RTI TCP Transport

Version 4.5e

Generated by Doxygen 1.5.5

Sun Oct 23 23:19:16 2011

Contents

Chapter 1

TCP Transport for RTI Data Distribution Service

Real-Time Innovations, Inc.

The RTI TCP Transport enables communication between RTI Data Distribution Service applications using TCP. The communication can be either over a LAN or over a WAN.

1.1 Available Documentation

This document contains:

- **Overview** (p. 1)
- **Configuring the TCP Transport** (p. 2)

For additional information, please see the following PDF documents:

- RTI TCP Transport API Reference.
- RTI Data Distribution Service User's Manual

1.2 Overview

Out of the box, RTI Data Distribution Service uses UDP and Shared Memory to communicate with other DDS applications. This configuration is appropriate

for systems running within a single LAN. However, using the UDP transport introduces some problems when DDS applications in different LANs need to communicate:

- UDP traffic is usually filtered out by the LAN firewalls for security reasons.
- Forwarded ports are usually TCP ports.
- Each LAN may run in its own private IP address space and use NAT (Network Address Translation) to communicate with other networks.

RTI TCP Transport enables participant discovery and data exchange using the TCP protocol either on a LAN, or over the public WAN. RTI TCP Transport enables RTI Data Distribution Service to address the challenges of using TCP as a low-level communication mechanism between peers and limits the number of exposed ports to one. When using the default UDP transport, an RTI Data Distribution Service application uses multiple UDP ports for communication, which may make it unsuitable for deployment across firewalled networks.

1.3 Configuring the TCP Transport

The TCP transport is distributed as a shared library located under <RTI Data Distribution Service installation directory>/lib/<architecture>. The library is called **nddstransporttcp**.

There are two ways in which this transport can be configured:

1. By setting up predefined property strings in the Property QoS Policy of the DomainParticipant (on UNIX, Solaris and Windows systems only).

   Refer to **Configure TCP Transport with Property QoS Policy** (p. 22) for the predefined property strings.

   With this first approach, RTI Data Distribution Service will dynamically load the TCP transport library at run time and then implicitly create and register the transport plugin.

2. By instantiating a new transport and registering it with the DomainParticipant (available in C/C++ API only).

   Refer to **NDDS_Transport_TCPv4_new** (p. 19) for information on how to create a new TCP transport instance.

   Refer to **NDDS_Transport_Support_register_transport** for documentation on how to register the transport instance with RTI Data Distribution Service.

   To use this second approach, you need access to the **TCP Transport Plugin API** (p. 11) at compile time. Your application will have to include the following header file:

```
#include "ndds/transport_tcp/transport_tcp_tcpv4.h"
```

(The above path assumes that RTI TCP Transport is installed in the same directory as RTI Data Distribution Service.)

Regardless of which way is used to configure the TCP transport, you will need to specify TCP as the transport in the initial peers. See **TCP Transport Initial Peers** (p. 21) for details.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

Module Documentation

4.1 Socket Utilities

Socket Utilities.

Modules

- ^ **Connection Endpoint Abstractions**
  Connection Endpoint and Factory Abstractions.

4.1.1 Detailed Description

Socket Utilities.

4.2 Connection Endpoint Abstractions

Connection Endpoint and Factory Abstractions.

4.3 TCP Transport Plugin API

TCP Transport Plugin interfaces and definitions.

Data Structures

- struct **NDDS_Transport_TCPv4_Property_t**

  Configurable TCP Transport-Plugin properties.

Defines

- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN** "tcpv4_lan"

  IPv4 TCP/IP Transport-Plugin class name for LAN case.

- #define **NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN** "tcpv4_wan"

  IPv4 TCP/IP Transport-Plugin class name for WAN case.

- #define **NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER** 7400

  The default value for the server bind port.

- #define **NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT** (9216)

  Default value of maximum message size.

- #define **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (-1)

  Used to specify that os default should be used for the socket buffer size.

- #define **NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT** (5)

  The max number of pending connections waiting for the control thread to accept them.

- #define **NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT** (64)

  Number of signficant bits of 16-byte address.

- #define **NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT** NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

  The default settings for the buffer pool used to provide memory for the received messages.

- #define **NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT** { 4L, 1000L, 10L}

  The default settings for the buffer used to provide memory for the bytes being written using asynchronous send.

- #define **NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT** NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

  The default settings for the buffer pool used to provide memory for the control messages read or written.

- #define **NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT** NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

  The default settings for the buffer used to provide memory for the control message factory.

- #define **NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT** NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

  The default settings for the buffer used to provide memory for the control message attribute factory.

- #define **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN**

  *Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 43) structure for LAN communication.*

- #define **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT** NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN

  *Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 43) structure.*

- #define **NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST** {{0,0,0,0, 0,0,0,0, 0xff,0xff, 0x1c, 0xe8, 127,0,0,1}}

  The definition of localhost for the TCP transport: 127.0.0.1:7400.

Typedefs

- typedef void(* **NDDS_Transport_TCPv4_OnConnectionEstablishedCallback**)(RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port)

  Prototype of the function that can be called when a new connection is successfully established with a remote peer.

- typedef void(* **NDDS_Transport_TCPv4_OnConnectionLostCallback**)(RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port, **NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t** reason)

  Prototype of the function that can be called when an existing connection is lost.

Enumerations

- enum **NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t** {

  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_UNKNOWN = 0,

  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_FINALIZED,

  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_BROKEN_PIPE,

  NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_DESTROYED }

  A set of possible reasons why a connection is lost.

Functions

- NDDS_Transport_Plugin * **NDDS_Transport_TCPv4_new** (const struct **NDDS_Transport_TCPv4_Property_t** *property_in)

  Creates an instance of the TCP transport plugin and initializes it from the specified TCP transport property structure.

- NDDS_Transport_Plugin * **NDDS_Transport_TCPv4_create** (**NDDS_Transport_Address_t** *default_network_address_out, const struct **DDS_PropertyQosPolicy** *property_in)

  Creates an instance of a TCP Transport Plugin, using PropertyQosPolicy.

ˆ RTIBool **NDDS_Transport_TCPv4_Plugin_stringToTransportAddress** (**NDDS_Transport_Address_t** *address_out, const char *address_in, RTIBool acceptHostName_in)

Converts a string into a transport address suitable to be used with the TCP transport.

4.3.1 Detailed Description

TCP Transport Plugin interfaces and definitions.

4.3.2 Define Documentation

4.3.2.1 #define NDDS_TRANSPORT_CLASSNAME_TCPV4_LAN "tcpv4_lan"

IPv4 TCP/IP Transport-Plugin class name for LAN case.

4.3.2.2 #define NDDS_TRANSPORT_CLASSNAME_TCPV4_WAN "tcpv4_wan"

IPv4 TCP/IP Transport-Plugin class name for WAN case.

4.3.2.3 #define NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER 7400

The default value for the server bind port.

4.3.2.4 #define NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT (9216)

Default value of maximum message size.

4.3.2.5 #define NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT (-1)

Used to specify that os default should be used for the socket buffer size.

4.3.2.6 #define NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT (5)

The max number of pending connections waiting for the control thread to accept them.

4.3.2.7 #define NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT (64)

Number of signficant bits of 16-byte address.

4.3.2.8 #define NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

The default settings for the buffer pool used to provide memory for the received messages.

4.3.2.9 #define NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT { 4L, 1000L, 10L}

The default settings for the buffer used to provide memory for the bytes being written using asynchronous send.

4.3.2.10 #define NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

The default settings for the buffer pool used to provide memory for the control messages read or written.

4.3.2.11 #define NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT NDDS_TRANSPORT_ALLOCATION_SETTINGS_DEFAULT

The default settings for the buffer used to provide memory for the control message factory.

4.3.2.12 #define NDDS_TRANSPORT_TCPV4_CONTROL_-MESSAGE_ATTRIBUTE_FACTORY_GROWTH_-POLICY_DEFAULT NDDS_TRANSPORT_-ALLOCATION_SETTINGS_DEFAULT

The default settings for the buffer used to provide memory for the control message attribute factory.

4.3.2.13 #define NDDS_TRANSPORT_TCPV4_PROPERTY_-DEFAULT_LAN

Value:

```
{ \
 { /* parent */ \
   NDDS_TRANSPORT_CLASSID_TCPV4_LAN, /* classid */ \
   NDDS_TRANSPORT_TCPV4_ADDRESS_BIT_COUNT, /* address_bit_count */ \
   NDDS_TRANSPORT_PROPERTY_BIT_BUFFER_ALWAYS_LOANED, /* properties_bitmap */ \
   128, /* gather_send_buffer_count_max */ \
   NDDS_TRANSPORT_TCPV4_MESSAGE_SIZE_MAX_DEFAULT, /* message_size_max */ \
   NULL, /* allow_interfaces_list */ \
   0,    /* allow_interfaces_list_length */ \
   NULL, /* deny_interfaces_list */ \
   0,    /* deny_interfaces_list_length;*/ \
   NULL, /* allow_multicast_interfaces_list */ \
   0,    /* allow_multicast_interfaces_list_length */ \
   NULL, /* deny_multicast_interfaces_list */ \
   0     /* deny_multicast_interfaces_list_length */ \
 }, \
 NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT, /* send_socket_buffer_size */ \
 NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT, /* recv_socket_buffer_size */ \
 1, /* ignore_loopback_interface */ \
 1, /* ignore_nonrunning_interfaces */ \
 0, 0, 0xffL,    /* transport priority mapping */ \
 NDDS_TRANSPORT_TCPV4_TRANSPORT_MODE_SYMMETRIC, /* transport_mode (deprecated) */ \
 NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT, /* server_socket_backlog */   \
 NDDS_TRANSPORT_ADDRESS_INVALID, /* public_address */ \
 NULL, /* bind_interface_address */ \
 NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER, /*server_bind_port */ \
 NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT, /* read_buffer_allocation */          \
 NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT, /* write_buffer_allocation */ \
 NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT, /* control_buffer_allocation */      \
 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT, /* control_message_allocation */ \
 NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT, /* control_attribute_allocat
 0, /* force_asynchronous_send */ \
 NULL,  /* on_connection_established */ \
 NULL,  /* on_connection_lost */ \
 -1L, /* max_packet_size */ \
 0, /* enable_keep_alive */ \
 -1L, /* keep_alive_time */ \
 -1L, /* keep_alive_internal */ \
 -1L, /* keep_alive_retry_count */ \
 0, /* disable_nagle */ \
```

```
 -1L, /* logging_verbosity_bitmap */ \
 1, /* use_windows_select */ \
 100, /* outstanding_connection_cookies */ \
 -1L, /* outstanding_connection_cookies_life_span */ \
 3,   /* send_max_wait_sec */ \
 RTITLS_OPENSSL_CONFIGURATION_DEFAULT /* tls */ \
}
```

Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 43) structure for LAN communication.

4.3.2.14 #define NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN

Use this to initialize a **NDDS_Transport_TCPv4_Property_t** (p. 43) structure.

4.3.2.15 #define NDDS_TRANSPORT_TCPV4_ADDRESS_LOCALHOST {{0,0,0,0, 0,0,0,0, 0xff,0xff, 0x1c, 0xe8, 127,0,0,1}}

The definition of localhost for the TCP transport: 127.0.0.1:7400.

4.3.3 Typedef Documentation

4.3.3.1 typedef void(* NDDS_Transport_TCPv4_OnConnectionEstablishedCallback)(RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port)

Prototype of the function that can be called when a new connection is successfully established with a remote peer.

Parameters:

remote_peer_addresss the address of the remote peer connected encoded as network-ordered IPv4 address.

remote_peer_port the port number of the remote connection encoded as network-ordered 16-bit value

4.3.3.2 typedef void(* NDDS_Transport_TCPv4_-OnConnectionLostCallback)(RTI_UINT32 remote_peer_addresss, RTI_UINT16 remote_peer_port, NDDS_Transport_TCPv4_OnConnectionLost_ReasonCode_t reason)

Prototype of the function that can be called when an existing connection is lost.

This method is always called after the on_connection_established is called.

It is also called if a sendresource or recvresource is destroyed (after closing the existing connections), and on shutdown

Parameters:

remote_peer_addresss the address of the remote peer connected encoded as network-ordered IPv4 address.

remote_peer_port the port number of the remote connection encoded as network-ordered 16-bit value

reason an enumeration that describes the reason of the disconnection

4.3.4 Enumeration Type Documentation

4.3.4.1 enum NDDS_Transport_TCPv4_OnConnectionLost_-ReasonCode_t

A set of possible reasons why a connection is lost.

See also:

NDDS_Transport_TCPv4_OnConnectionLostCallback (p. 18)

Enumerator:

NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_UNKNOWN
The reason why the connection was closed is not known

NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_FINALI
Server side: connection closed by remote client. Client side: the internal send resource was destroyed.

NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_CODE_BROKE
Describes a condition where the remote peer has ungracefully closed the connection (app was crashed or suddenly interrupted)

NDDS_TRANSPORT_TCPV4_ON_CONNECTION_LOST_REASON_DESTROYED
Describes a condition where the current node has decided to close the connection.

4.3.5 Function Documentation

4.3.5.1 NDDS_Transport_Plugin* NDDS_Transport_TCPv4_new (const struct NDDS_Transport_TCPv4_Property_t * *property_in*)

Creates an instance of the TCP transport plugin and initializes it from the specified TCP transport property structure.

This is the main initializer of the TCP transport plugin. To use the default settings, pass a pointer to a struct **NDDS_Transport_TCPv4_Property_t** (p. 43) initialized with the macro **NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT** (p. 17).

Parameters:

property_in <<**in**>> A pointer to a struct **NDDS_Transport_TCPv4_Property_t** (p. 43) containing the transport property. The plugin will make a copy of this structure before returning (caller can safely destroy the object).

Returns:

A pointer to a newly instantiated plugin or NULL if an error occurred during creation.

4.3.5.2 NDDS_Transport_Plugin* NDDS_Transport_TCPv4_create (NDDS_Transport_Address_t * *default_network_address_out*, const struct DDS_PropertyQosPolicy * *property_in*)

Creates an instance of a TCP Transport Plugin, using PropertyQosPolicy.

For all the properties that are not specified, the default value is assumed (as described in **Configure TCP Transport with Property QoS Policy** (p. 22)).

Parameters:

default_network_address_out <<**out**>> Unused parameter

property_in <<**in**>> The DDS_PropertyQoSPolicy object containing the transport properties.

Returns:

A pointer to a newly instantiated plugin or NULL if an error occurred during creation.

4.3.5.3 RTIBool NDDS_Transport_TCPv4_Plugin_stringToTransportAddress (NDDS_Transport_Address_t * *address_out*, const char * *address_in*, RTIBool *acceptHostName_in*)

Converts a string into a transport address suitable to be used with the TCP transport.

This method will convert a string in the form **host:port** into a **NDDS_Transport_Address_t** structure.

If the 'port' component is not specified, it will assign the default value as described by the constant **NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER** (p. 14).

Parameters:

address_out <<**out**>> Transport address to initialize

address_in <<**in**>> the NULL-terminated string containing the address to parse

acceptHostName_in <<**in**>> Set it to RTI_TRUE to allow this function to resolve the host component of the string address through a DNS lookup; set it to RTI_FALSE to allow only IPv4 addresses in the 'host' part of the address string.

Returns:

RTI_TRUE if success or RTI_FALSE if an error occurred during the conversion.

4.4 TCP Transport Initial Peers

This section describes how to specify the TCP transport initial peers. With the TCP transport, the addresses of the initial peers (NDDS_DISCOVERY_PEERS) that will be contacted during the discovery process have the following format:

- For WAN communication: tcpv4_wan://<IP address or hostname>:<port>
- For LAN communication: tcpv4_lan://<IP address or hostname>:<port>

For example: setenv NDDS_DISCOVERY_PEERS tcpv4_wan://10.10.1.165:7400,tcpv4_wan://10.10.1.111:7400,tcpv4_lan://192.168.1.1:7500

When the TCP transport is configured for LAN communication (with the **NDDS_Transport_Property_t::classid** property), the IP address is the LAN address of the peer and the port is the server port used by the transport (the **NDDS_Transport_TCPv4_Property_t::server_bind_port** (p. 50) property).

When the TCP transport is configured for WAN communication (with the **NDDS_Transport_Property_t::classid** property), the IP address is the WAN or public address of the peer and the port is the public port that is used to forward traffic to the server port in the TCP transport.

4.5 Configure TCP Transport with Property QoS Policy

Predefined properties that can be used to configure the TCP Transport plugin through the Property QoS Policy of the DomainParticipant.

.

Property Name	Description	Required?
dds.transport.load_plugins	Comma-separated strings indicating the prefix names of all plugins that will be loaded by RTI Data Distribution Service. Up to 8 plugins may be specified. For example, "dds.transport.TCPv4.tcp1". In the following examples, <*TCP_prefix*> indicates the string that is used as a prefix in the property names for all the settings that are related to this TCP transport plugin. <*TCP_prefix*> must begin with "dds.transport." (such as "dds.transport.TCPv4.tcp1").	**YES**
<*TCP_prefix*>.library	Must set to **"nddstransporttcp"**. The library **"libnddstransporttcp.so"** (Unix) or **"nddstransporttcp.dll"** (Windows) needs to be in the path during run time for use by RTI Data Distribution Service (in the **LD_LIBRARY_PATH** environment variable on UNIX-based systems, or in **PATH** for Windows systems).	**YES**
<*TCP_prefix*>.create_function	Must be set to **"NDDS_Transport_TCPv4_create"** .	**YES**
<*TCP_prefix*>.aliases	Used to register the transport plugin returned by **NDDS_Transport_TCPv4_create** (p. 19) (as specified by <***TCP_prefix***>**.create_function**) to the DomainParticipant. Refer to **aliases_in**	NO

4.6 TCP Transport Examples

TCP Transport Examples.

Modules

- **C Example of TCP Transport with PropertiesQosPolicy Configuration**

  TCP Transport example with PropertiesQosPolicy configuration.

- **C Example of TCP Transport with Transport Registration**

  TCP Transport example with explicit transport registration.

4.6.1 Detailed Description

TCP Transport Examples.

4.7 RTI TLS Support

OpenSSL configuration interfaces and definitions.

Data Structures

- struct **RTITLS_Verification**

  Set of TLS properties for certificate authorities (CAs) and verification.

- struct **RTITLS_Identity**

  Set of TLS properties for identity.

- struct **RTITLS_DHParamFile**

  Name of a Diffie-Hellman (DH) key file and the length of the contained key in bits.

- struct **RTITLS_Ciphers**

  Set of TLS properties for cipher configuration.

- struct **RTITLS_OpenSSL_Configuration**

  Full set of TLS properties.

Defines

- #define **RTITLS_VERIFY_DEFAULT**

  Use this to initialize a RTITLS_Verfication structure.

- #define **RTITLS_IDENTITY_DEFAULT**

  *Use this to initialize a **RTITLS_Identity*** (p. 61) *structure.*

- #define **RTITLS_CIPHER_LIST_DEFAULT** "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH"

  Cipher list string for default channel (encrypted).

- #define **RTITLS_CIPHER_LIST_ENCRYPT_HIGH** "AES:ALL:!aNULL:!eNULL:!LOW:!EXP:+RC4:@STRENGTH"

  Cipher list string for default channel (encrypted, no low-strength).

- #define **RTITLS_CIPHER_LIST_UNENCRYPTED** "aNULL"

  Cipher list string for authentication-only channel (no encryption).

- #define **RTITLS_CIPHER_DEFAULT**

  *Use this to initialize a **RTITLS_Ciphers** (p. 58) structure.*

- #define **RTITLS_OPENSSL_CONFIGURATION_DEFAULT**

  *Use this to initialize a **RTITLS_OpenSSL_Configuration** (p. 64) structure.*

Typedefs

- typedef int(* **RTITLS_Verify_Callback**)(int preverify_ok, X509_STORE_CTX *x509_ctx)

  Callback used to verify peer certificates.

Functions

- void **RTITLS_thread_exit** ()

  clean up OpenSSL resources for current thread (call before exit)

- int **RTITLS_default_verify_callback** (int ok, X509_STORE_CTX *store)

  Default verify callback: log errors when verification fails.

- int **RTITLS_verbose_verify_callback** (int ok, X509_STORE_CTX *store)

  Verbose verify callback: log information about successful verification as well as errors when verification fails.

4.7.1 Detailed Description

OpenSSL configuration interfaces and definitions.

4.7.2 Define Documentation

4.7.2.1 #define RTITLS_VERIFY_DEFAULT

Value:

```
{ \
    NULL, NULL, /* ca_file, ca_path */ \
    -1, /* verify_depth (no depth limit) */ \
    NULL, /* callback (use default verify callback) */ \
    NULL /* crl_file */ }
```

Use this to initialize a RTITLS_Verfication structure.

4.7.2.2 #define RTITLS_IDENTITY_DEFAULT

Value:

```
{ \
    NULL, /* certificate_chain */ \
    NULL, /* certificate_chain_file */ \
    NULL, /* private_key_password */ \
    NULL, /* private_key */ \
    NULL, /* private_key_file */ \
    NULL, /* rsa_private_key */ \
    NULL /* rsa_private_key_file */ }
```

Use this to initialize a **RTITLS_Identity** (p. 61) structure.

4.7.2.3 #define RTITLS_CIPHER_LIST_DEFAULT "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH"

Cipher list string for default channel (encrypted).

4.7.2.4 #define RTITLS_CIPHER_LIST_ENCRYPT_HIGH "AES:ALL:!aNULL:!eNULL:!LOW:!EXP:+RC4:@STRENGTH"

Cipher list string for default channel (encrypted, no low-strength).

4.7.2.5 #define RTITLS_CIPHER_LIST_UNENCRYPTED "aNULL"

Cipher list string for authentication-only channel (no encryption).

4.7.2.6 #define RTITLS_CIPHER_DEFAULT

Value:

```
{ \
    NULL, /* cipher_list (default cipher list) */ \
```

```
    0, NULL, /* dh_param_files_length, dh_param_files (no DH params) */ \
    NULL, /* engine_id (no engine) */ \
    0, NULL, NULL, /* engine_pre_cmd_length, engine_pre_cmd_names, engine_pre_cmd_parameters */ \
    0, NULL, NULL /* engine_post_cmd_length, engine_post_cmd_names, engeine_post_cmd_parameters */ }
```

Use this to initialize a **RTITLS_Ciphers** (p. 58) structure.

4.7.2.7 #define RTITLS_OPENSSL_CONFIGURATION_DEFAULT

Value:

```
{ \
    RTITLS_VERIFY_DEFAULT, /* verify */ \
    RTITLS_IDENTITY_DEFAULT, /* identity */ \
    RTITLS_CIPHER_DEFAULT, /* cipher */ \
    RTITLS_RENEGOTIATE_DEFAULT /* renegotiate */ }
```

Use this to initialize a **RTITLS_OpenSSL_Configuration** (p. 64) structure.

4.7.3 Typedef Documentation

4.7.3.1 typedef int(* RTITLS_Verify_Callback)(int preverify_ok, X509_STORE_CTX *x509_ctx)

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.7.4 Function Documentation

4.7.4.1 void RTITLS_thread_exit ()

clean up OpenSSL resources for current thread (call before exit)

4.7.4.2 int RTITLS_default_verify_callback (int *ok*, X509_STORE_CTX * *store*)

Default verify callback: log errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.7.4.3 int RTITLS_verbose_verify_callback (int *ok*, X509_STORE_CTX * *store*)

Verbose verify callback: log information about successful verification as well as errors when verification fails.

See the OpenSSL manual page for SSL_CTX_set_verify for more information.

4.8 C Example of TCP Transport with PropertiesQosPolicy Configuration

TCP Transport example with PropertiesQosPolicy configuration.

Modules

- **HelloWorld.idl**
- **HelloCommon.h**
- **HelloWorld.c**
- **HelloWorld_publisher.c**
- **HelloWorld_subscriber.c**
- **HelloWorldPlugin.c**
- **HelloWorldSupport.c**

4.8.1 Detailed Description

TCP Transport example with PropertiesQosPolicy configuration.

See README.txt for details.

[$(NDDSHOME)/example/C/helloWorldTCP/README.txt]

4.9 HelloWorld.idl

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.idl]

4.10 HelloCommon.h

[$(NDDSHOME)/example/C/helloWorldTCP/HelloCommon.h]

4.11 HelloWorld.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.h]

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld.c]

4.12 HelloWorld_publisher.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld_publisher.c]

4.13 HelloWorld_subscriber.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorld_subscriber.c]

4.14 HelloWorldPlugin.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldPlugin.h]

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldPlugin.c]

4.15 HelloWorldSupport.c

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldSupport.h]

[$(NDDSHOME)/example/C/helloWorldTCP/HelloWorldSupport.c]

4.16 C Example of TCP Transport with Transport Registration

TCP Transport example with explicit transport registration.

Modules

- **Hello.c**
- **HelloPublisher.c**
- **HelloSubscriber.c**

4.16.1 Detailed Description

TCP Transport example with explicit transport registration.

See READ_ME.txt for details.

[$(NDDSHOME)/example/C/Hello_builtin_tcp/READ_ME.txt]

4.17 Hello.c

[$(NDDSHOME)/example/C/Hello_builtin_tcp/src/Hello.h]

[$(NDDSHOME)/example/C/Hello_builtin_tcp/Hello.c]

4.18 HelloPublisher.c

[$(NDDSHOME)/example/C/Hello_builtin_tcp/HelloPublisher.h]

[$(NDDSHOME)/example/C/Hello_builtin_tcp/HelloPublisher.c]

4.19 HelloSubscriber.c

[$(NDDSHOME)/example/C/Hello_builtin_tcp/HelloSubscriber.h]

[$(NDDSHOME)/example/C/Hello_builtin_tcp/HelloSubscriber.c]

Chapter 5

Data Structure Documentation

5.1 NDDS_Transport_TCPv4_Property_t Struct Reference

Configurable TCP Transport-Plugin properties.

Data Fields

- struct **NDDS_Transport_Property_t parent**

  Generic properties of all transport plugins.

- RTI_INT32 **send_socket_buffer_size**

  Size in bytes of the send buffer of a socket used for sending.

- RTI_INT32 **recv_socket_buffer_size**

  Size in bytes of the receive buffer of a socket used for receiving.

- RTI_INT32 **ignore_loopback_interface**

  Prevents the transport plugin from using the IP loopback interface.

- RTI_INT32 **ignore_nonrunning_interfaces**

  Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

- RTI_INT32 **transport_priority_mask**

Mask for the transport priority field.

- RTI_INT32 **transport_priority_mapping_low**

  Sets the low value of the output range to IPv4 TOS.

- RTI_INT32 **transport_priority_mapping_high**

  Sets the high value of the output range to IPv4 TOS.

- RTI_INT32 **server_socket_backlog**

  Determines what is the maximum length of the queue of pending connections.

- **NDDS_Transport_Address_t public_address**

  Public locator (IP address and port) of the transport instantiation.

- char * **bind_interface_address**

  Interface IP address for the transport sockets.

- RTI_INT32 **server_bind_port**

  Private IP port (inside the LAN) used by the transport to accept TCP connections.

- struct **TransportAllocationSettings_t read_buffer_allocation**

  Allocation settings applied to read buffers.

- struct **TransportAllocationSettings_t write_buffer_allocation**

  Allocation settings applied to buffers used for asynchronous (nonblocking) write.

- struct **TransportAllocationSettings_t control_buffer_allocation**

  Allocation settings applied to buffers used to serialize and send control messages.

- struct **TransportAllocationSettings_t control_message_allocation**

  Allocation settings applied to control messages.

- struct **TransportAllocationSettings_t control_attribute_allocation**

  Allocation settings applied to control messages attributes.

- RTI_INT32 **force_asynchronous_send**

  Controls whether the plugin will send data synchronously or asynchronously.

- **NDDS_Transport_TCPv4_OnConnectionEstablishedCallback on_connection_established**
- **NDDS_Transport_TCPv4_OnConnectionLostCallback on_connection_lost**
- RTI_INT32 **max_packet_size**

  The maximum size of a TCP segment.

- RTI_INT32 **enable_keep_alive**

  Configures the sending of KEEP_ALIVE messages in TCP.

- RTI_INT32 **keep_alive_time**

  Specifies the interval of inactivity in seconds that causes TCP to generate a KEEP_ALIVE message.

- RTI_INT32 **keep_alive_interval**

  Specifies the interval in seconds between KEEP_ALIVE retries.

- RTI_INT32 **keep_alive_retry_count**

  The maximum number of KEEP_ALIVE retries before dropping the connection.

- RTI_INT32 **disable_nagle**

  Disables the TCP nagle algorithm.

- RTI_INT32 **logging_verbosity_bitmap**

  Bitmap that specifies the verbosity of log messages from the transport.

- RTI_INT32 **outstanding_connection_cookies**

  Maximum number of outstanding connection cookies allowed by the transport when acting as server.

- RTI_INT32 **outstanding_connection_cookies_life_span**

  Maximum lifespan (in seconds) of the cookies associated with pending connections.

- RTI_INT32 **send_max_wait_sec**

  Maximum number of seconds a low-level TCP sendto() function is allowed to block.

- struct **RTITLS_OpenSSL_Configuration tls**

  OpenSSL TLS parameters.

5.1.1 Detailed Description

Configurable TCP Transport-Plugin properties.

For communications within a LAN you should initialize the object as follows:

```
prop.parent.classid = NDDS_TRANSPORT_CLASSID_TCPV4_LAN;
```

If you want to use the transport in WAN configuration, you must also define the public_address and sever_bind_port:

```
prop.parent.classid = NDDS_TRANSPORT_CLASSID_TCPV4_WAN;
NDDS_Transport_TCPv4_Plugin_stringToTransportAddress(
        &prop.public_address,
        "142.123.123.111:7890",
        0);
prop.server_bind_port = 7400;
```

Remember that the public_address is the public address that your network gateway is exposing to a remote peer. It is not the local network address where the application is running.

5.1.2 Field Documentation

5.1.2.1 struct NDDS_Transport_Property_t NDDS_Transport_TCPv4_Property_t::parent [read]

Generic properties of all transport plugins.

[default] Refer to the property default **NDDS_Transport_TCPv4_Plugin::NDDS_TRANSPORT_TCPV4_PROPERTY_DEFAULT_LAN** (p. 16)

5.1.2.2 RTI_INT32 NDDS_Transport_TCPv4_Property_t::send_socket_buffer_size

Size in bytes of the send buffer of a socket used for sending.

On most operating systems, setsockopt() will be called to set the SENDBUF to the value of this parameter.

This value must be greater than or equal to **NDDS_Transport_Property_t::message_size_max** or **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 14).

The maximum value is operating system-dependent.

[default] **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 14)

5.1.2.3 RTI_INT32 NDDS_Transport_TCPv4_Property_t::recv_socket_buffer_size

Size in bytes of the receive buffer of a socket used for receiving.

On most operating systems, setsockopt() will be called to set the RECVBUF to the value of this parameter.

This value must be greater than or equal to parent.message_size_max or **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 14).

The maximum value is operating-system dependent.

[default] **NDDS_TRANSPORT_TCPV4_SOCKET_BUFFER_SIZE_OS_DEFAULT** (p. 14)

5.1.2.4 RTI_INT32 NDDS_Transport_TCPv4_Property_t::ignore_loopback_interface

Prevents the transport plugin from using the IP loopback interface.

This property is ignored when parent.classid is equal to **NDDS_TRANSPORT_CLASSID_TCPV4_WAN**.

Two values are allowed:

- 0: Enable local traffic via this plugin. The plugin will only use and report the IP loopback interface only if there are no other network interfaces (NICs) up on the system.
- 1: Disable local traffic via this plugin. This means "do not use the IP loopback interface, even if no NICs are discovered." This setting is useful when you want applications running on the same node to use a more efficient plugin like shared memory instead of the IP loopback.

[default] 1

5.1.2.5 RTI_INT32 NDDS_Transport_TCPv4_Property_t::ignore_nonrunning_interfaces

Prevents the transport plugin from using a network interface that is not reported as RUNNING by the operating system.

The transport checks the flags reported by the operating system for each network interface upon initialization. An interface which is not reported as UP will not be used. This property allows the same check to be extended to the IFF_RUNNING flag implemented by some operating systems. The RUNNING flag

is defined to mean that "all resources are allocated" and may be off if no link is detected (e.g., the network cable is unplugged).

Two values are allowed:

- 0: Do not check the RUNNING flag when enumerating interfaces, just make sure the interface is UP.
- 1: Check the flag when enumerating interfaces, and ignore those that are not reported as RUNNING. This can be used on some operating systems to cause the transport to ignore interfaces that are enabled but not connected to the network.

[default] 1

5.1.2.6 RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mask

Mask for the transport priority field.

This is used in conjunction with **transport_priority_mapping_low** (p. 48) and **transport_priority_mapping_high** (p. 49) to define the mapping from DDS transport priority (see **TRANSPORT_PRIORITY**) to the IPv4 TOS field. Defines a contiguous region of bits in the 32-bit transport priority value that is used to generate values for the IPv4 TOS field on an outgoing socket.

For example, the value 0x0000ff00 causes bits 9-16 (8 bits) to be used in the mapping. The value will be scaled from the mask range (0x0000 - 0xff00 in this case) to the range specified by low and high.

If the mask is set to zero, then the transport will not set IPv4 TOS for send sockets.

[default] 0

5.1.2.7 RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mapping_low

Sets the low value of the output range to IPv4 TOS.

This is used in conjunction with **transport_priority_mask** (p. 48) and **transport_priority_mapping_high** (p. 49) to define the mapping from DDS transport priority to the IPv4 TOS field. Defines the low value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

See also:

transport_priority_mask (p. 48)

[**default**] 0

5.1.2.8 RTI_INT32 NDDS_Transport_TCPv4_Property_t::transport_priority_mapping_high

Sets the high value of the output range to IPv4 TOS.

This is used in conjunction with **transport_priority_mask** (p. 48) and **transport_priority_mapping_low** (p. 48) to define the mapping from DDS transport priority to the IPv4 TOS field. Defines the high value of the output range for scaling.

Note that IPv4 TOS is generally an 8-bit value.

See also:

transport_priority_mask (p. 48)

[**default**] 0xFF

5.1.2.9 RTI_INT32 NDDS_Transport_TCPv4_Property_t::server_socket_backlog

Determines what is the maximum length of the queue of pending connections.

[**default**] **NDDS_TRANSPORT_TCPV4_SERVER_SOCKET_BACKLOG_DEFAULT** (p. 15)

5.1.2.10 NDDS_Transport_Address_t NDDS_Transport_TCPv4_Property_t::public_address

Public locator (IP address and port) of the transport instantiation.

Use the function **NDDS_Transport_TCPv4_Plugin_stringToTransportAddress** (p. 20) to compose the transport locator from a string form.

This field is used and required only when **NDDS_Transport_Property_t::classid** is set to **NDDS_TRANSPORT_CLASSID_TCPV4_WAN**.

The public IP address and port are necessary to support communication over WAN that involves Network Address Translators (NATs).

Typically, the address is the public address of the IP router that provides access to the WAN. The port is the IP router port that is used to reach the private **server_bind_port** (p. 50) inside the LAN from the outside.

See also:

NDDS_Transport_TCPv4_Property_t::server_bind_port (p. 50)

[default] NDDS_TRANSPORT_ADDRESS_INVALID

5.1.2.11 char* NDDS_Transport_TCPv4_Property_t::bind_interface_address

Interface IP address for the transport sockets.

The TCP transport can be configured to bind all sockets to a specified interface.

If the value is NULL, the sockets will be bound to the special IP address INADDR_ANY. This address allows the sockets to receive packets destined to any of the interfaces.

This field should be set in multi-homed systems communicating across NAT routers.

[default] NULL

5.1.2.12 RTI_INT32 NDDS_Transport_TCPv4_Property_t::server_bind_port

Private IP port (inside the LAN) used by the transport to accept TCP connections.

If this property is set to zero, the transport will disable the internal server socket, making it impossible for external peers to connect to this node. In this case, the node is considered unreachable and will only communicate using the asymmetric mode with other (reachable) peers.

For WAN communication, this port must be forwarded to a public port in the NAT-enabled router that connects to the outer network.

See also:

NDDS_Transport_TCPv4_Property_t::public_address (p. 49)

[default] NDDS_TRANSPORT_TCPV4_DEFAULT_PORT_NUMBER (p. 14)

5.1.2.13 struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::read_buffer_allocation [read]

Allocation settings applied to read buffers.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] **NDDS_TRANSPORT_TCPV4_READ_BUFFER_POOL_GROWTH_POLICY_DEFAULT** (p. 15)

5.1.2.14 struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::write_buffer_allocation [read]

Allocation settings applied to buffers used for asynchronous (nonblocking) write.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

Note that for the write buffer pool, the default max_count is not set to unlimited. This is to avoid having a fast writer quickly exhaust all the available system memory, in case of a temporary network slowdown. When this write buffer pool reaches the maximum, the low-level send command of the transport will fail; at that point RTI Data Distribution Service will take the appropriate action (retry to send or drop it), according to the applications QoS (if the transport is used for reliable communication, the data will still be sent eventually).

[default] **NDDS_TRANSPORT_TCPV4_WRITE_BUFFER_POOL_GROWTH_POLICY_DEFAULT** (p. 15)

5.1.2.15 struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_buffer_allocation [read]

Allocation settings applied to buffers used to serialize and send control messages.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] **NDDS_TRANSPORT_TCPV4_CONTROL_BUFFER_POOL_GROWTH_POLICY_DEFAULT** (p. 15)

5.1.2.16 struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_message_allocation [read]

Allocation settings applied to control messages.

These settings configure the initial number of buffers, the maximum number of buffers and the buffers to be allocated when more buffers are needed.

[default] NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_FACTORY_GROWTH_POLICY_DEFAULT (p. 15)

5.1.2.17 struct TransportAllocationSettings_t NDDS_Transport_TCPv4_Property_t::control_attribute_allocation [read]

Allocation settings applied to control messages attributes.

These settings configure the initial number of attributes, the maximum number of attributes and the attributes to be allocated when more attributes are needed.

[default] NDDS_TRANSPORT_TCPV4_CONTROL_MESSAGE_ATTRIBUTE_FACTORY_GROWTH_POLICY_DEFAULT (p. 16)

5.1.2.18 RTI_INT32 NDDS_Transport_TCPv4_Property_t::force_asynchronous_send

Controls whether the plugin will send data synchronously or asynchronously.

When this parameter is set to 0, the TCP transport will attempt to send data as soon as the internal send() function is called. When it is set to 1, the transport will make a copy of the data to send and enqueue it in an internal send buffer. Data will be sent as soon as the low-level socket buffer has space.

Normally setting it to 1 delivers better throughput in a fast network, but will result in a longer time to recover from various TCP error conditions. Setting it to 0 may cause the low-level send() function to block until the data is physically delivered to the lower socket buffer. For an application writing data at a very fast rate, it may cause the caller thread to block if the send socket buffer is full. This could produce lower throughput in those conditions (the caller thread could prepare the next packet while waiting for the send socket buffer to become available).

[default] 0 (synchronous send)

5.1.2.19 NDDS_Transport_TCPv4_OnConnectionEstablishedCallback NDDS_Transport_TCPv4_Property_t::on_connection_established

Pointer to a function that is called whenever the plugin establish a connection with a remote peer.

For plugins configured to behave as servers, this function is called every time a remote client successfully establish a data communication.

For plugins configured to behave as clients, this function is called for every successful connection to a remote server.

It can be set to NULL (no notification callbacks are performed)

IMPORTANT: This function is called from the receive thread (both client/server side). if you don't return the control fast enough, the control protocol may be affected causing disconnections and/or delays in establishing connections with remote peers.

[**default**] NULL

5.1.2.20 NDDS_Transport_TCPv4_OnConnectionLostCallback NDDS_Transport_TCPv4_Property_t::on_connection_lost

Pointer to a function that is called when a previously established connection with a remote peer gets closed.

It can be set to NULL (no notification callbacks are performed)

IMPORTANT: This function is called from the receive thread (both client/server side). if you don't return the control fast enough, the control protocol may be affected causing disconnections and/or delays in establishing connections with remote peers.

[**default**] NULL

5.1.2.21 RTI_INT32 NDDS_Transport_TCPv4_Property_t::max_packet_size

The maximum size of a TCP segment.

This parameter is only supported on Linux architectures.

By default, the maximum size of a TCP segment is based on the network MTU for destinations on a local network, or on a default 576 for destinations on non-local networks. This behavior can be changed by setting this parameter to a value between 1 and 65535.

[**default**] -1 (use OS default)

5.1.2.22 RTI_INT32 NDDS_Transport_TCPv4_Property_t::enable_keep_alive

Configures the sending of KEEP_ALIVE messages in TCP.

Setting this value to 1, causes a KEEP_ALIVE packet to be sent to the remote peer if a long time passes with no other data sent or received.

This feature is implemented only on architectures that provide a lowlevel implementation of the TCP keep-alive feature.

On Windows systems, the TCP keep-alive feature can be globally enabled through the system's registry: **\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Tcpip\Parameters**.

Refer to MSDN documentation for more details.

On Solaris systems, most of the TCP keep-alive parameters can be changed though the kernel properties.

[default] 0 (disabled)

5.1.2.23 RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_time

Specifies the interval of inactivity in seconds that causes TCP to generate a KEEP_ALIVE message.

This parameter is only supported on Linux architectures.

[default] -1 (OS default value)

5.1.2.24 RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_interval

Specifies the interval in seconds between KEEP_ALIVE retries.

This parameter is only supported on Linux architectures.

[default] -1 (OS default value)

5.1.2.25 RTI_INT32 NDDS_Transport_TCPv4_Property_t::keep_alive_retry_count

The maximum number of KEEP_ALIVE retries before dropping the connection.

This parameter is only supported on Linux architectures.

[default] -1 (use system's default).

5.1.2.26 RTI_INT32 NDDS_Transport_TCPv4_Property_t::disable_nagle

Disables the TCP nagle algorithm.

When this property is set to 1, TCP segments are always sent as soon as possible, which may result in poor network utilization.

[**default**] 0 (disabled)

5.1.2.27 RTI_INT32 NDDS_Transport_TCPv4_Property_t::logging_verbosity_bitmap

Bitmap that specifies the verbosity of log messages from the transport.

Logging values:

- -1 (0xffffffff): do not change the current verbosity
- 0x00: silence
- 0x01: errors
- 0x02: warnings
- 0x04: local
- 0x08: remote
- 0x10: period
- 0x80: other (used for control protocol tracing)

Note: the logging verbosity is a global property shared across multiple instances of the TCP transport. If you create a new TCP Transport instance with logging_verbosity_bitmap different than -1, the change will affect all the other instances as well.

The default TCP transport verbosity is errors and warnings.

Note: The option of 0x80 (other) is used only for tracing the internal control protocol. Since the output is very verbose, this feature is enabled only in the debug version of the TCP Transport library.

[**default**] -1 (do not change default verbosity).

5.1.2.28 RTI_INT32 NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies

Maximum number of outstanding connection cookies allowed by the transport when acting as server.

A connection cookie is a token provided by a server to a client; it is used to establish a data connection. Until the data connection is established, the cookie cannot be reused by the server.

To avoid wasting memory, it is good practice to set a cap to the maximum number of connection cookies (pending connections).

When the maximum value is reached, a client will not be able to connect to the server until new cookies become available.

[default] 100

5.1.2.29 RTI_INT32 NDDS_Transport_TCPv4_Property_t::outstanding_connection_cookies_life_span

Maximum lifespan (in seconds) of the cookies associated with pending connections.

If a client does not connect to the server before the lifespan of its cookie expires, it will have to request a new cookie.

Range: 1 second or higher, or -1.

[default] -1, which means an unlimited amount of time (effectively disabling the feature).

5.1.2.30 RTI_INT32 NDDS_Transport_TCPv4_Property_t::send_max_wait_sec

Maximum number of seconds a low-level TCP sendto() function is allowed to block.

This property controls the maximum time (in seconds) the low level sendto() is allowed to block the caller thread when the TCP send buffer becomes full.

If the bandwidth used by the transport is limited, and the sender thread try to push data faster than the OS can handle, the low level sendto() function will block the caller until there is some room available on the queue.

By limiting this delay we eliminate possibility of deadlock and increase the response time of the internal DDS thread.

This property affects both CONTROL and DATA stream, and affect only SYNCHRONOUS send operations. Asynchronous send never blocks a send operation.

For synchronous send() this property limit the time the DDS sender thread can block for a send buffer full. If is too large, DDS not only won't be able to send more data, but it won't be able to receive any more data because of an internal resource mutex.

Setting this property to 0 cause the low level function to report an immediate failure if the TCP send buffer is full.

Setting this property to -1 cause the low level function to block forever until space becomes available in the TCP buffer

[default] 3 seconds.

5.1.2.31 struct RTITLS_OpenSSL_Configuration NDDS_Transport_TCPv4_Property_t::tls [read]

OpenSSL TLS parameters.

[default] **RTITLS_OpenSSL::RTITLS_OPENSSL_CONFIGURATION_DEFAULT** (p. 28)

5.2 RTITLS_Ciphers Struct Reference

Set of TLS properties for cipher configuration.

Data Fields

- const char * **cipher_list**

  List of available TLS ciphers.

- **DDS_Long dh_param_files_length**

  Number of DH key files supplied.

- struct **RTITLS_DHParamFile** * **dh_param_files**

  List of available DH key files.

- const char * **engine_id**

  ID of OpenSSL cipher engine to request.

5.2.1 Detailed Description

Set of TLS properties for cipher configuration.

5.2.2 Field Documentation

5.2.2.1 const char* RTITLS_Ciphers::cipher_list

List of available TLS ciphers.

See the OpenSSL manual page for SSL_set_cipher_list(3) or ciphers(1) for more information on the format of this string. Some typical values are defined: **RTITLS_Plugin::RTITLS_CIPHER_LIST_DEFAULT** (p. 27), **RTITLS_Plugin::RTITLS_CIPHER_LIST_ENCRYPT_HIGH** (p. 27), and **RTITLS_Plugin::RTITLS_CIPHER_LIST_UNENCRYPTED** (p. 27).

[default] NULL

5.2.2.2 DDS_Long RTITLS_Ciphers::dh_param_files_length

Number of DH key files supplied.

[default] 0

5.2.2.3 struct RTITLS_DHParamFile* RTITLS_Ciphers::dh_param_files [read]

List of available DH key files.

[**default**] NULL

5.2.2.4 const char* RTITLS_Ciphers::engine_id

ID of OpenSSL cipher engine to request.

[**default**] NULL

5.3 RTITLS_DHParamFile Struct Reference

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

Data Fields

- const char * **file**

  Name of DH key file.

- **DDS_Long bits**

  Length of DH key in bits.

5.3.1 Detailed Description

Name of a Diffie-Helman (DH) key file and the length of the contained key in bits.

5.3.2 Field Documentation

5.3.2.1 const char* RTITLS_DHParamFile::file

Name of DH key file.

[default] 0

5.3.2.2 DDS_Long RTITLS_DHParamFile::bits

Length of DH key in bits.

[default] NULL

5.4 RTITLS_Identity Struct Reference

Set of TLS properties for identity.

Data Fields

- const char * **certificate_chain**

  String containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

- const char * **certificate_chain_file**

  File containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

- const char * **private_key_password**

  Password for private key.

- const char * **private_key**

  String containing private key (in PEM format).

- const char * **private_key_file**

  File containing private key (in PEM format).

- const char * **rsa_private_key**

  String containing additional RSA private key (in PEM format).

- const char * **rsa_private_key_file**

  File containing additional RSA private key (in PEM format).

5.4.1 Detailed Description

Set of TLS properties for identity.

5.4.2 Field Documentation

5.4.2.1 const char* RTITLS_Identity::certificate_chain

String containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

An identifying certificate is required for secure communication. The string must be sorted starting with the certificate to the highest level (root CA). If this is specified certificate_chain_file must be empty.

[default] NULL

5.4.2.2 const char* RTITLS_Identity::certificate_chain_file

File containing identifying certificate (in PEM format) or certificate chain (appending intermediate CA certs in order).

An identifying certificate is required for secure communication. The file must be sorted starting with the certificate to the highest level (root CA). If this is specified certificate_chain must be empty.

Optionally, a private key may be appended to this file. If no private key option is specified, this file will be used to load a private key.

[default] NULL

5.4.2.3 const char* RTITLS_Identity::private_key_password

Password for private key.

[default] NULL (no password)

5.4.2.4 const char* RTITLS_Identity::private_key

String containing private key (in PEM format).

At most one of private_key and private_key_file may be specified. If no private key is specified (all values are NULL), the private key will be read from the certificate chain file.

[default] NULL

5.4.2.5 const char* RTITLS_Identity::private_key_file

File containing private key (in PEM format).

At most one of private_key and private_key_file may be specified. If no private key is specified (all values are NULL), the private key will be read from the certificate chain file.

[default] NULL

5.4.2.6 const char* RTITLS_Identity::rsa_private_key

String containing additional RSA private key (in PEM format).

For use if both an RSA and non-RSA key are required for the selected cipher. At most one of rsa_private_key and rsa_private_key_file may be specified.

[**default**] NULL

5.4.2.7 const char* RTITLS_Identity::rsa_private_key_file

File containing additional RSA private key (in PEM format).

For use if both an RSA and non-RSA key are required for the selected cipher. At most one of rsa_private_key and rsa_private_key_file may be specified.

[**default**] NULL

5.5 RTITLS_OpenSSL_Configuration Struct Reference

Full set of TLS properties.

Data Fields

- struct **RTITLS_Verification verify**
  Verification properties.
- struct **RTITLS_Identity identity**
  Identity properties.
- struct **RTITLS_Ciphers cipher**
  Cipher properties.

5.5.1 Detailed Description

Full set of TLS properties.

Should be initialized with **RTITLS_Plugin::RTITLS_OPENSSL_CONFIGURATION_DEFAULT** (p. 28)

5.5.2 Field Documentation

5.5.2.1 struct RTITLS_Verification RTITLS_OpenSSL_Configuration::verify [read]

Verification properties.

5.5.2.2 struct RTITLS_Identity RTITLS_OpenSSL_Configuration::identity [read]

Identity properties.

5.5.2.3 struct RTITLS_Ciphers RTITLS_OpenSSL_Configuration::cipher [read]

Cipher properties.

5.6 RTITLS_Verification Struct Reference

Set of TLS properties for certificate authorities (CAs) and verification.

Data Fields

- const char * **ca_file**

  Name of file containing Certificate Authority certificates.

- const char * **ca_path**

  Paths to directories containing Certificate Authority certificates.

- **DDS_Long verify_depth**

  Maximum certificate chain length for verification.

- **RTITLS_Verify_Callback callback**

  Callback used to verify peer certificates.

- const char * **crl_file**

  Name of file containing Certificate Revocation List.

5.6.1 Detailed Description

Set of TLS properties for certificate authorities (CAs) and verification.

5.6.2 Field Documentation

5.6.2.1 const char* RTITLS_Verification::ca_file

Name of file containing Certificate Authority certificates.

File should be in PEM format. See the OpenSSL manual page for SSL_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[**default**] NULL

5.6.2.2 const char* RTITLS_Verification::ca_path

Paths to directories containing Certificate Authority certificates.

Files should be in PEM format, and follow the OpenSSL-required naming conventions. See the OpenSSL manual page for SSL_CTX_load_verify_locations for more information.

At least one of ca_file and ca_path must be specified; both may be specified.

[default] NULL

5.6.2.3 DDS_Long RTITLS_Verification::verify_depth

Maximum certificate chain length for verification.

[default] -1 (no limit)

5.6.2.4 RTITLS_Verify_Callback RTITLS_Verification::callback

Callback used to verify peer certificates.

See the OpenSSL manual page for SSL_set_verify for more information. There are a number of default callbacks included in the Secure Transport. See **RTITLS_default_verify_callback()** (p. 28) , **RTITLS_verbose_verify_callback()** (p. 29) .

[default] NULL (use **RTITLS_default_verify_callback()** (p. 28))

5.6.2.5 const char* RTITLS_Verification::crl_file

Name of file containing Certificate Revocation List.

File should be in PEM format.

[default] NULL