neat

TAPS-related topics from the NEAT project

Naeem Khademi

TAPS WG - IETF 98

Chicago- USA

28 March 2017

Horizon 2020
European Union funding
for Research & Innovation

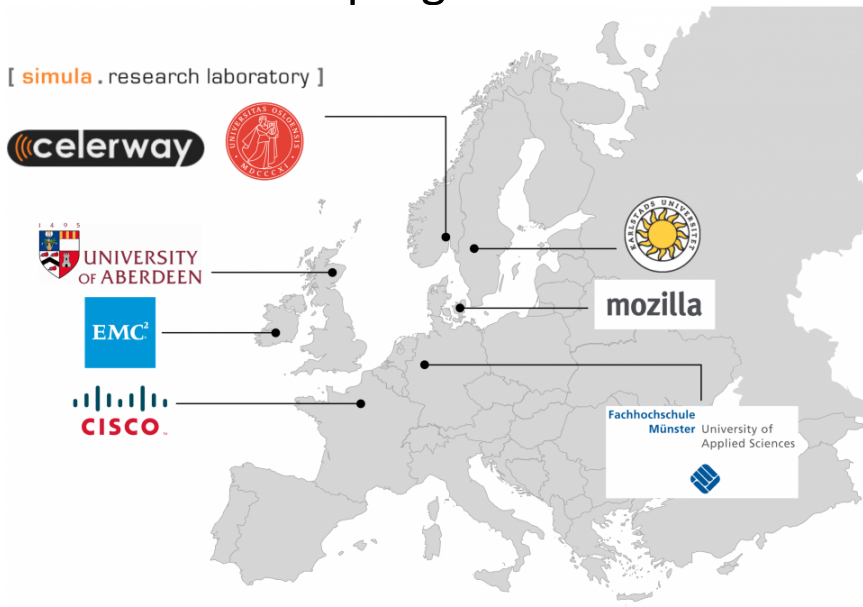
Introduction on NEAT

- NEAT project has been ongoing since March 2015

- NEAT library builds a TAPS-like prototype system
  - Protocol- and platform-independent
  - Open source, BSD Licensed (3 clause), implemented in C
  - Currently supports FreeBSD, Linux (Ubuntu), Mac OSX, and NetBSD
  - Event-based (using callbacks), libuv-based
  - Most core components are in place but still work-in-progress

- The NEAT API was first presented at IETF 95 (Buenos Aires)

- NEAT User API based on draft-ietf-taps-transports-usage and NEAT internal use-cases



neat

Key Features

- **API properties**
- NEAT Policy system
- Simpler/flexible coding with the NEAT User API
- Application feedback (Happy Apps)

Application properties in NEAT

- **NEAT gives users a chance to control as much as they want, yet allow automatization**

```
{
    "property_name": {
        value: "property_value",
       precedence: 1
    }
}
```

- It uses a key/value-based property system in JSON format
  - They can have different types and metadata attached to them, e.g. precedence
  - can set multiple/all properties with one API call

```
neat_set_property(ctx, flow, properties);
```

```
{
    "transport": [
        {
            "value": "SCTP",
            "precedence": 1
        },
        {
            "value": "TCP",
            "precedence": 1
        }
    ]
}
```

- Properties are given “precedence” -- e.g. 1=desired; 2=required
  1) Desired: try and fallback if unsuccessful
  2) Required: fail if unsuccessful

Key Features

- API properties
- **NEAT Policy system**
- Simpler/flexible coding with the NEAT User API
- Application feedback (Happy Apps)

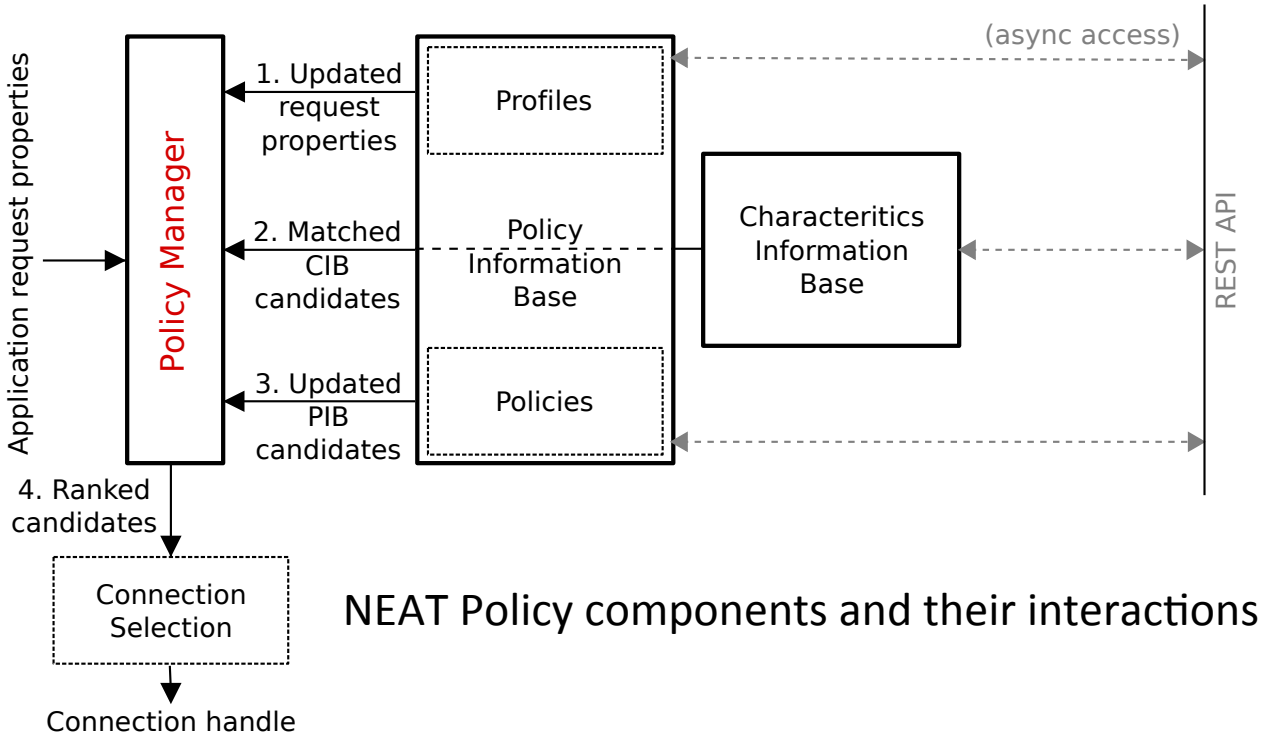
NEAT Policy System (#1)

NEAT's selection of protocols and parameterization is based on:

① Configured policies (PIB lookup)

② Tested capabilities (Happy Eyeballs)

③ Known and learned capabilities (CIB lookup)



NEAT Policy components and their interactions

NEAT Policy System (#2)

- **NEAT provides a flexible way for defining policies; also allows for creation of profiles depending on the networking scenario**

- **Policies:** based on NEAT properties with priorities among themselves, in JSON format; set by the user, system administrator or developer

- **Profiles:** are policies applied before CIB lookup; match (high-level) property in the request is *replaced* with the associated profile (low-level) properties

- Policies and profiles are stored in Policy Information Base (PIB)

```
{
  "name": "Low latency",
  "match": {
    "low_latency": {
      "precedence": 1,
      "value": true
    }
  },
  "properties": {
    "interface_latency":
{
      "precedence": 2,
      "value": [0,40]
    },
    "is_wired": {
      "precedence": 1,
      "value": true
    }
  }
}
```

An example of profile

Key Features

- API properties
- NEAT Policy system
- **Simpler/flexible coding with the NEAT User API**
- Application feedback (Happy Apps)

Simpler/flexible coding with the NEAT User API

- **Built in NEAT:** many common network programming tasks like address resolution, buffer management, encryption, connection establishment and handling

- Address resolution and connection establishment with a *single* function call

```
neat_open(ctx, flow, "bsd10.fh-muenster.de", 80, NULL, 0);
```

- **Example #1:** Establishing a listening socket (42 SLoC with NEAT API vs 59 SLoC with socket API)

- **Example #2:** we ported Nghttp2 (a HTTP/2 implementation) web server/ client and a few other smaller http/https-based clients to use NEAT
  - Interoperable with TCP
  - Can benefit from using SCTP

~20% reduction in code lines

Key Features

- API properties
- NEAT Policy system
- Simpler/flexible coding with the NEAT User API
- **Application feedback (Happy Apps)**

Application feedback (Happy Apps) (#1)

- **Happy Apps:** offers selection mechanisms when the underlying transport protocol does not provide the signals required -- e.g. QoS fallback with UDP(-Lite)

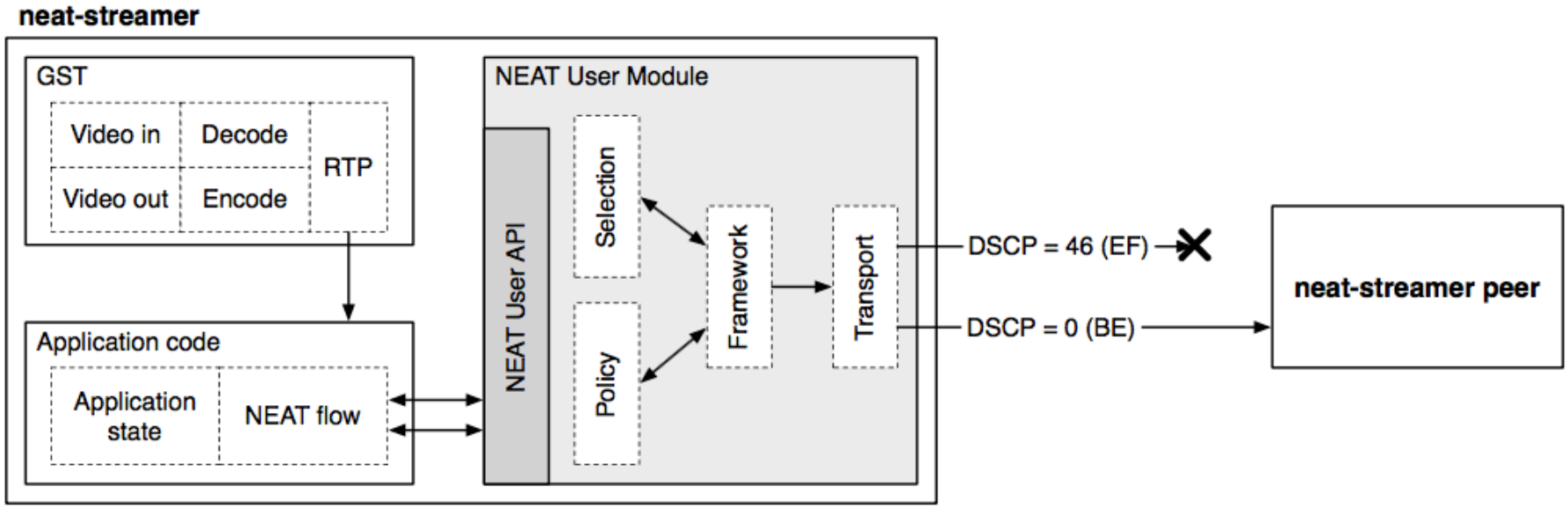
- **Network QoS:** often limited to controlled network environment due to lack of *high-level API*

- **Key challenge:** how to express service requirements, while still enabling policy to influence choice and providing flexibility when the network is unable to directly satisfy the requirements

- **With NEAT:** can use user requirements, policy, and dynamic info collected from other connections to drive an appropriate DSCP code-point

Application feedback (Happy Apps) (#2)

Example: we developed **neat-streamer** based on Gstreamer (pipeline-based media library for audio/video)



Major updates since IETF 95 (#1)

- **Multi-streaming:** transparent use of SCTP multi-streaming (compile-time option)

- **Flow-level priority:** API support for flow group (local) priorities to leverage:
  - Coupled-CC with TCP based on draft-welzl-tcp-ccc-00
  - Stream scheduling with SCTP *(WiP)*

- **"transport protocol" HE mechanism:** improvements in the code
  - Including investigation of transport HE's cost **(presented in TAPS, IETF 96)**
  - Uses priorities among "candidate transport solutions" with a fixed delay
  - NEAT transport-level HE is explained in draft-grinnemo-taps-he-02

Major updates since IETF 95 (#2)

- **Datagram support** for the API (UDP, UDP-Lite)
- **Support for SCTP, SCTP/UDP** (both in kernel and userland)
- **Server-side support** (listening on multiple protocols)
- **Multi-homing support** (with STCP)
- **Multipath support** (with MPTCP) *(unmerged)*
- **Security** (TLS/TCP; DTLS/UDP and ongoing work on DTLS/SCTP)
- Lots of improvements, debuggings and coding optimizations!

NEAT EU project: https://www.neat-project.org

Github Repository: https://github.com/NEAT-project/neat

API documentation and tutorial: http://neat.readthedocs.io/en/latest

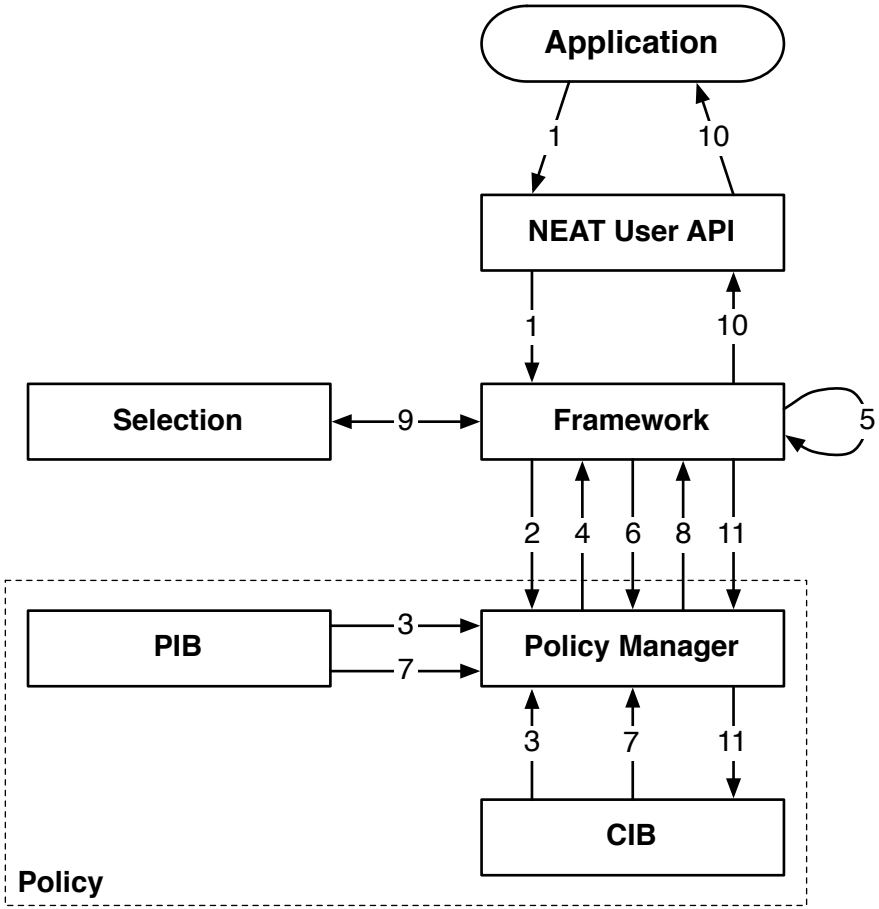
Also to appear in IEEE Communications Magazine, June 2017

N. Khademi, D. Ros, M. Welzl, Z. Bozakov, A. Brunstrom, G. Fairhurst, K.-J. Grinnemo, D. Hayes, P. Hurtig, T. Jones, S. Mangiante, M. Tüxen, and F. Weinrank. ***"NEAT: A Platform- and Protocol-Independent Internet Transport API"***. IEEE Communications Magazine, accepted for publication, March 2017.

Comments, feedback, patches, test results, suggestions on target apps are welcome!

Q&A

NEAT Workflow



1. Request to open flow & pass application requirements
2. Query PM about feasible transport candidates based on destination domain name
3. PM determines available transport candidates that fulfil policy (PIB) and cached information (CIB)
4. Return ranked list of feasible transport candidates as pre-filter for address resolution
5. Resolve addresses
6. Query PM about feasible transport candidates for resolved destination address
7. PM builds candidates, assigning priorities based on PIB/CIB matches
8. Return ranked list of feasible transport candidates for flow establishment
9. Do Happy Eyeballs with candidates, according to specified priorities
10. Return handle to selected transport solution
11. Cache results from Happy Eyeballs in the CIB

Source: N. Khademi, D. Ros, M. Welzl, Z. Bozakov, A. Brunstrom, G. Fairhurst, K.-J. Grinnemo, D. Hayes, P. Hurtig, T. Jones, S. Mangiante, M. Tüxen, and F. Weinrank. "*NEAT: A Platform- and Protocol-Independent Internet Transport API*". IEEE Communications Magazine, accepted for publication, March 2017.

A Simple Client using the NEAT API (#1)

```
static char *properties =
"{\"transport\": {\"value\": \"reliable\", \"precedence\": 2}}";

int main(void) {
    struct neat_ctx *ctx;
    struct neat_flow *flow;
    struct neat_flow_operations ops;

    ctx  = neat_init_ctx();
    flow = neat_new_flow(ctx);
    memset(&ops, 0, sizeof(ops));
    ops.on_connected = on_connected;
    neat_set_operations(ctx, flow, &ops);
    neat_set_property(ctx, flow, properties);
    neat_open(ctx, flow, "bsd10.fh-muenster.de", 5000, NULL, 0);
    neat_start_event_loop(ctx, NEAT_RUN_DEFAULT);
    neat_free_ctx(ctx);
    return EXIT_SUCCESS;
}
```

A Simple Client using the NEAT API (#2)

```
static neat_error_code on_connected(struct neat_flow_operations *ops) {
    ops->on_writable    = on_writable;
    ops->on_all_written = on_all_written;
    neat_set_operations(ops->ctx, ops->flow, ops);
    return NEAT_OK;
}

static neat_error_code on_writable(struct neat_flow_operations *ops) {
    neat_write(ops->ctx, ops->flow, "Hi!", 3, NULL, 0);
    return NEAT_OK;
}

static neat_error_code on_all_written(struct neat_flow_operations *ops) {
    ops->on_readable = on_readable;
    ops->on_writable = NULL;
    neat_set_operations(ops->ctx, ops->flow, ops);
    return NEAT_OK;
}
```

A Simple Client using the NEAT API (#3)

```
static neat_error_code on_readable(struct neat_flow_operations *ops) {
    uint32_t bytes_read = 0;
    char buffer[32];

    if (neat_read(ops->ctx, ops->flow, buffer, 31,
                  &bytes_read, NULL, 0) == NEAT_OK) {
        buffer[bytes_read] = 0;
        fprintf(stdout, "Read %u bytes:\n%s", bytes_read, buffer);
    }
    neat_close(ops->ctx, ops->flow);
    neat_stop_event_loop(ops->ctx);

    return NEAT_OK;
}
```