TSVWG
Internet-Draft
Intended status: Informational
Expires: May 01, 2018

G. Fairhurst
T. Jones
University of Aberdeen
A. Brunstrom
Karlstad University
D. Ros
Simula Research Laboratory
October 30, 2017

The NEAT Interface to Transport Services
draft-fairhurst-taps-neat-00

Abstract

The NEAT System provides an example of a system designed to implement the TAPS Transport Services. This document presents the transport services that the NEAT User API provides to an application or upper-layer protocol. It also describes primitives needed to interface to the NEAT Policy Manager and how policies can be adjusted to match the API behaviour to the properties required by an application or upper-layer protocol using the NEAT User API.

Table of Contents

1. Introduction

The NEAT (New, Evolutive API and Transport-Layer Architecture for the Internet) [NEAT] System provides a call-back driven API to the network transport layer. It presents a set of transport services [RFC8095] that the NEAT User API provides to an application or upper-layer protocol.

The NEAT System has been implemented in the NEAT User Module. The focus of the present document is on the NEAT User API providing transport services to applications. This utilises a lower interface provided by a Kernel Programming Interface (KPI), to access the traditional Socket API or a transport service implemented in userspace.

Applications that use the NEAT User API can provide information about the features desired from the transport layer and determine the properties of the offered transport service. It is this additional information that enables the NEAT System to move beyond the constraints of the traditional Socket API, since the stack then becomes aware of what the application/user actually desires or requires for each traffic flow. The additional information can be used to automatically identify which transport components (protocol and other transport mechanisms) could be used to realise the required transport service. This can drive the selection by the NEAT System of the best transport components to use and determine how these need to be configured [I-D.grinnemo-taps-he]. In making decisions, the NEAT System can utilise policy information provided at configuration,

previously discovered path characteristics and probing techniques. This can be provided by a policy manager acting below the NEAT User API.

The architecture of the NEAT System is presented in [D1.1]. Some important features of NEAT compared to the existing sockets API are:

- Event-driven call-back driven interface, enabling applications to be designed so that they respond to events signalling the reception of data blocks, ability to send data blocks, or the successful transmission of data blocks. This concrete API is described in [D2.3].

- High-level transport interface independent of the selected transport protocol, allowing applications to be written without depending on the features of specific transport protocols, and hence allowing the most suitable transport protocol to be matched to the application, based on the transport features an application requires [RFC8095].

- Support for unordered/unreliable and reliable transport services.

- Explicit support for multistreaming and multipath transport protocols and network architectures.

- A flexible policy framework, allowing applications to describes the properties they expect or require of the transport system and thus enabling the transport services to be configured to match the capabilities of the network that is being used.

- Ability to work with other network-layer protocols (e.g., network signalling) to realise the required transport service.

The NEAT Library is an open source implementation and is available for download [NEAT-GIT]. This also provides tutorials and examples of code utilising the API and descriptions of the way in the which callback mechanisms can be used to build applications that use this interface. Further documentation for the current NEAT System is available at the NEAT Project web page, [NEAT-DOC].

2. The NEAT Context

Applications interact with the network by sending, receiving and controlling NEAT Flows.

The first step in establishing a flow with the NEAT System is to call a primitive to create and configure a Context. In the remainder of this document, the label P: is used to identify a primitive that may be called for a NEAT Context, and the label E: to identify an event provided by the NEAT System. Each primitive/event is associated with a particular NEAT Context. Most primitives specify the Context and provide a handle to the NEAT Flow upon which they operate, and the primitives and events for manipulating data can only be used after a NEAT Flow has been created.

P: INIT_CTX()

The INIT_CTX primitive sets up the datastructures needed by the NEAT System.

After all network operations are completed it can free the context. It returns a pointer to the newly allocated and initialized NEAT context.

P: FREE_CTX()

The FREE_CTX primitive is called when an created context is no longer needed. It frees the memory associated with the datastructures used by the NEAT System.

3. NEAT User API Primitives and Events

An application using the NEAT System needs to take the following steps to use the network after establishing a context:

1. Initialisation: create a flow by calling P: INIT_FLOW; and then calling P: SET_PROPERTIES to express the application requirements. This is used by the NEAT policy manager. Finally, it needs to bind call-back functions to respond to the events generated by the NEAT System.

2. Establishment / Availability: Connect the NEAT Flow (either actively to a destination endpoint or passively to receive from the network).

3. Writing and reading data: Call primitives to write data or respond to events requesting it to read data.

4. Maintenance: Call maintenance primitives, as needed, to configure attributes of the flow (e.g., while writing reading data).

5. Termination: Close (or abort) the NEAT Flow.

3.1. NEAT Flow Initialisation

An application needs to create and initialise a flow object before it can be used.

P: INIT_FLOW()

The INIT_FLOW primitive creates the essential data structures required to use a NEAT Flow. The application also needs to then call a primitive associate functions with each of the events that it wishes to process.

P: SET_PROPERTIES(property_list)

property_list : A set of flow properties expressed in JSON.

Each NEAT Flow has a set of properties that are set at the flow initialisation time. The SET_PROPERTIES primitive sets properties for the NEAT Flow. Properties are related to Transport Features and Services. For instance: link-layer security, transport-layer security, certificate verification, certificate and key properties can be set at initialisation time are related to a Confidentiality Transport Feature. A flow can also have attributes that can be read by an application using maintenance primitives after a flow has been initialised.

3.2. NEAT Flow Establishment

P: OPEN(destname port [stream_count])

destname : a NEAT-conformant name (which can be a DNS name or a set of IP addresses) to which to connect.

port : port number (integer) or service name (string) to which to connect.

stream_count : the number of requested streams to open (integer). Note that, if this parameter is not used, the system may still use multi-streaming underneath, e.g., by automatically mapping NEAT Flows between the same hosts onto streams of an SCTP association. Using this parameter disables such automatic functionality.

Returns: success or failure. If success, it also returns a handle for a NEAT Flow.

The OPEN primitive opens a flow actively for transports that require a connection handshake (e.g., TCP, SCTP), and opens the flow passively for transports that do not (e.g., UDP, UDP-Lite). Calling P:OPEN alone may not actually have an effect "on the wire", i.e., a P: ACCEPT at the peer may not be triggered by it. Since it is possible that the remote endpoint only returns when data arrives, this may only happen after the local host has called P: WRITE. (This does not result in a problem, since P: ACCEPT does not block).

E: on_connected

The on_connected event indicates a successful connection setup. An

application that receives this event can then use other primitives with this flow.

P: OPEN_WITH_EARLY_DATA(destname port [stream_count] [flow_group] [stream] [pr_method pr_value] [unordered_flag] data datalen)

destname : defined in the same way as in P: OPEN.

port : defined in the same way as in P: OPEN.

stream_count : defined in the same way as in P: OPEN.

flow_group : defined in the same way as in P: OPEN.

stream : the number of the stream to be used. At the moment this function is called, a connection is still not initialised and the protocol may not be known. If the protocol chosen by the NEAT Selection components supports only one stream, this parameter will be ignored.

pr_method and pr_value : if these parameters are used, then partial reliability is enabled and pr_method must have an integer value from 1 to 2 to specify which method to implement partial reliability is requested. Value 1 means: pr_value specifies a time in milliseconds after which it is unnecessary to send this data block. Value 2 means: pr_value specifies a re- quested maximum number of attempts to retransmit the data block. If the selected NEAT transport does not support partial reliability these parameters will be ignored. (See P: WRITE for more information).

unordered_flag : The data block may be delivered out-of-order if this boolean flag is set. Default: false. If the protocol chosen by the NEAT Selection components does not support unordered delivery, this parameter will be ignored.

data : the data-block to be sent.

datalen : the amount (positive integer) of data supplied in the data-block.

Returns: success or failure. If success, it also returns a handle for a NEAT Flow and the amount of supplied data that was buffered.

The OPEN_WITH_EARLY_DATA primitive allows data to be sent at the time when a flow is opened. To accommodate TLS 1.3 [I-D.ietf-tls-tls13] early data and the TCP Fast Open option [RFC7413], application data need to be supplied at the time of opening a NEAT Flow. This primitive opens a flow and sends early data if the protocol supports it. If the protocol chosen does not support early application data. The data will be buffered then sent after connection establishment, similar to calling P: WRITE. For this reason, in addition to the parameters of P: OPEN, this primitive also needs the same parameters as P: WRITE. The supplied data can be delivered multiple times (replayed by the network); an application must take this into account when using this function. This is commonly known as idempotence.

3.3. NEAT Flow Availability

This section describes how an application prepares a flow to accept communication from another NEAT endpoint.

P: ACCEPT([name] port [stream_count])

name : local NEAT-conformant name (which can be a DNS name or a set of IP addresses) to constrain acceptance of incoming requests to local address(es). If this is missing, requests may arrive at any local address.

port : local port number (integer) or service name (string), to constrain acceptance to incoming requests at this port.

stream_count : the number of requested streams to open (integer). Default value: 1.

Returns: one or more destination IP addresses, information about which destination IP address is used by default, inbound stream count (= the outbound stream count that was requested on the other side), and outbound stream count (= maximum number of allowed outbound streams).

The ACCEPT primitive prepares a NEAT Flow to receive network data. UDP and UDP-Lite do not natively support a POSIX-style accept mechanism; in this case, NEAT emulates this functionality. P: ACCEPT can only return once data arrives, not necessarily after the peer has called P: OPEN (The callback-based implementation does not have this problem because P: ACCEPT does not block).

E: on_connected

The on_connected event indicates a NEAT peer endpoint has connected, and other primitives can then be used.

3.4. Writing and reading data

The primitives in this section refer to actions that may be performed an open NEAT Flow, i.e., a NEAT Flow that was either actively established or successfully made available for receiving data. It permits an application to send and receive data-blocks over the API.

E: on_writable

The on_writable event indicates there is buffer space available and the application may write new data using P:WRITE.

P: P: WRITE([stream] [pr_method pr_value] [unordered_flag] data datalen)

stream : the number of the stream to be used (positive integer). This can be omitted if the NEAT Flow contains only one stream.

pr_method and pr_value : if these parameters are used, then partial reliability is enabled and pr_method must have an integer value from 1 to 2 to specify which method to implement partial reliability is requested. Value 1 means: pr_value specifies a time in milliseconds after which it is unnecessary to send this data-block. Value 2 means: pr_value specifies a requested maximum number of attempts to retransmit the data-block. If the selected NEAT transport does not support partial reliability these parameters will be ignored

unordered_flag : The data block may be delivered out-of-order if this boolean flag is set. Default: false. If the protocol chosen by the NEAT Selection components does not support unordered delivery, this parameter will be ignored.

data : The data block to be sent.

datalen : the amount (positive integer) of data supplied in data.

The WRITE primitive provide a NEAT Flow with a data block for transmission to the remote NEAT peer endpoint (with reliability limited by the conditions specified via pr_method, pr_value and the transport protocol used). NEAT Flows can support message delineation as a property of the NEAT Flow that is set via the INIT_FLOW primitive (S. 2.2.1). If a NEAT Flow supports message delineation, the data block is a complete message.

E: on_all_written

The on_all_written event indicates that all data requested to be written using P:WRITE has been sent.

E: on_send_failure

The on_send_failure event may be returned instead of E:on_all_written when the NEAT System was temporarily unable to complete a P:WRITE call, and it not known that all data has been written.

E: on_readable

The on_readable event indicates there is data available for the application that may be read using P:READ.

P: READ( )

data : the received data block.

datalen : the amount of data received.

Returns: [unordered_flag] [stream_id] data datalen If a message arrives out of order, this is indicated by unordered_flag. If the underlying transport protocol supports streams, the stream_id parameter is set.

The READ primitive reads a data block from a NEAT Flow into a provided buffer. NEAT Flows can support message delineation as a property of the NEAT Flow that is set via the INIT_FLOW primitive. If a NEAT Flow supports message delineation, the data block is a complete message.

3.5. Flow Maintenance Primitives

The primitives and events below are out-of-band calls that can be issued at any time after a NEAT Flow has been opened and before it has been terminated.

P: CHANGE_TIMEOUT(toval)

toval : the timeout value in seconds.

The CHANGE_TIMEOUT primitive adjusts the time after which a NEAT Flow will terminate if the written data could not be delivered. If this is not called, NEAT will make an automatic default choice for the timeout.

P: SET_PRIMARY(dst_IP_address)

dst_IP_address : the destination IP address that should be used as the primary address.

The SET_PRIMARY primitive is to be used with NEAT Flows that have multiple destination IP addresses, with protocols that do not use load sharing. It should not have an effect otherwise. This will overrule this general per-flow setting. If this is not called, the NEAT System will make an automatic default choice for the destination IP address.

P: SET_LOW_WATERMARK(watermark)

watermark : upper limit of unsent data in the socket buffer, in bytes.

The SET_LOW_WATERMARK primitive allows the application to limit the amount of unsent data in the underlying socket buffer. If set, NEAT will only execute E: WRITABLE when the amount of unsent data falls below the watermark. This allows applications to reduce sender-side queuing delay.

P: SET_MIN_CHECKSUM_COVERAGE(length)

length : The number of bytes that must be covered by the checksum for a datagram to be delivered to the application.

The SET_MIN_CHECKSUM_COVERAGE primitive allows an application to set the minimum acceptable checksum coverage length. This primitive only has effect for a received UDP-Lite datagram. A receiver that receives a UDP-Lite datagram with a smaller coverage length will not hand over the data to the receiving application. This is ignored for other protocols, where all data are fully covered by the checksum.

P: SET_CHECKSUM_COVERAGE(length)

length : sets the number of bytes covered by the checksum on outgoing UDP-Lite datagrams. This is ignored for other protocols, where all data are fully covered by the checksum.

The SET_CHECKSUM_COVERAGE primitive allows an application to set the number of bytes covered by the checksum in a UDP-Lite datagram. This only has effect when the UDP-Lite protocol is selected.

P: SET_TTL(ttl)

ttl : the hop limit to be used for reception.

The SET_TTL primitive sets the minimum IPv4 TTL or IPv6 Hop Count on a datagram that is required for it to be passed to the application.

E: on_network_status_changed

The on_network_status_changed event informs the application that something has happened in the network; it is safe to ignore without harm by many applications. A status code indicates what has happened in accordance with a table that includes at least the following three values: 1) ICMP error message arrived; 2) Excessive retransmissions; 3) one or more destination IP addresses have become available/ unavailable.

P: GET_PROPERTY(property)

property : string with a property name.

Returns: value set to the property by the Policy Manager expressed

as JSON.

The GET_PROPERTY primitive allows an application to discover the value assigned to a property by the Policy Manager. Properties are expressed as part of policies and handled by the NEAT Policy Manager and can only be read by an application once a flow has been initialised.

These currently are:

- Transport parameters: Parameters used (e.g., congestion control mechanism, TCP sysctl parameters, . . .). For example, the choice of congestion control mechanism is likely to depend on the capacity_profile parameter of the INIT_FLOW primitive, if that parameter is used -\u002D but does not specify a concrete congestion control algorithm, which this read- able property returns. More generally, this property gives the application a more concrete view of the choices made by the NEAT System.

- Interface statistics: Interface MTU, addresses, connection type (link layer), etc.

- Path statistics: Experienced RTT, packet loss (rate), jitter, throughput, path MTU, etc

- UsedDSCP: The DSCP assigned to an active NEAT Flow. This may differ from the requested DSCP when the QoS has been mapped by the policy system

3.6. NEAT Flow Termination

This set of primitives and events are related to gracefully or forcefully closing a NEAT Flow, or being informed about this happening.

P: CLOSE( )

The CLOSE primitive terminates a NEAT Flow after satisfying all the requirements that were specified regarding the delivery of data that the application has already given to NEAT. If the peer still has data to send, it cannot then be received after this call. Data buffered by the NEAT System that has not yet been given to the network layer will be discarded.

E: on_close

The on_close event informs the application that a NEAT Flow was successfully closed. This can be received at any time for an active NEAT Flow.

P: ABORT( )

The ABORT primitive terminates a connection without delivering remaining data.

E: on_aborted

The on_aborted event informs the application that the other side has aborted the NEAT Flow. The event can be received at any time for an active NEAT Flow.

E: on_timeout

The on_timeout event informs the application that the NEAT Flow is aborted because the default timeout has been reached before data could be delivered. This timeout adjusted by the P: CHANGE_TIMEOUT NEAT Flow maintenance primitive. The event can be received at any time for an active NEAT Flow.

3.7. NEAT Error Events

Errors that occur within the NEAT System or that are notified by the network result in an on_error event:

E: on_error

This event notifies a hard or soft error to the upper layer using the NEAT System.

4. Security Considerations

This document is about the design and usage of a transport API. The transport protocols accessed via this API each have security considerations.

The API may be used to request the use of security protocols accessed via the transport API.

5. IANA Considerations

XX RFC ED - PLEASE REMOVE THIS SECTION XXX

This memo includes no request to IANA.

6. References

[D1.1] Fairhurst, G., Jones, T., Damjanovic, D., Eckert, K., Grinnemo, K., Hansen, A., Mangiante, S., McManus, P., Papastergiou, G., Ros, D., Vyncke, E., Welzl, M. and M. Tuexen, "The NEAT Architecture (D1.1)", 2015, <https://www.neat-project.org/wp-content/uploads/2016/02/D1.1.pdf>.

[D2.3] Khademi, N., Bozakov, Z., Brunstroem, A., Dale, O., Damjanovic, D., Evensen, KR., Fairhurst, G., Fischer, A., Grinnemo, K., Jones, T., Mangiante, S., Petlund, A., Ros, D., Ruengeler, I., Stenberg, D., Tuexen, M., Weinrank, F. and M. Welzl, "The Final Version of Core Transport System (D2.3)", 2017, <https://www.neat-project.org/wp-content/uploads/2017/10/D2.3.pdf>.

[I-D.grinnemo-taps-he] Grinnemo, K., Brunstrom, A., Hurtig, P., Khademi, N. and Z. Bozakov, "Happy Eyeballs for Transport Selection", Internet-Draft draft-grinnemo-taps-he-03, July 2017.

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", Internet-Draft draft-ietf-tls-tls13-21, July 2017.

[NEAT-DOC] Stenberg, D., Weinrank, F., Khademi, N., Dreibholz, T., Jones, T., Bozakov, Z. and O. Dale, "NEAT Programming API Documentation", , <http://neat.readthedocs.io/>.

[NEAT-GIT] "NEAT Source Code Repository", , <https://github.com/neat-project/neat>.

[NEAT] "The EU New, Evolutive API and Transport-Layer Architecture for the Internet (NEAT) Project", 2017, <https://www.neat-project.org/>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S. and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC8095] Fairhurst, G., Ed., Trammell, B.Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <https://www.rfc-editor.org/info/rfc8095>.

Appendix A. Revision Information

-00 This is an individual draft for the IETF community, for consideration by the IETF TAPS WG.

Authors' Addresses

Godred Fairhurst
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen, AB24 3UE
Scotland

Email: gorry@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk/

Tom Jones
University of Aberdeen
Department of Engineering
Fraser Noble Building
Aberdeen, AB24 3UE
Scotland

Email: tom@erg.abdn.ac.uk
URI: http://www.erg.abdn.ac.uk/

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad, 651 88
Sweden

Email: anna.brunstrom@kau.se

David Ros
Simula Research Laboratory
Martin Linges vei 25
1364 Fornebu
Oslo,
Norway

Email: dros@simula.no