Zurich University of Applied Sciences
School of Engineering
InES Institute of Embedded Systems

PRP-1 stack

Authors:	S. Meier
Creation date:	08 February 2007
Last modification:	24 May 2012
Version:	1.0.7
State:	released

Contact address

Institute of Embedded Systems (InES)
Zurich University of Applied Sciences
Technikumstr. 22
Postfach
CH-8401 Winterthur

Tel: +41 58 934 75 25
Fax: +41 58 935 75 25

Mail: support.ines@zhaw.ch
http: ines.zhaw.ch/

Document

Distribution: ZHAW – InES

History

Version	Date	Author/Revisor	Description
1.0.1	08. Feb. 2007	Meier Sven	Created
1.0.2	16. Mar. 2007	Meier Sven	Changed the API etc
1.0.3	20. Mar. 2007	Meier Sven	Updated some classes
1.0.4	19. Sep. 2007	Meier Sven	SRP stuff added
1.0.5	13. Feb. 2008	Meier Sven	Updated Graphics etc
1.0.6	01. Jul. 2011	Weibel Hans	
1.0.7	01. May 2012	Walch Oliver	PRP-1 Updates

Contents

1 About this document

This is a design and implementation description of the Parallel Redundancy Protocol (PRP-1) Software.

1.1 The PRP-1 Stack

The requirements are as followed:

The Parallel Redundancy Protocol Stack shall support the whole PRP-1 standard.
It shall allow running IEEE1588/PTP.
It shall be manageable.
It shall be easy portable to other Operating Systems.
It shall be as fast as possible.
It shall use as little resources as possible.
It shall be extendable.
It shall have a clear API.

With these requirements in mind the design described in the following chapters was chosen.

2 Design

2.1 Block Diagram

Fehler! Verweisquelle konnte nicht gefunden werden. illustrates the relations between the different components of a PRP device.

Each PRP device owns a PRP Environment and a PRP Node Table, which is embedded into the PRP Environment. The PRP Environment provides the protocol functionalities like Discard Algorithm, Supervision or Redundancy Control Trailer as well as global and node specific data. The interfaces (PRP, PRP_xxxItf) decouple all the operating system dependent functions from the PRP software which is generic. These interfaces are implemented as singletons what means that every interface exists only once. The access layer PRP (API) is the communication path between user programs and the PRP software.

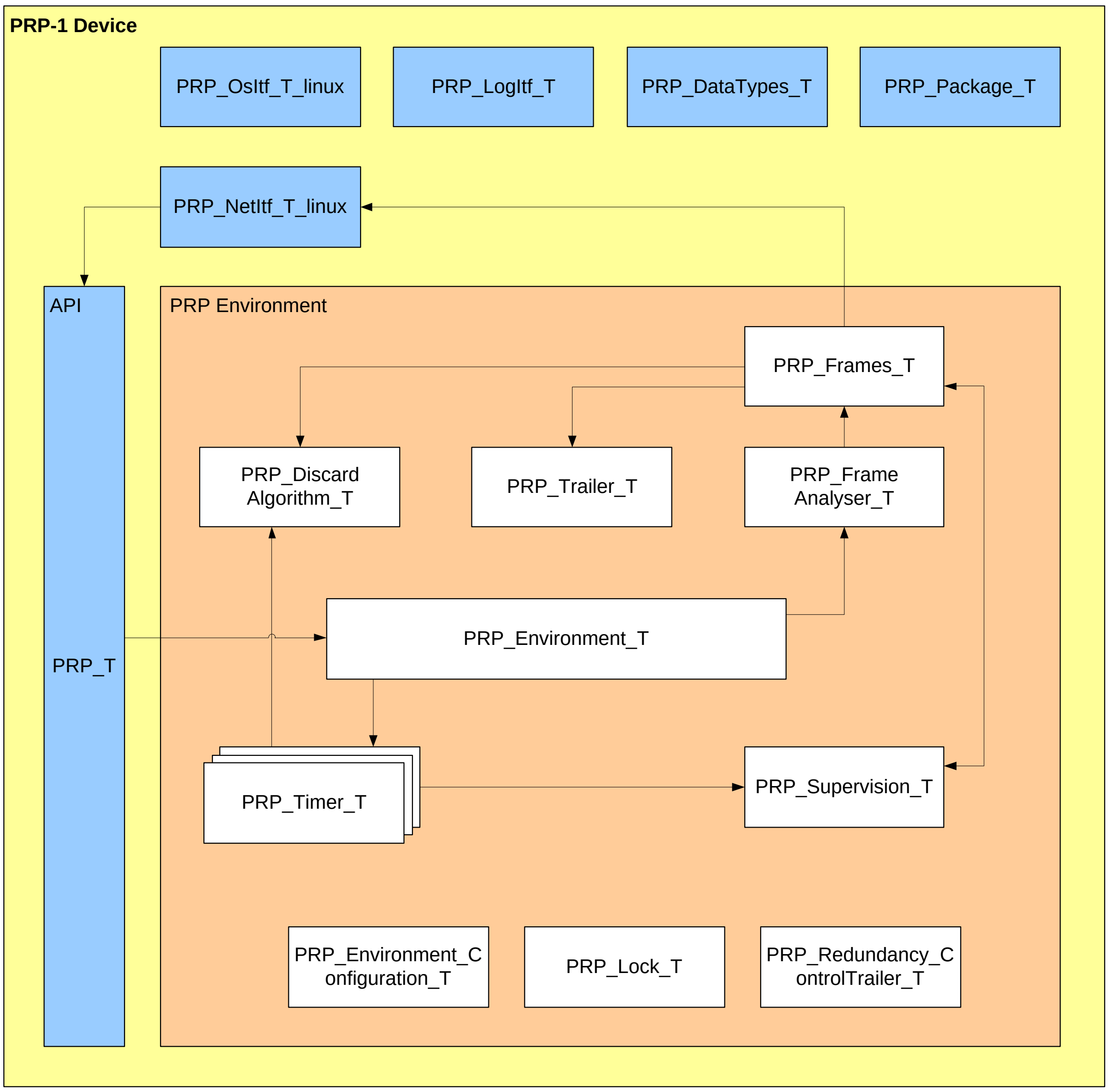

Figure 1 Components of the PRP-1 SW stack

2.2 Mode of operation

Basically the following steps are executed when an event occurs:

An event occurs (e.g. a frame was received).
The respective API function gets called (e.g. receive()).
The PRP_Environment invokes further actions depending on the current event (e.g. pass the received frame to the PRP_Frame_Analyser).

2.3 Layer Diagram

PRP-1 software mainly consists of two layers. One layer is the protocol engine – it's generic and hasn't got to be changed at all in order to run on a specific operating system. It's implemented according to the PRP-1 standard. The other layer is the OS abstraction layer with the interfaces – this is where all the environment dependent code is located at. The interfaces have to be adapted to the OS so the protocol engine has access to the used resources which the OS -environment provides. The layers are shown in **Fehler! Verweisquelle konnte nicht gefunden werden.**.

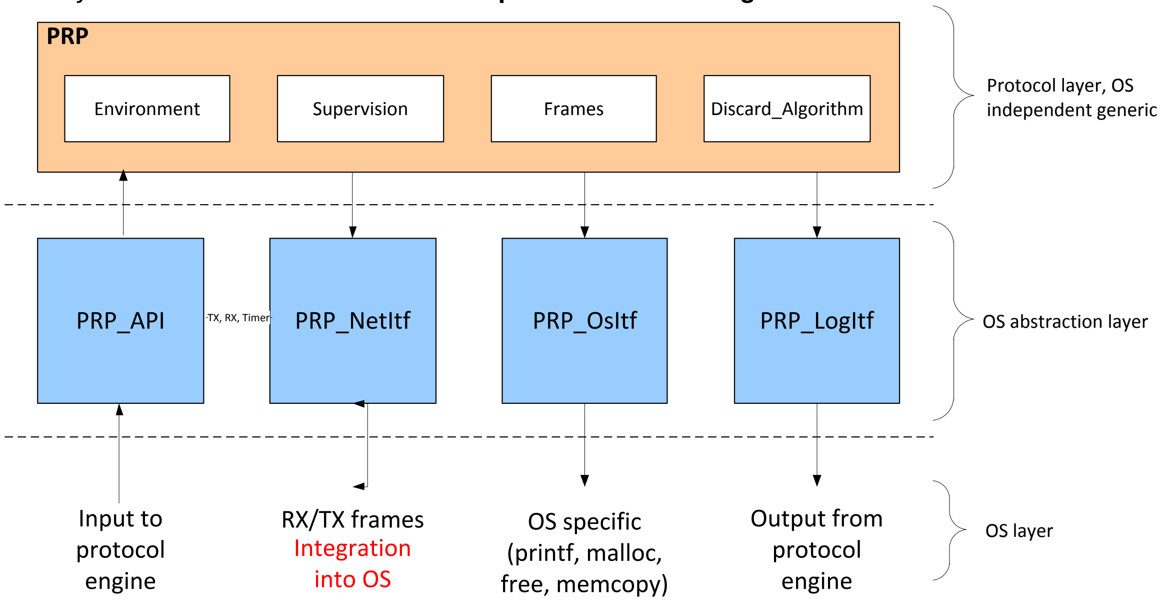

Figure 2 Layers and interaction

2.4 Components

2.4.1.1 PRP_Environment_T

The PRP_Environment_T is kind of a multiplexer. It forwards API calls coming from the PRP_API to the respective class.

2.4.1.2 PRP_Frames_T

This is the actual core of the design. It adds or removes the RCT and creates duplicates in the transmission path.

2.4.1.3 PRP_FrameAnalyser_T

The PRP_FrameAnalyser_T checks the frame type, whether it is a normal, a PTP or a supervision frame, and forwards it to the respective handler.

2.4.1.4 PRP_DiscardAlgorithm_T

This module implements the discard algorithm, the actual heart of the protocol.

2.4.1.5 PRP_Supervision_T

The PRP_Supervision_T block sends and receives supervision frames.

2.4.1.6 PRP_Timer_T

This is the Timer needed for sending supervision frames and for duplicate detection aging.

2.4.1.7 PRP_Lock_T

This block is used to guarantee atomic access on shared resources.

2.4.1.8 PRP_Trailer_T

This module provides functions to get, add and remove the RCT.

2.4.1.9 PRP_EnvironmentConfiguration_T

The PRP_EnvironmentConfiguration_T provides the general information about the protocol engine defined in the PRP-1 standard.

2.4.1.10 PRP_T

The PRP_T is the API of the PRP-1 software. It encapsulates the functions of the PRP_Environment_T that must be called from outside. It guarantees atomic access to the protocol engine.

2.4.1.11 PRP_OsItf_T

The whole software runs in user space to get an OS independent protocol core.

2.4.1.12 PRP_NetIf_T

This PRP_NetItf_T module provides functions to transmit and receive frames. It is the wrapper class around the protocol core. The integration into the network stack of the operating system is done here.

2.4.1.13 PRP_LogItf_T

The PRP_LogItf_T is the output interface of the software.

3 Implementation

The implementation of PRP-1 is done with object-oriented C. So the advantages of object-oriented analysis / design / programming can be applied (also methodology, encapsulating, reusability etc.).

3.1 Object-oriented C

This clause shows how a C++ class is realised in C.

A C++ class:

```
/*** PRP_Timer class in C++ ***/
class PRP_Timer
{
   /** Attributes */
   private:
            boolean enabled_;
            Unsigned32 value_;
            Unsigned32 timeout_;

   /** Methods */                                              Constructor
   public:
            PRP_Timer( Unsigned32 timeout );                    Destructor

            ~PRP_Timer();


            tick();
            restart();
            stop();
}
```

The same class PRP_Timer in C is realised as the following.

```
/*** PRP_Timer class in C ***/
typedef struct PRP_Timer_T PRP_Timer_T
{
   /** Attributes */
   boolean enabled_;
   Unsigned32 value_;
   Unsigned32 timeout_;
};
                                                                Constructor
/** Methods */
PRP_Timer_Init( PRP_Timer_T *const me, Unsigned32 timeout );    Destructor

PRP_Timer_Cleanup(PRP_Timer_T *const me );


PRP_Timer_tick(PRP_Timer_T *const me );
PTP_Timer_restart(PTP_Timer_T *const me );
PTP_Timer_stop(PTP_Timer_T *const me );
```

The attributes of a class are encapsulated in a structure. To avoid the struct keyword, the class struct PRP_Timer_T is defined as a new type: typedef struct PRP_Timer_T. Because there is no this pointer in C, a pointer of type of this class is passed to every function of this class. The names of the methods (or functions) of a class shall always begin with the class name. Constructor and destructor shall be named <CLASSNAME>_Init and <CLASSNAME>_Cleanup respectively. Some classes own functions named <CLASSNAME>_Create and <CLASSNAME>_Destroy. The _Create function creates an instance of the class dynamically and the _Destroy function destroys the instance and frees the allocated memory. These two functions also call the _Init and _Cleanup functions.

3.2 API and Interfaces

3.2.1 PRP_T

This class represents the API of the PRP-1 stack and describes the functions which can be called from outside in order to impact it. PRP_T handles inputs to the stack as function calls. Every time an API function is called, it will pass a lock saved section, so every API call is atomic. This class is implemented as a singleton therefore there is no me* pointer.
In order to ensure an error free operation of the PRP-1 Stack, inputs have to be passed to the stack by (only) using the API functions. PRP_T is thread save and has not to be adapted for a specific OS.

3.2.2 PRP_NetItf_T

The Network Interface integrates the protocol engine into the network stack and is therefore the wrapper class around the OS independent protocol engine. The network interface must receive frames from the two Adapters and must be able to send to the two Adapters. All traffic to the two interfaces must pass this Interface. On receiving the OS depending receive function will call the API PTP_T_receive() function. In the sending path it will forward the frames coming from the upper layers to the API PTP_T_transmit() function. The protocol engine will call the PRP_NetItf_T_transmit() function. This Interface also provides functions to change properties of the two adapters. PRP_NetItf_T has to be adapted for a specific OS.

3.2.3 PRP_LogItf_T

This interface provides log macros for outputs of the protocol engine. There are several different log levels. PRP_LogItf_T has not to be adapted for a specific OS (but maybe for a not GNU compliant C compiler because list of arguments is not supported by all compilers).

3.2.4 PRP_OsItf_T

The Operating System Interface abstracts functions like printf, malloc, free, memcpy etc.
PRP_OsItf_T has to be adapted for a specific OS.

3.2.5 PRP_Environment_T

The Environment is the central part of the protocol engine. It distributes function calls coming from the API to the respective classes. It also runs the timers.

3.2.6 PRP_EnvironmentConfiguration_T

The environment configuration interface contains general information about the protocol engine and its environment.

3.2.7 PRP_Lock_T

This class provides the functions to create locked sections.

3.2.8 PRP_Supervision_T

The Supervision class is one of the core classes it processes received Supervision frames as well as it send periodically Supervision by itself.

3.2.9 PRP_DiscardAlgorithm_T

This class is another core class. It implements the discard algorithm specified in the standard. The decision whether to keep a frame or not is done here.

3.2.10 PRP_FrameAnalyser_T

The Frame Analyzer checks the frame type and forwards the frame to the respective frame handler.

3.2.11 PRP_Frames_T

The Frames class is the core class of the whole protocol engine. It adds/removes the RCT, runs the discard algorithm, etc.

3.2.12 PRP_Trailer_T

The Trailer class provides functions to add/remove/get the RCT to/from the frame.

3.2.13 PRP_Timer_T

The Timer class is to run scheduled task like Supervision frame transmission or handling the aging for the duplicate detection.

3.3 Frame Path Diagrams

3.3.1 Normal/Supervision frame receiving

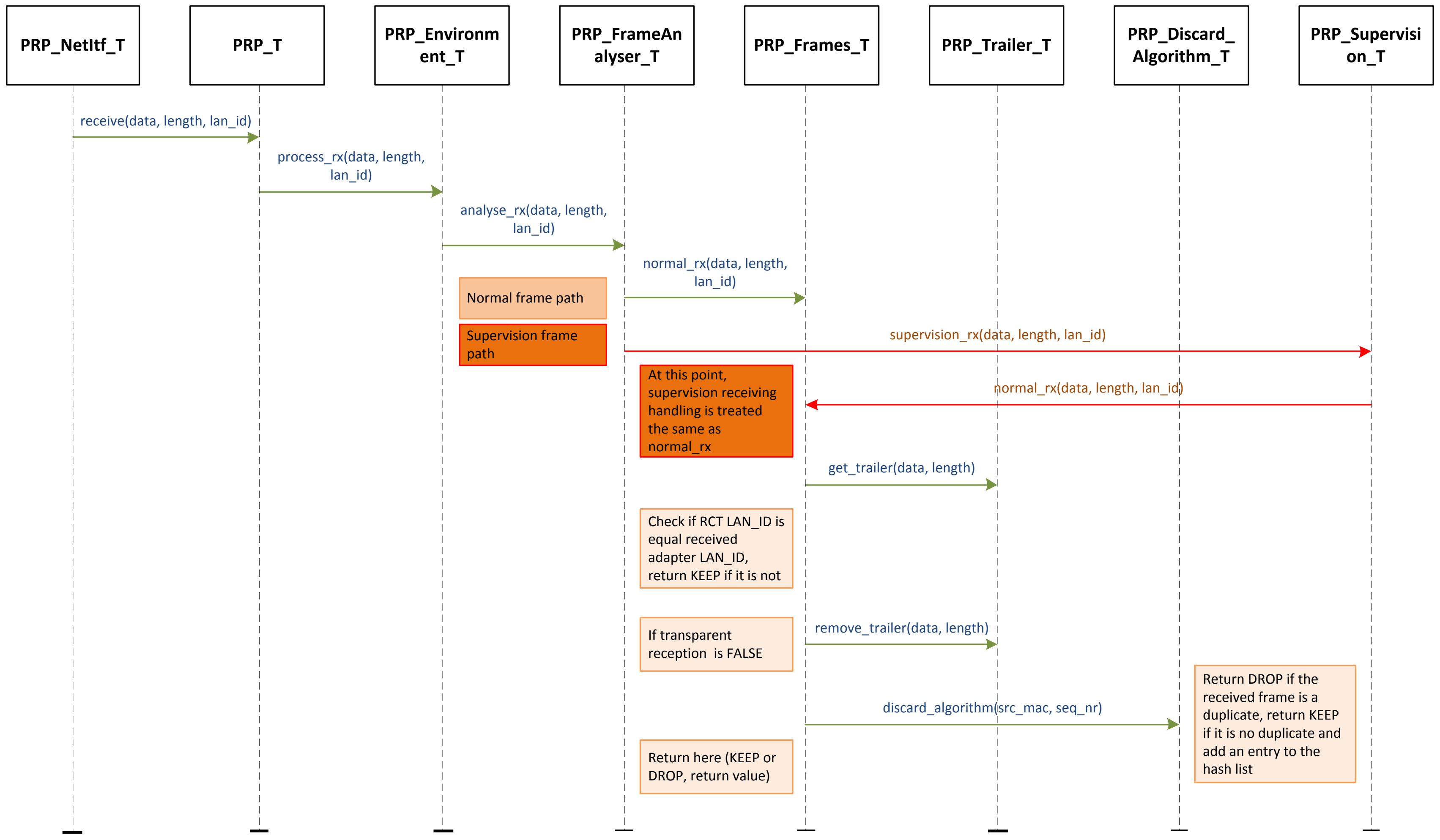
PRP_NetItf_T
PRP_T
PRP_Environment_T
PRP_FrameAnalyser_T
PRP_Frames_T
PRP_Trailer_T
PRP_Discard_Algorithm_T
PRP_Supervision_T
receive(data, length, lan_id)
process_rx(data, length, lan_id)
analyse_rx(data, length, lan_id)
normal_rx(data, length, lan_id)
Normal frame path
Supervision frame path
supervision_rx(data, length, lan_id)
At this point, supervision receiving handling is treated the same as normal_rx
normal_rx(data, length, lan_id)
get_trailer(data, length)
Check if RCT LAN_ID is equal received adapter LAN_ID, return KEEP if it is not
If transparent reception is FALSE
remove_trailer(data, length)
discard_algorithm(src_mac, seq_nr)
Return DROP if the received frame is a duplicate, return KEEP if it is no duplicate and add an entry to the hash list
Return here (KEEP or DROP, return value)

Figure 3 Reception of a frame

3.3.2 Normal frame transmitting

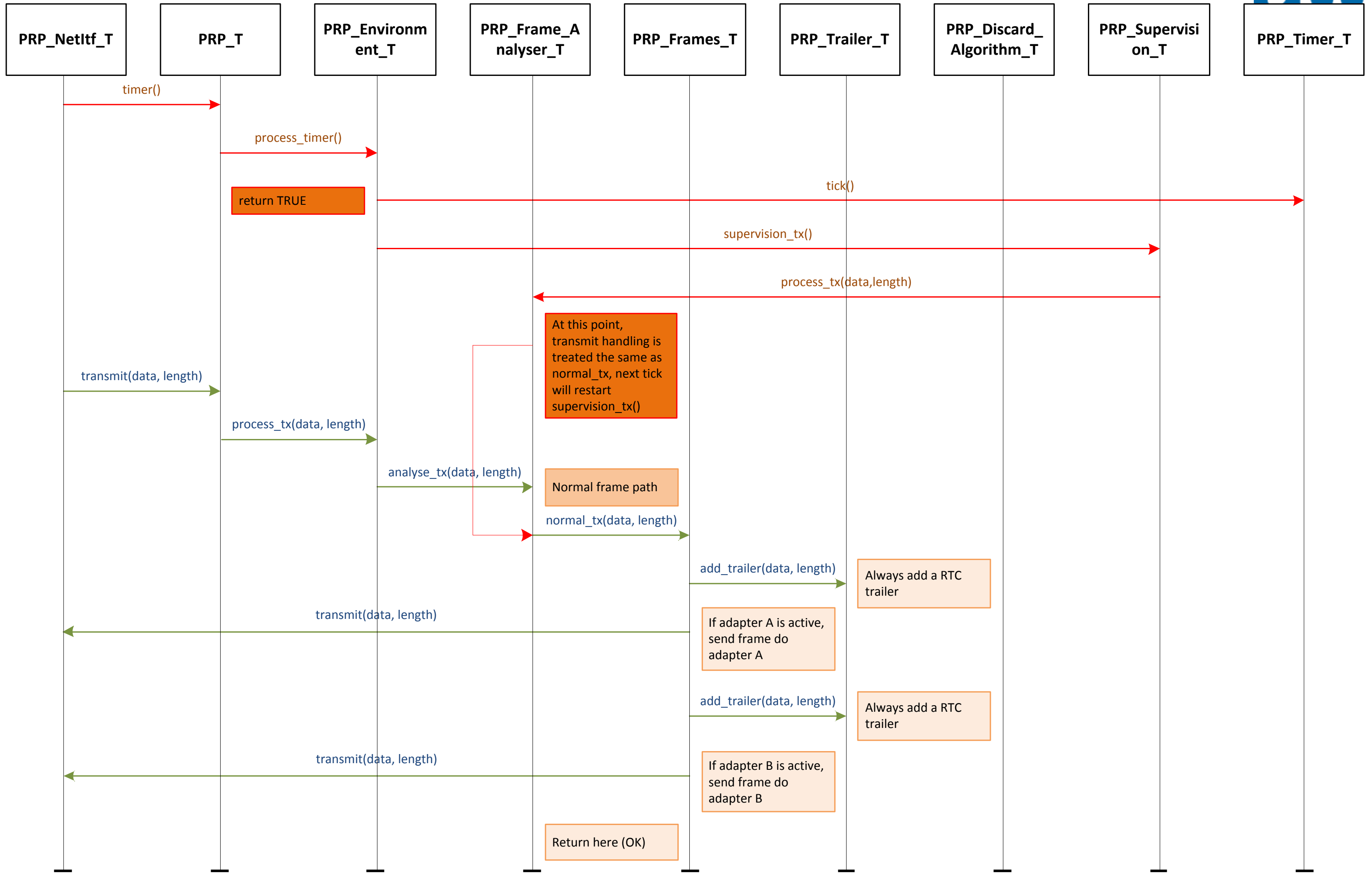

Figure 4 Transmission of a frame

Integration in Linux

3.4 Example Userspace integration in Linux

The PRP protocol engine receives the frames from the two real Ethernet interfaces, does the Duplicate elimination and forwards it to a virtual network interface which forwards the frames back to the operating systems protocol engine. Therefore all applications are talking with the virtual network device.

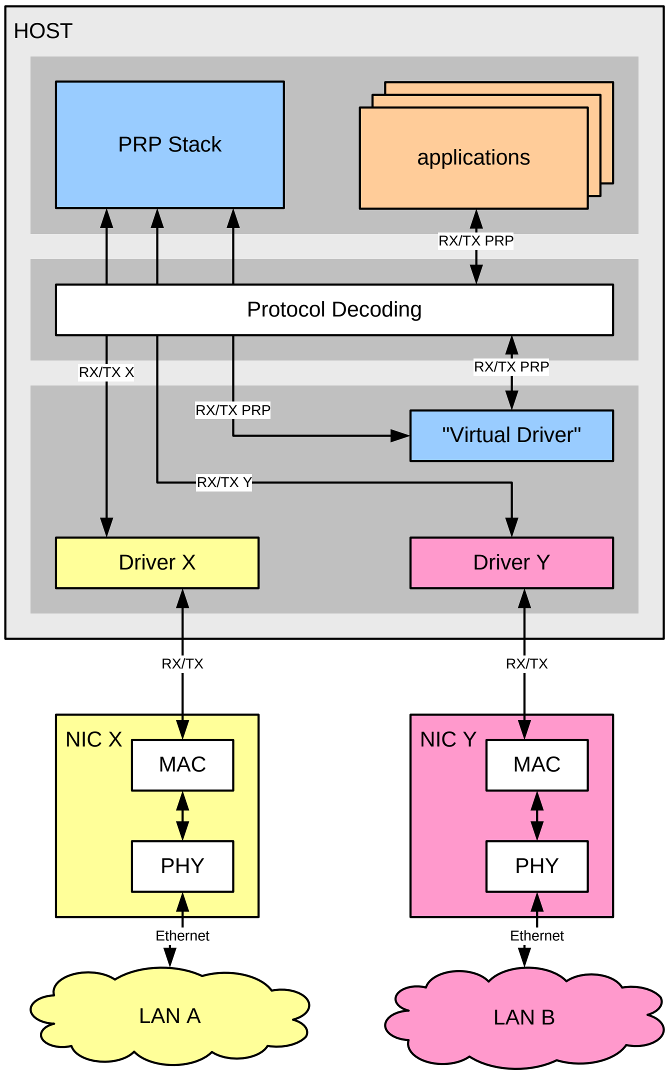

Figure 5 Linux Userspace integration

3.5 Compile and run application

The make file is located in the prp_pcap_tap_userspace directory. Change into this directory and build the PRP-1 SW-stack by calling make. If the application gets compiled successfully, the application prp_pcap_tap_userspace can be executed.

4 Graphic index