- Transport System Extensions
  - E.g., transport selection