TCNOpen	**Open Source Developement**

TRDP

Train Real Time Data Protocol

System Architecture & Design Specification

Document reference no: TCN-TRDP2-D-BOM-O19-01

Author :	Bernd Löhr
Organisation :	Bombardier
Document date:	31 May 2012
Revision:	1
Status:	draft

Dissemination Level		
PU	Public	
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

DOCUMENT SUMMARY SHEET

Participants		
Name and Surname	Organisation	Role

History			
V1	31 May 12	Bernd Löhr	Initial version

Table of Contents

Table of Figures

Table of Tables

1. Introduction

1.1. Purpose

This document describes the basic design of the TRDP protocol stack and may aid in extending and including further protocol features apart from process data handling.

1.2. Intended Audience

The intended audience is the programmers, who want to add additional features and/or want to support more target systems.

1.3. References/Related Documents

Reference	Number	Title
[Wire]	IEC51375-3-2	TRDP Protocol (Annex A)
[1]	refman.pdf	TRDP Light (generated from source code by Doxygen)

Table 1: References

1.4. Abbreviations and Definitions

Abbreviation	Definition
API	Application Programming Interface
IDE	Integrated Development Environment
MD	Message Data
PD	Process Data
QoS	Quality of Service
TAU	TRDP Application Utility (Function calling prefix)
TLC	TRDP Light Common (Function calling prefix)
TLP	TRDP Light Process data (Function calling prefix)
TLM	TRDP Light Message data (Function calling prefix)
TTL	Time To Live
VOS	Virtual Operating System

Table 2: Abbreviations and Definitions

2. System Description

2.1.1. Overview

The TRDP prototype stack consists of the following functional groups:

- Application/Session layer handling
- Process Data handling (publish/subscribe)
- Optional Message Data handling
- Statistics handling
- Virtual Operating System

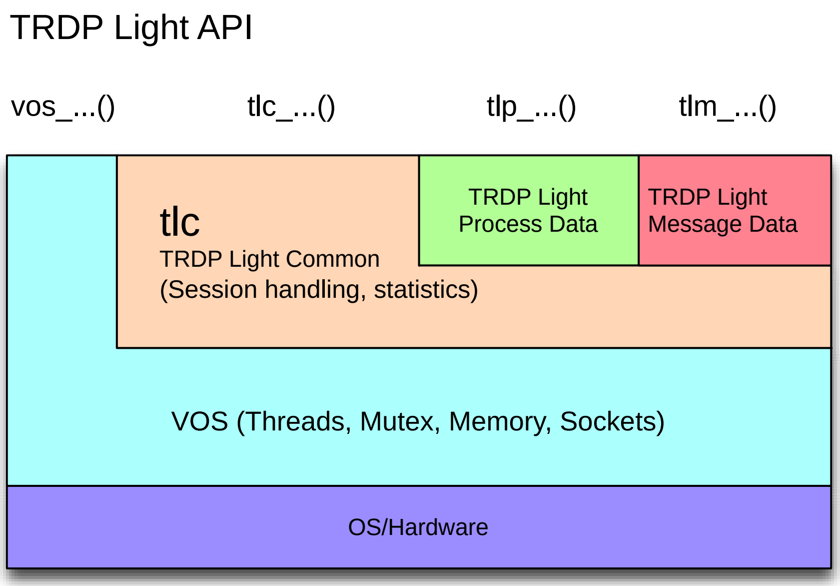

Figure 1: Functional Layers

2.1.2. Application/Session layer handling

The application/session layer functions handle the basic resource management – they register a session of a client application and control the overall behaviour and options for that session. The same application can have several sessions open at the same time.
Each session is internally represented by an instance of the data object `TRDP_SESSION_T`. Sessions are maintained by a singly linked list.

The first function an application must call before any other call to the protocol stack can succeed, is `tlc_init()`.
Each call to `tlc_init()` will create a new session element and will initialize it with the supplied default parameters.
Only the very first call to `tlc_init()` will reserve memory and will create a recursive mutex to eventually protect concurrent access to the session data.

On successful initialization a session handle will be returned and will be used by the caller to select the right session. If the application quits or doesn't need the session anymore, it should release it by calling `tlc_terminate()`.
The session handle is actually the pointer to the memory area reserved for the session and must be checked for validity before dereferencing (using `isValidSession()`).

To allow different packet options for PD, like different TTLs and QoS values, and open sockets are quite expensive, the number of concurrent sockets should be limited. In `tlc_init()` a pool of socket descriptors is created and maintained. If a new publish call is done the pool of already open sockets is searched for the same send parameters and source address. Either a new or an already open socket will be returned and a reference counter incremented. Handling of the socket pool is done through the functions `trdp_initSockets()`, `trdp_requestSocket()` and `trdp_releaseSocket()` in `trdp_utils.c`.

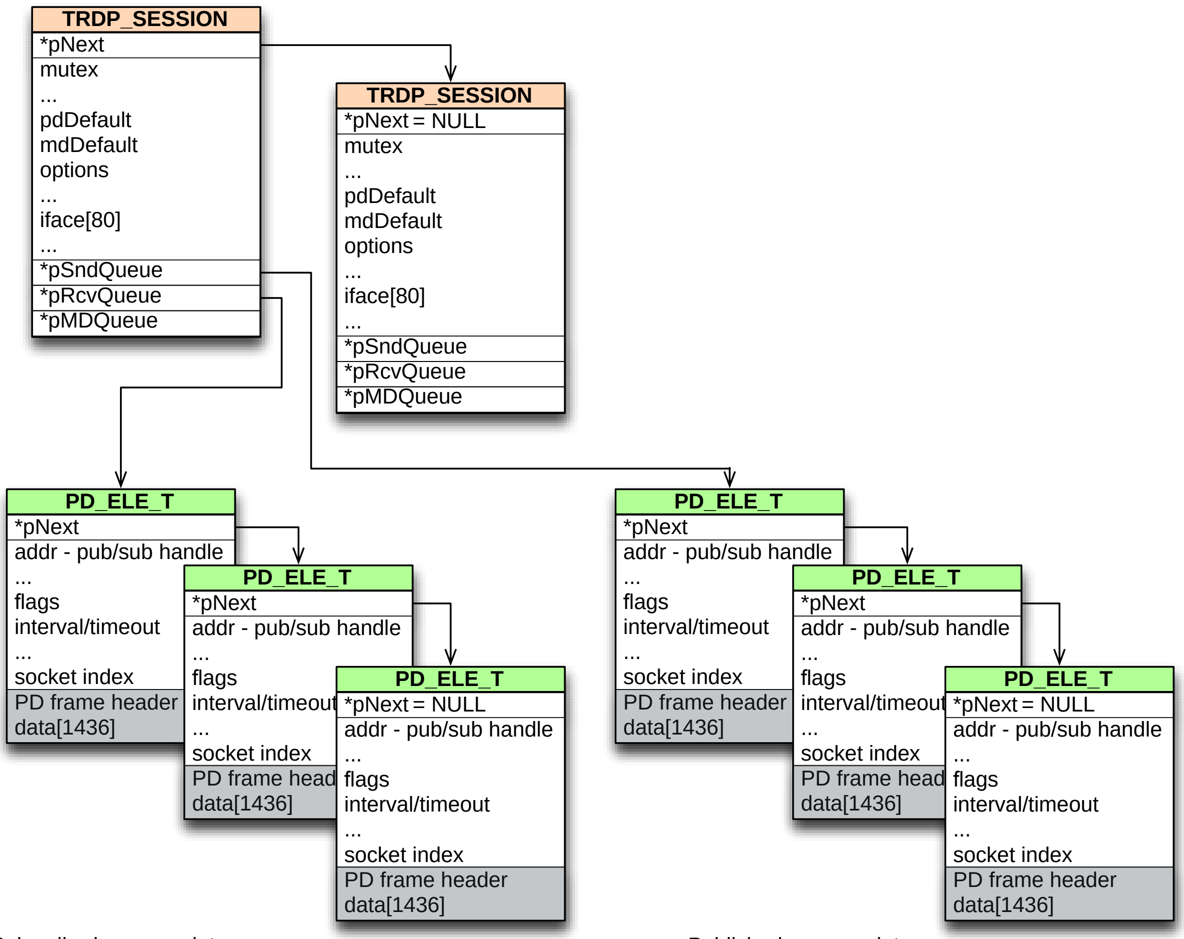

Figure 2: Main Data Structures

2.1.3. Process Data

For process data handling, two queues are maintained – a send and a receive queue. The send queue is filled or diminished by `tlp_publish()` resp. `tlp_unpublish()`.

2.1.3.1. Publish

On publishing, the caller supplies all necessary data to construct the PD header:

- comId
- current topology counter
- source and destination IP address
- interval
- …
- send parameters
- pointer to initial data

`tlp_publish()` will request an adequate socket and will create and insert a new queue element into the send queue of the used session. Data is initially copied into the element by calling `tlp_put()`. Data can be updated at any time later using `tlp_put()`. It will be sent only when `tlc_process()` is called, though!

2.1.3.2. Subscribe

On subscribing, the functional flow is very similar to publishing; a new element is inserted into the receive queue instead and an eventual multicast group may be joined.
On `tlp_unsubscribe()` the subscription element will be released from the queue.

2.1.4. Message Data

Message data handling is a compile time option, which is controlled by `#define MD_SUPPORT`.
Message data will be handled in `tlc_init()`, `tlc_process()`, `tlc_getInterval()` and `tlc_terminate()`. For message data, an `MD_ELE_T` element must be created for each transaction to perform.

Tbd

2.1.5. Processing

`tlc_process()` is the main working function. It is the only function where sending and receiving takes place. Depending on the selected mode, whether blocking/non-blocking, using the `select()` function or polling, receiving process data is done.
`tlc_process()` has to be called at least at the lowest interval that has been defined when publishing process data. The send queue is checked for the next pending packet – if it is due, `trdp_pdSend()` is called.
Next job is to look for timeouts of subscribed packets. If a late packet is detected, the PD_ELE entry is marked as overdue and, if call-backs are enabled, the supplied user function will be called.
If blocking mode is on (`select()` is used), the receive queue is scanned for the ready socket descriptor and `trdp_pdReceive()` is called, getting new packet data. Again, the user supplied call-back may be called.
After process data has been handled, optionally message data will be processed depending on the state of the transaction kept in the relevant `mdQueue` element.

Note: Call-backs will only be called from within `tlc_process()`. `tlc_process()` must not be called from within a call-back.

To support the application in determining the right time to call `tlc_process()` and to minimize CPU usage through polling, the function `tlc_getInterval()` checks all queues for pending packets to send or receive and computes the next time when `tlc_process()` must be called before any timing constraints may hit. `tlc_getInterval()` should always be called before `tlc_process()`.

Sample calling sequences are shown below.

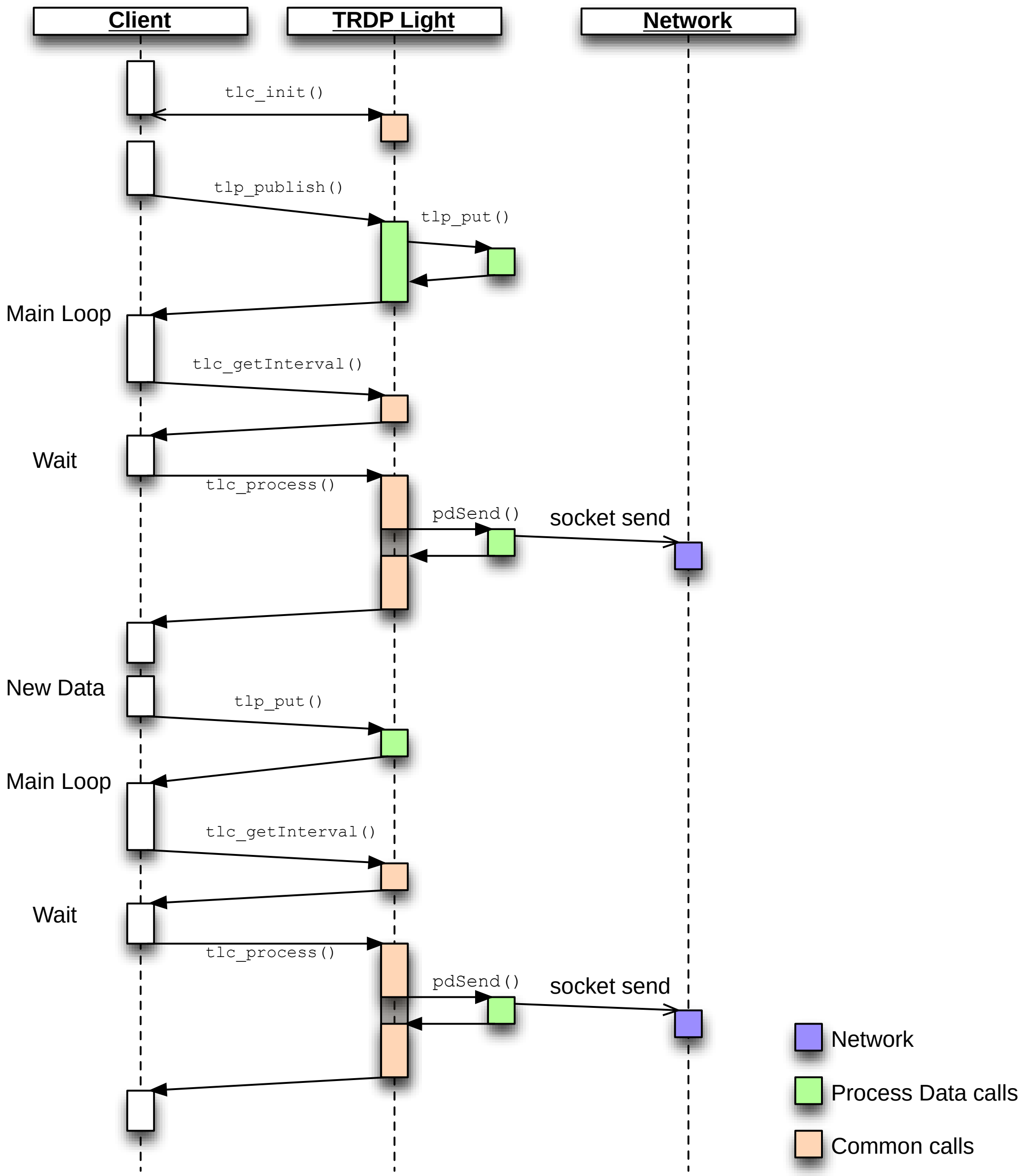

Figure 3: Sending Process Data

When receiving data (subscribe), data can be fetched either by using call-backs (out of `tlc_process()`) or at any time by polling with `tlp_get()`.
A call-back, if enabled, will only originate from `tlc_process()` context.

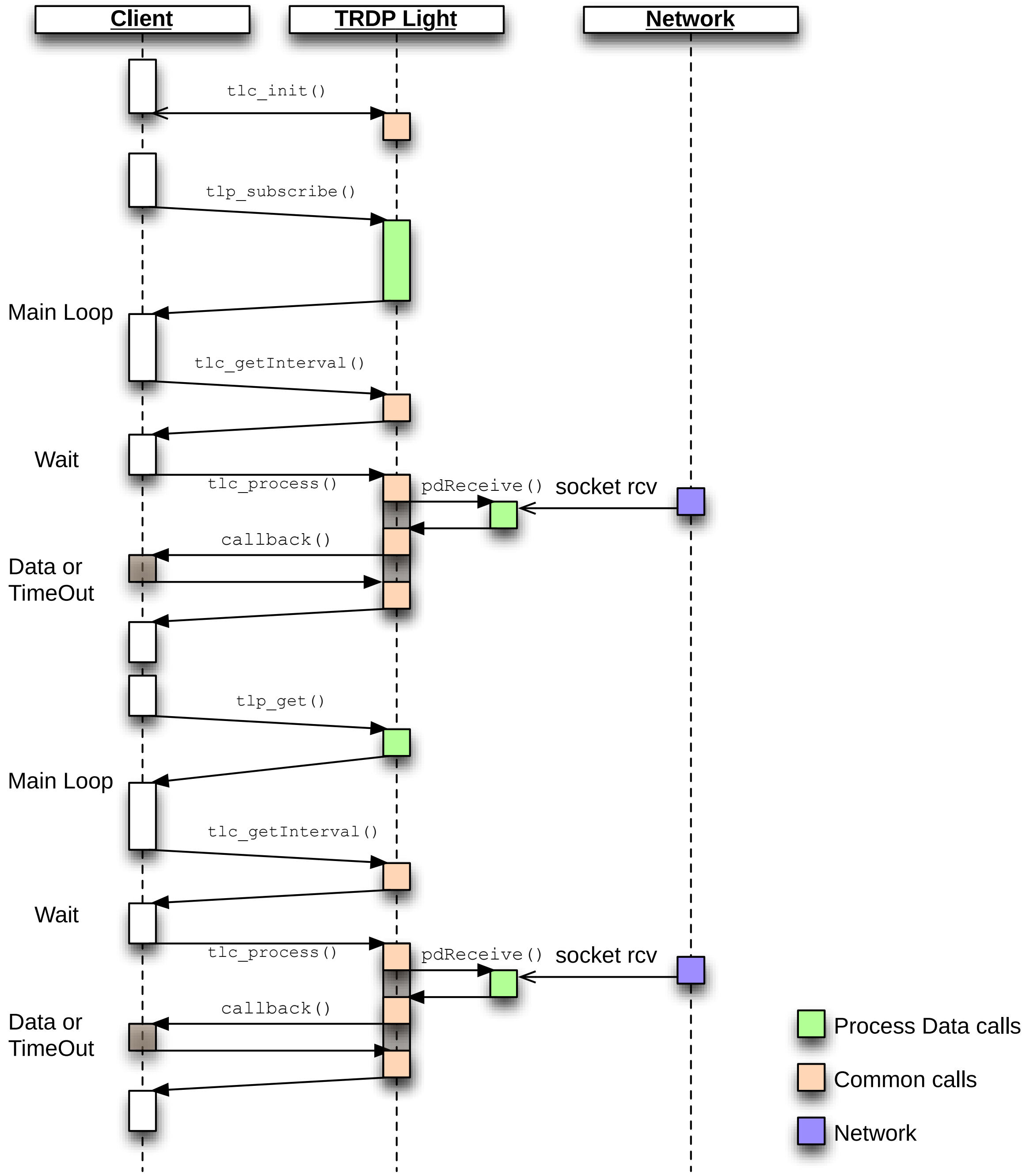

Figure 4: Call Sequence for Subscription

Detailed description of the PDCom-related functions can be found in [1].

2.1.6. Statistics

All functions effectively taking part in communication will update the counters kept in the global variable `TRDP_STATISTICS_T gStats` in the file `trdp_stats.c`.

To prevent accessing a global variable directly but prevent unnecessary overhead by real accessor functions, inline/makro functions are defined to set and increment specific members.

…

2.1.7. VOS

All calls to system dependent services like memory, network, CPU (threading) are mapped over an abstraction layer. Target dependent implementations are located in directories named after their targets.

Selecting a specific target involves changes of the `Makefile` and/or IDE project settings, see 2.2.1 and 2.2.2.

2.1.8. Source Files

File	Content
echoPolling.c	Demo echoing application for TRDP
echoSelect.c	Demo echoing application for TRDP
sendHello.c	Demo sending application for TRDP
tau_addr.h	TRDP address resolution declarations
tau_marshall.h	TRDP marshalling declarations
tau_types.h	TRDP utility interface declarations
tau_xml.h	TRDP configuration interface declarations
trdp_if.c	TRDP Light interface functions
trdp_if_light.h	TRDP Light interface declarations (API)
trdp_mdcom.c	Functions for MDcommunication
trdp_mdcom.h	Functions for MDcommunication
trdp_pdcom.c	Functions for PDcommunication
trdp_pdcom.h	Functions for PDcommunication
trdp_private.h	Typedefs for TRDPcommunication
trdp_types.h	Typedefs for TRDP communication
trdp_utils.c	Helper functions for TRDPcommunication
trdp_utils.h	Commonutilities for TRDPcommunication
vos_mem.c	Memory functions

File	Content
echoPolling.c	Demo echoing application for TRDP
echoSelect.c	Demo echoing application for TRDP
vos_mem.h	Memory and queue functions for OS abstraction
vos_sock.c	Socket functions
vos_sock.h	Typedefs for OS abstraction
vos_thread.c	Multitasking functions

2.2. Build System

2.2.1. Command Line

On the top level of the project, a `Makefile` preset for the 'POSIX' target can be invoked.

With the command

```
Make help
```

the possible options are listed. For different targets and compilers, the head of the Makefile allows to change several path and configuration variables.

To generate the documentation from the source files, 'Doxygen' version 1.5 or higher must be installed on the host system. Depending on your installation it might be necessary to adjust a path in the file `Doxyfile` on the same level.

2.2.2. IDE

2.2.2.1. Xcode

The folder Xcode contains a project file for the Mac OS X Xcode IDE version 3.2 with several predefined targets.

2.2.2.2. Visual C

n/a