Low Level Design Specification

GPF-based TCP/IP Entity

Department:	Berlin Wireless Center		
Creation Date:	2004-08-19		
Last Modified:	2004-08-19 by Frank Stolte		
ID:	8462.701.04.01	Version:	001
Status:	Draft	ECCN:	Not Applicable

0 Document Control

0.1 Export Control Statement

Recipient agrees that it will not knowingly export or re-export, directly or indirectly, any product or technical data (as defined by the U.S, EU and other Export Administration Regulations) including software, or any controlled product restricted by other applicable national regulations, received from Disclosing party under this Agreement, or any direct product of such technology, to any destination to which such export or re-export is restricted or prohibited by U.S or other applicable laws, without obtaining prior authorisation from U.S. Department of Commerce and other competent Government authorities to the extent required by those laws. This provision shall survive termination or expiration of this Agreement.

According to our best knowledge of the state and end-use of this product or technology, and in compliance with the export control regulations of dual-use goods in force in the origin and exporting countries, this

technology is classified as given on the front page.

This product or technology may require export or re-export license for shipping it in compliance with certain countries regulations.

0.2 Document History

Date	Version	Status	Author
2004-08-19 Initial version.	001	Draft	Jürgen Nickelsen, Dirk Schmidt, Frank Stolte

0.3 References, Abbreviations, Terms

[C_4711.005]	G23-GPRS Protocol Stack Release 1.4.0, Product Specification, Condat AG, May 2002
[C_7010.801]	7010.801, References and Vocabulary, Condat AG
[C_8462.601]	8462.601, Socket API for GPF-based TCP/IP, Texas Instruments
[RIV201]	RNET Interface – Riviera NET API (TCP/IP), Version 0.8.2, Texas Instruments
[Stevens 1994]	TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 1994.
[Stevens 1995]	TCP/IP Illustrated, Volume 2: The Implementation, Addison-Wesley, 1995.
[Stevens 1998]	UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998.
[NexGenIP]	
[RNET API]	
[TCPIP SAP]	TCPIP SAP specification, Texas Instruments

ACI	Application Control Interface (includes the AT command interpreter)
API	Application Programming Interface
CSD	Circuit Switched Data
DCM	Data Connection Manager; the component of the protocol stack that manages GSM/GPRS data connection for use with TCP/IP.
DNS	Domain Name Service; the system to map Internet domain names to their IP addresses
GPF	Generic Protocol Framework; communication infrastructure and virtual system interface environment of the G23 protocol stack, developed at TI Berlin (former Condat)
IP	Internet Protocol; the layer 3 protocol of the Internet
TCP	Transmission Control Protocol; layer 4 protocol of the Internet protocol suite providing a connection-oriented, reliable byte stream service
TCP/IP	the Internet protocol suite; the name is derived from the layer 3 protocol IP and the most widely used layer 4 protocol TCP
UDP	User Datagram Protocol; layer 4 protocol of the Internet protocol suite providing a best-effort packet-oriented service
VSI	Virtual System Interface; GPF's abstraction of operating system services

Table of Contents

1 Summary

To provide TCP/IP services to GPF-based applications, the TI G23 GSM/GPRS protocol stack contains an entity that implements layer 3 (IP, ICMP) and the most common parts of layer 4 (UDP, TCP) of the Internet protocol suite. The core of this entity is the *NexGenIP* implementation from NexGen Software, France. The interface to the other entities of the protocol stack is based on GPF's communication primitives as described in the TCPIP, DCM, and DTI SAP documents. Parts of the current design stem from the history of the current implementation and may be cleaned up later.

1.1 Open Issues

Currently the following issues are still open and need further consideration:

- IPv6 transition
- Wireless profiled TCP
- Removal of RNET layer

A few areas in the scope of this document are implemented, but not yet described here:

- handling of payload data, including the DTI interface on the NexGenIP side, is incomplete
- General architecture of the embedded RNET integration of NexGenIP

1.2 Risks

These risks are currently identified:

- Multiple incoming connections may overload an application.

2 Introduction

2.1 GPF-based TCP/IP Subsystem

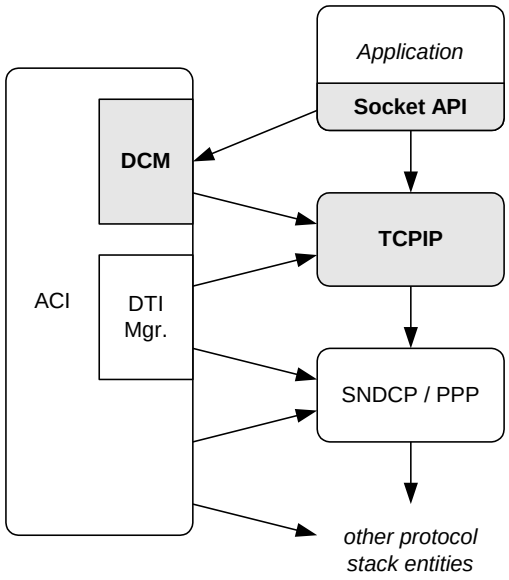


Figure 2-1: TCP/IP Subsystem Components

The TCP/IP subsystem in the GPF protocol stack framework consists of three parts:

1. The Socket API library, running in the application task context, which allows applications to open an according bearer connection (via DCM) and to access the services of the TCPIP entity,
2. the Data Connection Manager (DCM), which opens or closes a bearer connection over the wireless network for use with TCP/IP on request of the application.
3. the TCPIP entity, which contains the implementation of the TCP/IP protocols (IP, TCP, UDP, etc.),

The lower-layer neighbor entity of TCPIP is SNDCP for a GPRS bearer connection or PPP for a CSD (GSM) bearer connection. DCM will do the internal connection of these specific entities automatically, according to the required bearer, with the help of DTI.

The GPF-based TCP/IP implementation runs as an own GPF task. Payload data exchange with the lower layer uses the Data Transmission Interface (DTI). The TCP/IP entity does no GSM or GPRS signaling itself, which is handled by the DCM .

The Socket API library uses the TCPIP SAP for signaling (control plane) and payload data (user plane). It is specified in [C_8462.601].

TCP/IP needs an underlying data (bearer) connection over the mobile network to send and receive IP packets. This can be a GSM circuit-switched data call (CSD) or a GPRS PDP context. Management of the bearer connection is done by a component called *Data Connection Manager* (DCM).

2.2 History of the Implementation

The current implementation of the TCP/IP entity is derived from the integration of NexGenIP into the Riviera application framework. This integration was done by NexGen Software and the SSA group of TI Nice, called RNET (for Riviera Networking).

Instead of using Riviera as an application platform, customers demanded TCP/IP services for GPF-based applications. Due to the timing constraints of the projectit was decided to port more or less the whole RNET entity from Riviera to GPF, including its Riviera integration, adaptation layers, and API.

2.3 Scope of this document

This document describes the design of the existing TCP/IP entity implementation in the Texas Instruments G23 GSM/GPRS protocol stack; to be more precise, it describes how the entity is embedded into the protocol stack and how the glue layers that implement this embedding interact with the RNET API layer and the core TCP/IP. It does not describe the design of the RNET layer or of the core TCP/IP implementation. For these topics the reader is re ferred to [NexGenIP] and [RNET API] etc.

The document is not intended to replace text books like, for instance, [Stevens 1998] or [Stevens 1994]. [Stevens 1995] also provides excellent reference material for further work on this entity.

2.4 Terminology

The abbreviation “TCP/IP”, containing the slash character “/”, refers to the Internet protocol suite; in this document it is also used in compound terms related to the Internet protocol suite as in “TCP/IP entity” or “TCP/IP application”, meaning the GPF entity containing the implementation of the Internet protocol suite or an application using the Internet protocols, respectively.

When the term “TCPIP” is used, without the slash character, it denotes something that bears this name also in formal documents like specifications or program code. It is used either as a shorthand for the TCP/IP entity or for the SAP of the TCP/IP entity. The context should make any further distinction unnecessary. For example, “TCPIP primitives” refers to primitives defined in the TCPIP SAP.

3 Architecture of the TCP/IP Entity

3.1 Overview: Interfaces and Layers

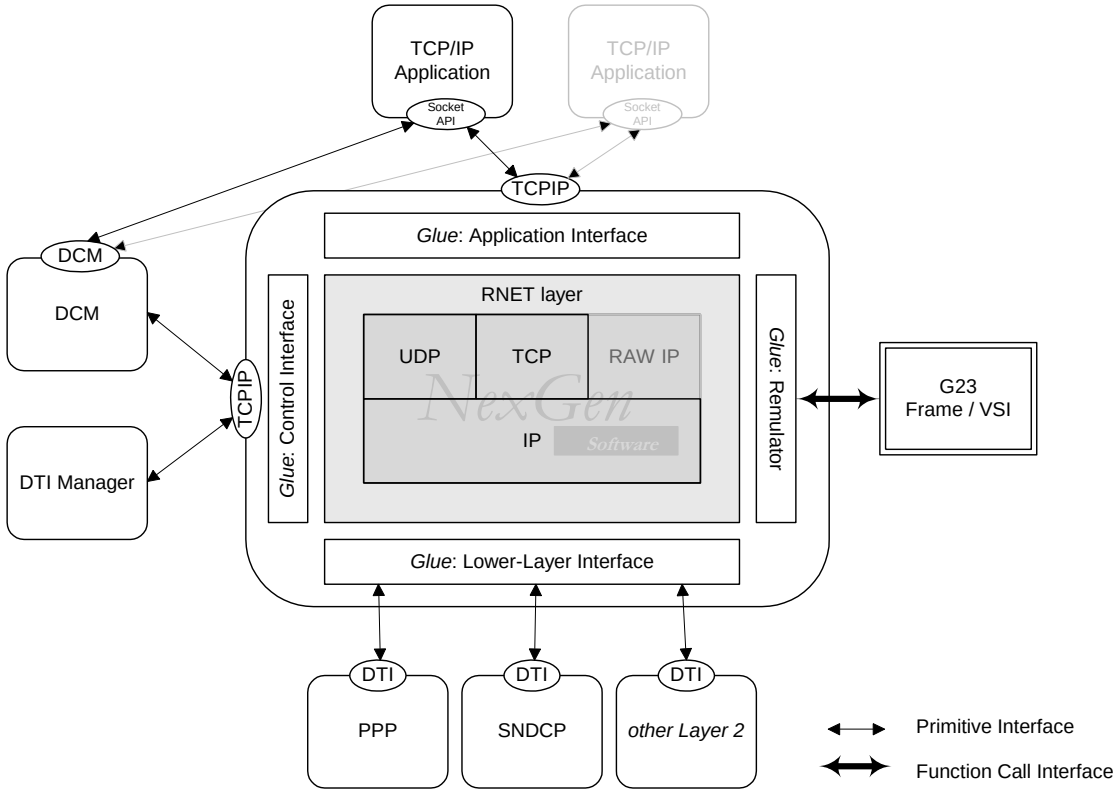


Figure 3-1: Architecture Overview of the TCP/IP entity

The core of TCPIP consists of the NexGenIP implementation. Around the core is, for historical reasons (see section 2.2), the Riviera adaptation layer called RNET. Around that, there are four glue layers to adapt RNET to the interfaces provided by the protocol stack:

1. the interface to applications, which is one part of the TCPIP SAP; implemented in `tcpip_api_layer.c`
2. the control interface, the other part of the TCPIP SAP; also implemented in `tcpip_api_layer.c`
3. the lower-layer interface, using the DTI SAP; implemented in `tcpip_dti.c`
4. the interface to system services, using the "Riviera Emulator" (Remulator); the Remulator is not described in this document

3.2 TCPIP SAP

The control and application interface is specified in [C_8462.601] and [TCPIP SAP]. Most of these primitives are used in a request/confirmation kind of interaction, and some are indications sent from TCPIP to the application, with a response from the application in one case (TCPIP_DATA_IND/RES).

For every primitive in the TCPIP SAP sent to TCPIP there is a handler function in `tcpip_api_layer.c`; the name of the handler function is the same as the name of the primitive, only in lower case. Likewise, for every TCPIP primitive to be sent there is a sender function with the name of the primitive in lower case (except TCPIP_INTERNAL_IND, which is sent by and received from TCPIP).

Request/confirmation interactions pattern fall into two categories:

(a) Some requests produce a *direct result*, possibly obtained as the return value of a function call of the RNET interface or the TCP/IP core (i. e. NexGenIP).

(b) Others may return a direct result in some error case, but in the successful case, the real result is obtained by some interaction of the TCP/IP core with the network, which takes some time. Because we cannot block the entity for this time, the result is delivered by RNET as an event via a *callback* function.

For case (a), the request handler function can directly call the confirmation sender function. For case (b), the callback function must finally call the confirmation sender function.

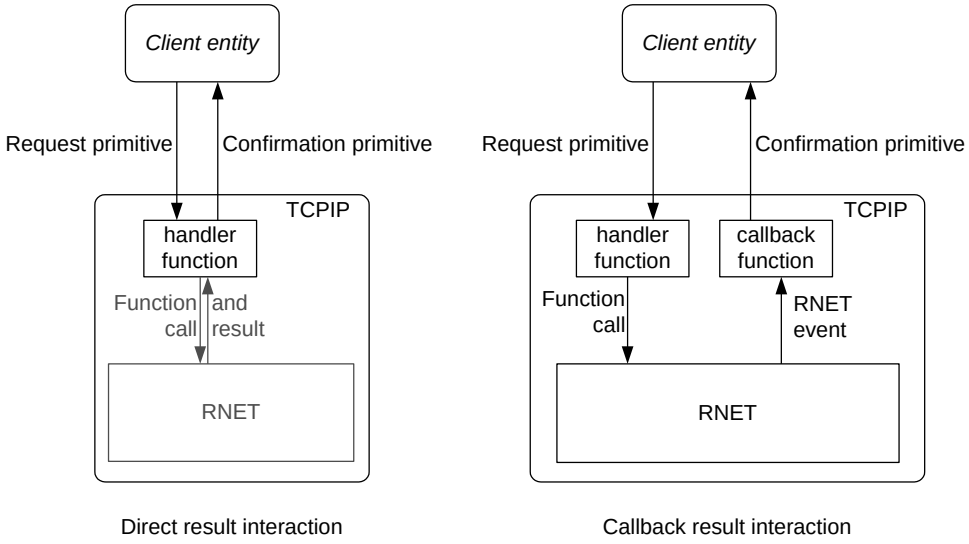


Figure 3-2: Request/Confirmation Interaction Patterns

The following table shows what pattern of interaction is used by the TCPIP requests.

Table 3-1: TCPIP Requests and Corresponding Interaction Patterns

Direct result interaction	*Callback result interaction*
TCPIP_INITIALIZE_REQ TCPIP_SHUTDOWN_REQ TCPIP_IFCONFIG_REQ TCPIP_DTI_REQ TCPIP_CREATE_REQ TCPIP_CLOSE_REQ TCPIP_BIND_REQ TCPIP_LISTEN_REQ TCPIP_SOCKNAME_REQ TCPIP_PEERNAME_REQ TCPIP_MTU_SIZE_REQ	TCPIP_CONNECT_REQ TCPIP_DATA_REQ TCPIP_HOSTINFO_REQ

3.2.1 Handling of RNET callbacks

The application interface has a single callback function `tcpip_rnet_callback()` for all socket-related events. According to the event type it dispatches to the corresponding event handler function, which in turn sends a primitive to the application, if necessary.

For DNS lookups, another callback function, `tcpip_hostinfo_callback()`, is used, which directly sends the confirmation primitive to the application.

3.2.2 Data Structures

To associate the socket identifiers in the TCPIP primitives with the RNET socket descriptors and for administering the interaction with RNET and the application, a few data structures are needed in the application interface.

In the primitive interface, we need the socket identifiers as small integers in order to have deterministic socket IDs for the testing system. (With pointers, the actual value would be different after each change in the software.) In the RNET interface, we have to use RNET's socket descriptors, which are actually pointers. Finally for TCPIP's application interface's own housekeeping, we need a data structure of socket parameters as well, called "`sockpar`", which is accessed through pointers.

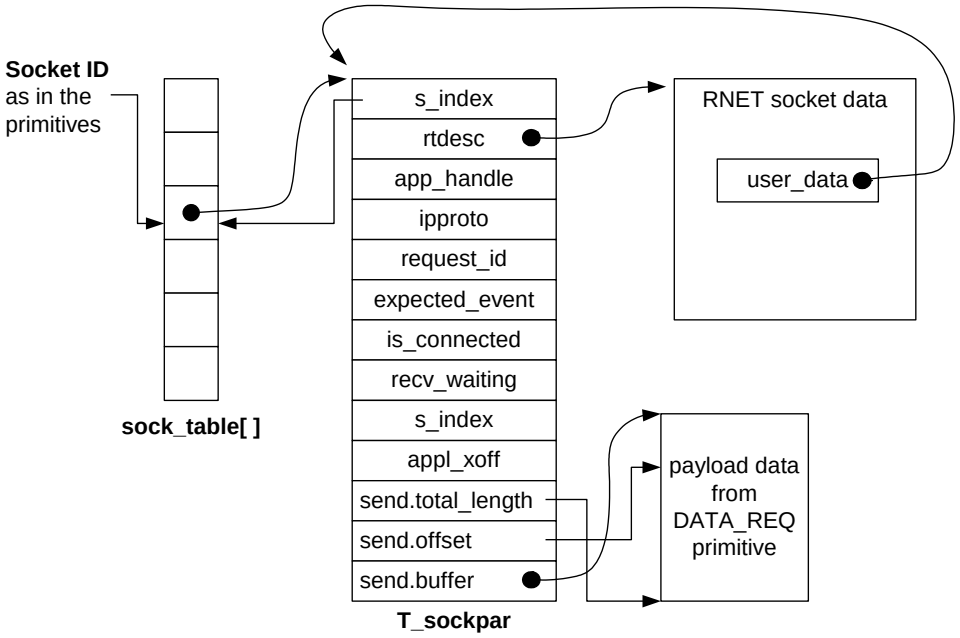


Figure 3-3: Per-Socket Data Structures in Application Interface Layer

This way, we can address a socket through (a) the socket ID in the primitive, (b) the RNET socket descriptor, and (c) the `sockpar` pointer. To get from one to the other, the following methods are used:

Table 3-2: Conversion Between Socket Representations

	From		*To*
Macro `SOCKPAR_GET()`	RNET desc.	→	`sockpar` pointer
Macro `SOCK_S_INDEX()`	RNET desc	→	socket ID
Macro `SOCK_RT_DESC()`	socket ID	→	RNET desc.
Array `sock_table[]`	socket ID	→	`sockpar` pointer
pointer`->s_index`	`sockpar` pointer	→	socket ID
pointer`->rtdesc`	`sockpar` pointer	→	RNET desc.

The fields of the socket parameter structure are used as follows:

Table 3-3: Fields of the Per-Socket Data Structure

`s_index`	Index of the socket in the `sock_table[]`, also used in the TCPIP primitives

`rtdesc`	RNET descriptor of the socket
`app_handle`	Communication handle of the application entity, used to send primitives to the entity
`ipproto`	IP transport protocol number used with the socket (currently only for UDP and TCP), used where different the transport protocols need to be treated differently
`request_id`	Used to associate request and confirmation primitives for `TCPIP_CREATE_REQ/CNF` and `TCPIP_HOSTINFO_REQ/CNF`
`expected_event`	Marks the next event type that is expected for the socket; necessary for error handling because the RNET error indication contains no information *which* request caused the error
`is_connected`	TRUE iff `rnet_connect()` has been called for this socket; used to detect if the application tries to send data on a non-connected socket (NexGenIP apparently does not catch this error)
`recv_waiting`	TRUE iff received data may be waiting to be picked up
`appl_xoff`	TRUE iff flow control to the application is in "xoff" status
`send.total_length`	Total length of send data buffer
`send.offset`	Next offset from which payload data shall be sent from the buffer
`send.buffer`	Pointer to send data buffer

3.2.3 Application Interface Primitives

Table 3-4: TCPIP Application Primitives

Primitives	*Description*	*BSD Socket API equivalent*
TCPIP_CREATE_REQ TCPIP_CREATE_CNF	Create a socket and return result	socket()
TCPIP_CLOSE_REQ TCPIP_CLOSE_CNF	Close a socket and return result	close()
TCPIP_BIND_REQ TCPIP_BIND_CNF	Bind socket to a port	bind()
TCPIP_LISTEN_REQ TCPIP_LISTEN_CNF	Listen for incoming connections	listen(), accept()
TCPIP_CONNECT_REQ TCPIP_CONNECT_CNF	Connect to a remote peer	connect()
TCPIP_DATA_REQ TCPIP_DATA_CNF	Send data uplink and acknowledge	send(), sendto(), sendmsg(), write()
TCPIP_DATA_IND TCPIP_DATA_RES	Send data downlink and acknowledge	recv(), recvfrom(), recvmsg(), read()
TCPIP_SOCKNAME_REQ TCPIP_SOCKNAME_CNF	Get local address of socket	getsockname()
TCPIP_PEERNAME_REQ TCPIP_PEERNAME_CNF	Get remote address of connection	getpeername()
TCPIP_HOSTINFO_REQ TCPIP_HOSTINFO_CNF	DNS lookup	gethostbyname(), gethostbyaddr()
TCPIP_MTU_SIZE_REQ TCPIP_MTU_SIZE_CNF	Get MTU size of interface	getsockopt() (?)

TCPIP_CONNECT_IND	Indicate incoming TCP connection	select(), accept()
TCPIP_CONN_CLOSED_IND	Indicate connection close by peer	select(), read()
TCPIP_ERROR_IND	Indicate error on connection	select(), *return values of functions*

3.2.4 Control Interface Primitives

The interface to the DCM is specified in [TCPIP SAP]. The following table briefly summarizes the purpose of the application-related TCPIP primitives.

Table 3-5: TCPIP Control Primitives

Primitives	*Description*
TCPIP_INITIALIZE_REQ TCPIP_INITIALIZE_CNF	Make TCPIP allocate its private memory and initialize its data structures.
TCPIP_SHUTDOWN_REQ TCPIP_SHUTDOWN_CNF	Make the entity deallocate its private memory.
TCPIP_IFCONFIG_REQ TCPIP_IFCONFIG_CNF	Notify TCPIP about a bearer connection (network interface in TCP/IP speak) coming up or going down and related parameters.
TCPIP_DTI_REQ TCPIP_DTI_CNF	Make TCPIP establish or tear down a DTI connection to the appropriate neighbor entity for a bearer connection.

The control interface does not hold own data structures.

3.3 Handling of Payload Data

{ *tcpip_dti.c contains stubs for SAT class E over TCP/IP, which will probably not be used anymore; due to lack of documentation lower-layer interface designed mainly through reverse-engineering of ATP interface code in rnet_rt_atp.c* }

3.3.1 To Upper Layer

{ *send a TCPIP_DATA_IND to upper layer only after receiving a TCPIP_DATA_RES for that socket, except for the first one (flow control); RNET sends RNET_RECV_IND when incoming data is available; incoming data can be fetched from RNET until rnet_recv() returns zero length; only then RNET sends a new RNET_RECV_IND for new incoming data* }

3.3.2 From Upper Layer

{ *application may send TCPIP_DATA_REQ only after receiving a TCPIP_DATA_CNF for that socket, except for the first one; tcpip_try_send_data() sends data buffer using rnet_send() partially, if necessary, and sends a TCPIP_DATA_CNF only after the full buffer has been sent to RNET* }

3.3.3 To Lower Layer

{ *rnet_rt_dti.c modeled after rnet_rt_atp.c; tcpip_dtiOutput() called by NexGenIP to send data to LL via DTI; flow control state maintained in tcpip_data->ll[0].flowstat_ul (ul meaning "uplink"); DTI event handling in tcpip_dti.c* }

3.3.4 From Lower Layer

{ *tcpip_ll_dti_data_ind() adapts incoming data for ngIfGenInput()* }

3.3.5 Loopback

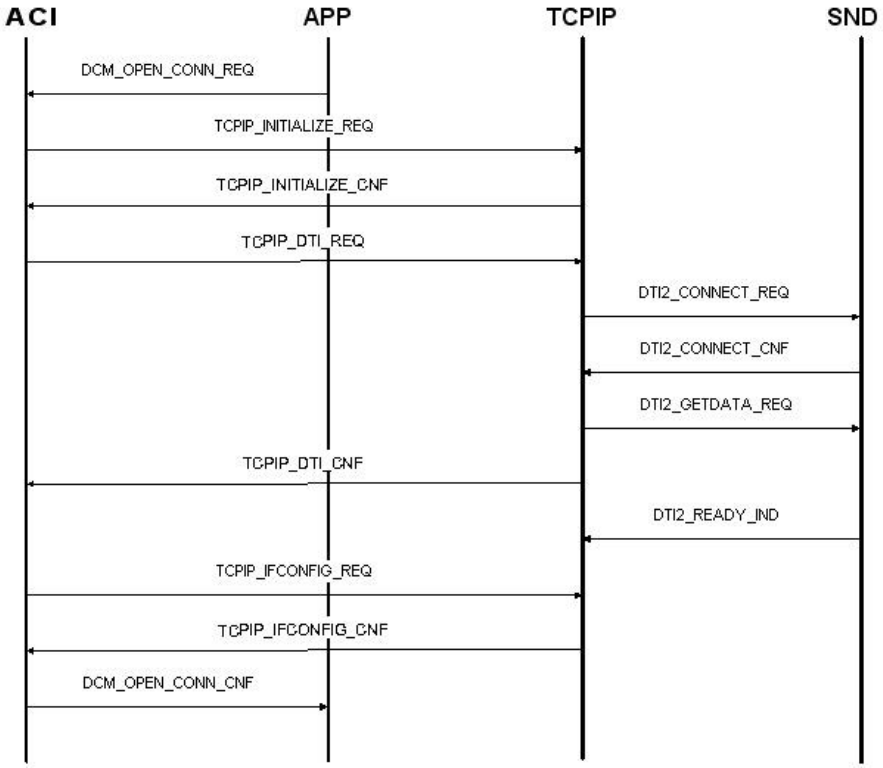
To enable TCP/IP communication between different applications attached to the TCP/IP entity, a loopback interface can be configured in RNET. It has the standard loopback IP address 127.0.0.1. To decouple the sending and receiving function call chains, IP packets over the loopback interface are passed via a TCPIP_INTERNAL_IND primitive sent by the entity to itself.

3.4 System Services Interface (Remulator)

{ *adapts several Riviera frame functions to GPF VSI services; anything about that? does not really relate to other things in TCPIP* }

4 Message Sequence Charts

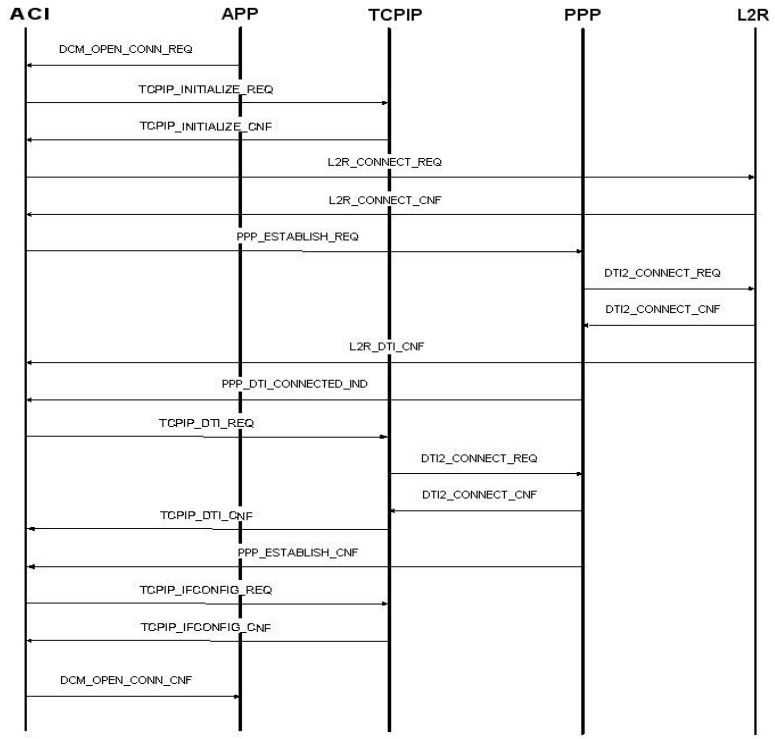
4.1 TCPIP Initialisation (GPRS as bearer)



Description:

1. DCM_OPEN_CONN_REQ will be send on the execution of the Socket API library function sock_open_bearer
2. TCPIP_INITIALIZE_REQ requests the initialisation of the TCPIP entity
3. TCPIP_INITIALIZE_CNF confirms the initialisation of the TCPIP entity
4. TCPIP_DTI_REQ triggers TCPIP entity to get connected with SNDCP since we will use GPRS as a bearer
5. DTI2_CONNECT_REQ TCPIP requests the connection with SNDCP
6. DTI2_CONNECT_CNF SNDCP confirms successful connection with TCPIP
7. DTI2_GETDATA_REQ used for flow control and indicates that TCPIP is ready to get the next DTI2_DATA_IND primitive.
8. TCPIP_DTI_CNF confirms the successful connection of SNDCP and TCPIP
9. DTI2_READY_IND is used for flow control and indicates that TCPIP is ready to get the next DTI2_DATA_REQ primitive
10. TCPIP_IFCONFIG_REQ informs TCPIP about an successfully created network interface(PDP context activation)
11. TCPIP_IFCONFIG_CNF response to previous send request
12. DCM_OPEN_CONN_CNF now informs the application about a successful set up of a GPRS network connection (GPRS) which now can be used by TCPIP as a bearer for sending/receiving data

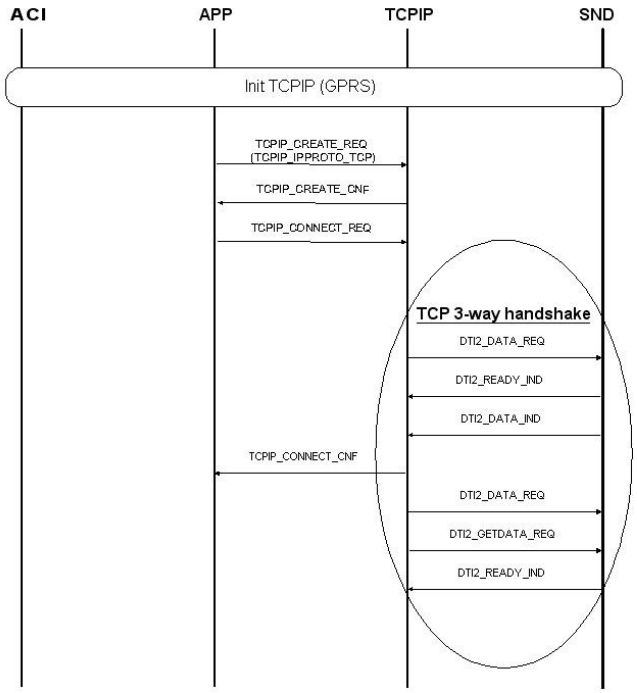
4.2 TCPIP Initialisation (CSD as bearer)



Description:

1. DCM_OPEN_CONN_REQ will be send on the execution of the Socket API library function sock_open_bearer
2. TCPIP_INITIALIZE_REQ requests the initialisation of the TCPIP entity
3. TCPIP_INITIALIZE_CNF confirms the initialisation of the TCPIP entity
4. L2R_CONNECT_REQ is used to connect the L2R entity to its peer entity
5. L2R_CONNECT_CNF acknowledges the previous L2R_CONNECT_REQ
6. PPP_ESTABLISH_REQ is used to request establishment of PPP connection
7. DTI2_CONNECT_REQ PPP requests the connection with L2R
8. DTI2_CONNECT_CNF L2R confirms successful connection with PPP
9. L2R_DTI_CNF acknowledges L2R - > PPP connection
10. PPP_DTI_CONNECTED first DTI channel is connected
11. TCPIP_DTI_REQ triggers TCPIP entity to get connected with PPP since we will use CSD as a bearer
12. DTI2_CONNECT_REQ TCPIP requests the connection with PPP
13. DTI2_CONNECT_CNF PPP confirms successful connection with TCPIP
14. TCPIP_DTI_CNF confirms the successful connection of PPP and TCPIP
15. PPP_ESTABLISH_CNF informs about successful establishment of a PPP link
16. TCPIP_IFCONFIG_REQ informs TCPIP about an successfully created network interface(PDP context activation)
17. TCPIP_IFCONFIG_CNF response to previous send request
18. DCM_OPEN_CONN_CNF now informs the application about a successful set up of a GPRS network connection (GPRS) which now can be used by TCPIP as a bearer for sending/receiving data

4.3 Open TCP socket (GPRS)

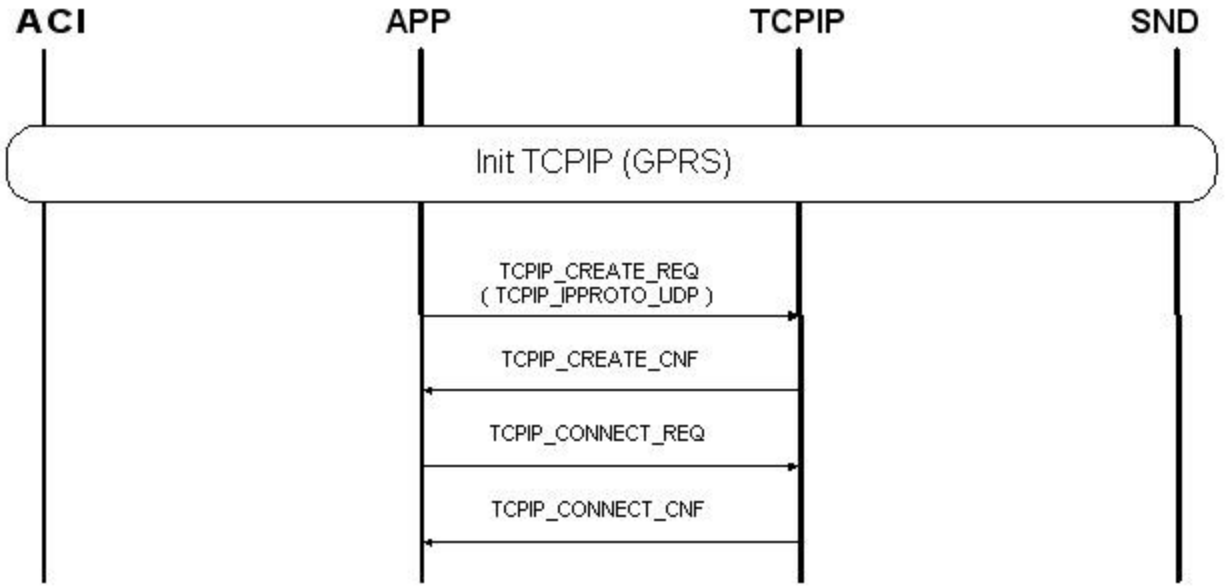


1. TCPIP_CREATE_REQ requests a newly created TCP socket from TCPIP (TCPIP_IPPROTO_TCP indicates the type of socket)
2. TCPIP_CREATE_CNF returns a newly created TCP socket
3. TCPIP_CONNECT_REQ requests TCPIP to connect a socket to a remote peer.
4. TCP 3-way handshake ensures that both sides are ready to transmit data, and that *both* ends know that the other end is ready *before* transmission actually starts.
5. TCPIP_CONNECT_CNF informs about succesfull connection

Note:

For openeing a TCP socket and using CSD as bearer SND will be PPP instead and the initialisation will of course be the CSD specific one.

4.4 Open UDP socket (GPRS)

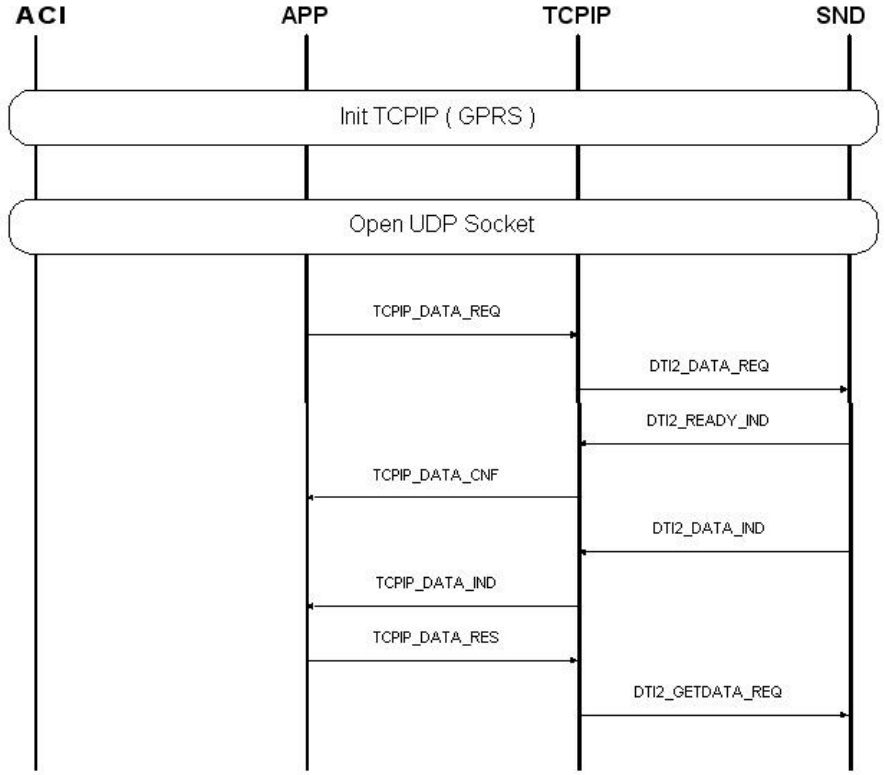


1. TCPIP_CREATE_REQ requests a newly created UDP socket from TCPIP (TCPIP_IPPROTO_UDP indicates the type of socket)
2. TCPIP_CREATE_CNF returns a newly created UDP socket
3. TCPIP_CONNECT_REQ requests TCPIP to connect a socket to a remote peer.
4. TCPIP_CONNECT_CNF informs about succesfull connection

Note:

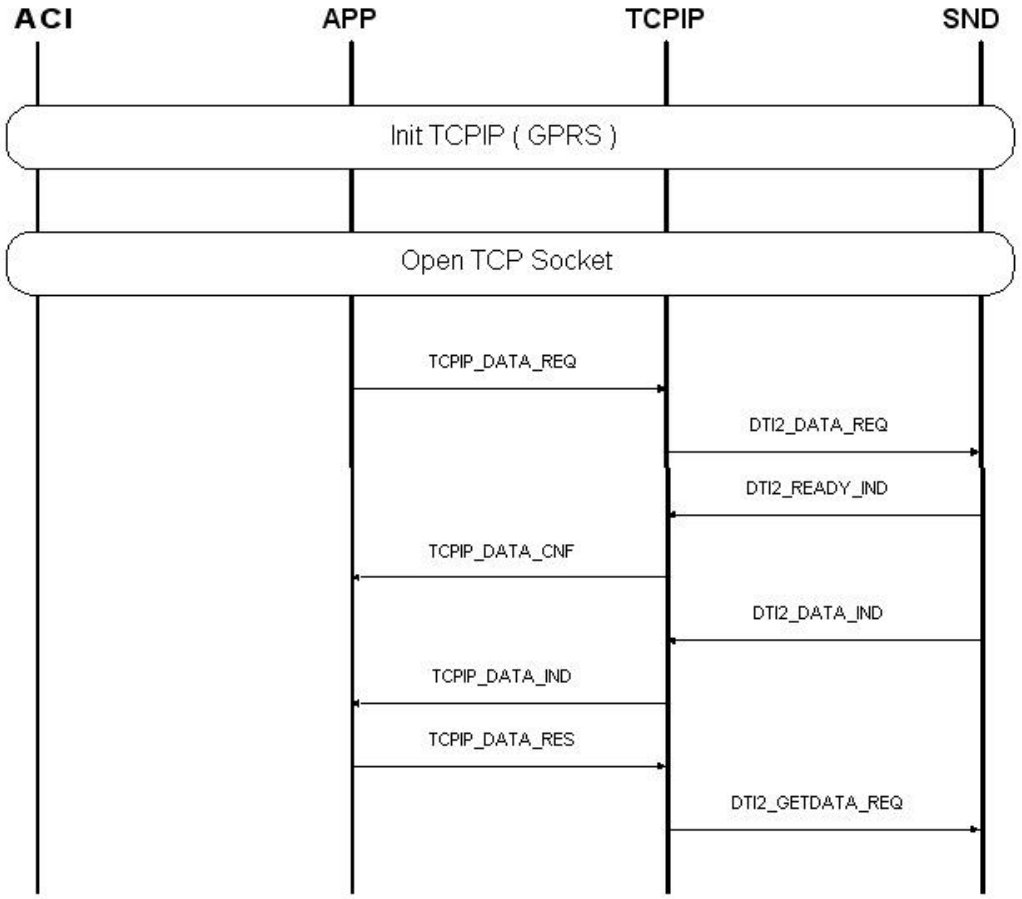
For openeing a UDP socket and using CSD as bearer SND will be PPP instead and the initialisation will of course be the CSD specific one.

4.5 UDP client



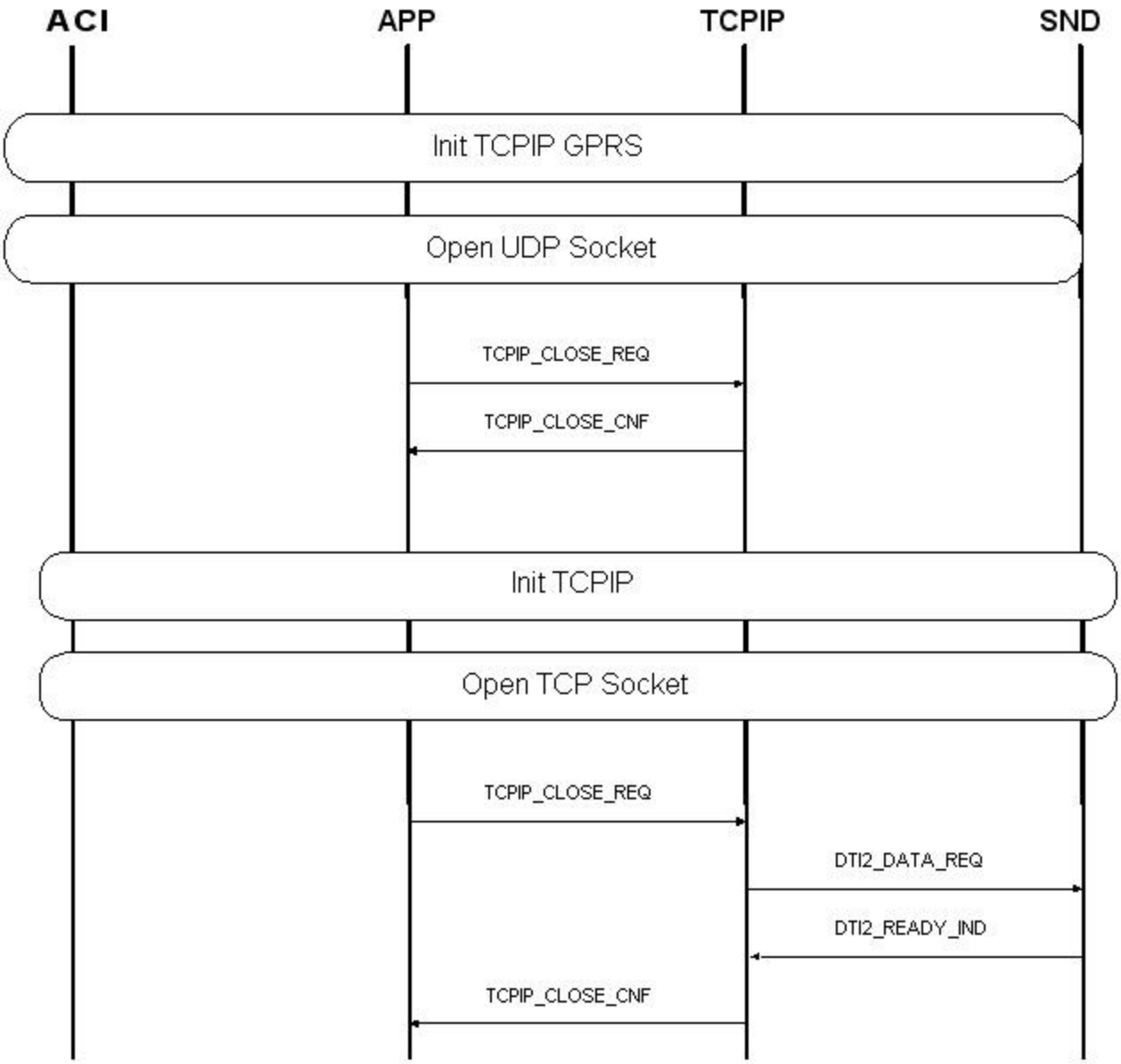
1. TCPIP_DATA_REQ The application requests TCPIP to send payload data on a socket
2. DTI2_DATA_REQ is used to pass on data to the SNDCP entity
3. DTI2_READY_IND indicates that SNDCP is ready to get the next DTI2_DATA_REQ primitive
4. TCPIP_DATA_CNF acknowledges the TCPIP_DATA_REQ and signals to the application how much data the application is allowed to send (window size). The application may now send data on this socket again; it may send as much data as the “window” parameter specifies, but at least one
5. DTI2_DATA_IND is used to pass on data to the neighbor entity
6. TCPIP_DATA_IND sends payload data to the application for the specified socket
7. TCPIP_DATA_RES acknowledges a TCPIP_DATA_IND and signals to TCPIP how much data TCPIP is allowed to send (window size). TCPIP may now send data on this socket again; it may send as much data as the “window” parameter specifies, but at least one
8. DTI2_GETDATA_REQ is used for flow control and indicates that TCPIP is ready to get the next DTI2_DATA_IND primitive

4.6 TCP client

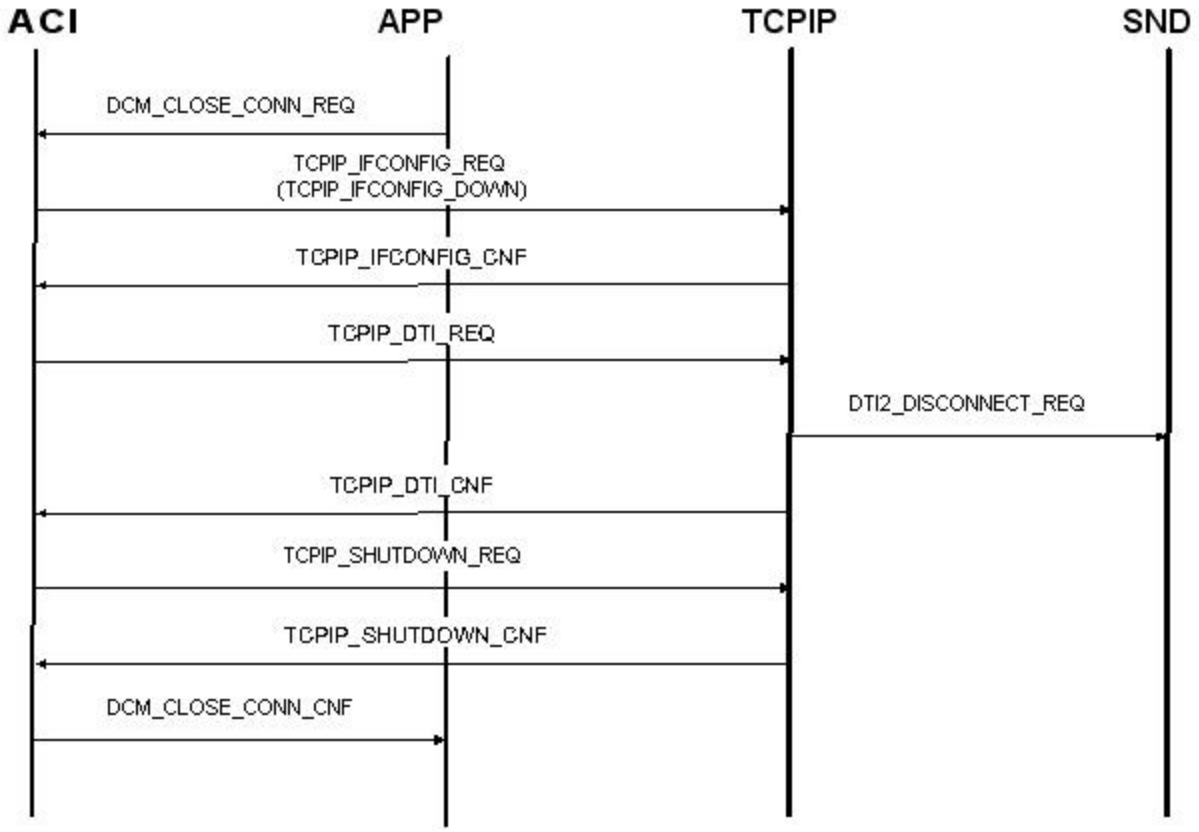


1. TCPIP_DATA_REQ The application requests TCPIP to send payload data on a socket
2. DTI2_DATA_REQ is used to pass on data to the SNDCP entity
3. DTI2_READY_IND indicates that SNDCP is ready to get the next DTI2_DATA_REQ primitive
4. TCPIP_DATA_CNF acknowledges the TCPIP_DATA_REQ and signals to the application how much data the application is allowed to send (window size). The application may now send data on this socket again; it may send as much data as the "window" parameter specifies, but at least one
5. DTI2_DATA_IND is used to pass on data to the neighbor entity
6. TCPIP_DATA_IND sends payload data to the application for the specified socket
7. TCPIP_DATA_RES acknowledges a TCPIP_DATA_IND and signals to TCPIP how much data TCPIP is allowed to send (window size). TCPIP may now send data on this socket again; it may send as much data as the "window" parameter specifies, but at least one
8. DTI2_GETDATA_REQ is used for flow control and indicates that TCPIP is ready to get the next DTI2_DATA_IND primitive

4.7 Close UDP/TCP Socket



4.8 Shutdown TCP (GPRS)



1. DCM_CLOSE_CONN_REQ starts the disconnection of the GPRS connection
2. TCPIP_IFCONFIG_REQ informs TCPIP that interface is going down(parameter TCPIP_IFCONFIG_DOWN)
3. TCPIP_DTI_REQ initiates the disconnection of the SNDCP and the TCPIP entity
4. DTI2_DISCONNECT_REQ closes SNDCP dti connection to TCPIP
5. TCPIP_DTI_CNF confirms the disconnection of SNDCP and TCPIP
6. TCPIP_SHUTDOWN_REQUEST requests TCPIP to shutdown
7. TCPIP_SHUTDOWN_CNF confirms the shutdown of TCPIP
8. DCM_CLOSE_CONN_CNF confirms the closing of the GPRS connection

5 Appendices

5.1 List of Figures

5.2 List of Tables