INTERNATIONAL FORECOURT STANDARD FORUM

STANDARD FORECOURT PROTOCOL

PART II

COMMUNICATION SPECIFICATION

OVER TCP/IP

1.02 - JUNE 2004

This document was written by the IFSF - Working Group:

Name	Company
Barry McGugan	Marconi Commerce Systems
Steve Cramp	Marconi Commerce Systems
Peter Maeers	MPS
Jaroslav Dvorak	Beta Control Ltd.

The latest revision of this document can be downloaded form the Internet

address: www.ifsf.org

Any queries regarding this document should be addressed to: secretary@ifsf.org

Document Contents

[1]	IFSF STANDARD FORECOURT PROTOCOL PART II – COMMUNICATION SPECIFICATION
[2]	IFSF STANDARD FORECOURT PROTOCOL PART III.I – DISPENSER APPLICATION
[3]	Comer, Douglas E.: Internetworking with TCP/IP – Principles, Protocols, and Architectures, Volume 1, Fourth Edition, 2000, 1995 Prentice Hall
[4]	Unix Manual Pages
[5]	**M**icro**S**oft **D**eveloper **N**etwork (MSDN) Helps

1 Record of Changes

Date	Version number	Modifications
March 2001	1.00	First draft release
August 2001	1.00	Formal release
February 2002	1.01	Appendix 2 – changes in connection with the removal of block cutting over TCP/IP
June 2004	1.02	Glossary – Added definition for the 'Well known' IFSF Heartbeat Port.

2 Glossary

ARP - Address Resolution Protocol. An Internet protocol that enables the resolution of a logical address (IP) to a physical address (MAC) on a LAN.

BOOTP - Bootstrap Protocol. An Internet protocol that enables remote static configuration of hosts on an IP network.

Client - A process that issues a connection request to a service either on the same computer or a remote computer.

DHCP - Dynamic Host Configuration Protocol. An Internet protocol that enables dynamic configuration of hosts on an IP network.

DNS - Domain Name System. A hierarchical system for identifying hosts on a LAN, whether public or private. It provides for mapping of an IP address to a friendly host name, resolving of host names to IP addresses so that communications can be established with the remote host, and a distributed mechanism for storing and maintaining list of names and IP addresses.

ICMP - Internet Control Message Protocol. An internet layer protocol that is used to build and maintain routing tables, error reporting, control messages, and adjusting flow rates.

Internet - The name given to the interconnection of many isolated networks into a virtual single network.

IP - Internet Protocol. The main protocol used in internetworking to route a message from one computer to another. The Internet Protocol is located in the internet layer of the IP stack and does not guarantee reliable delivery of messages.

IP address - A logical address of a physical device. Version 4 of TCP/IP, called IPV4, uses four hexadecimal bytes written in dotted decimal notation, to specify the address.

IP Stack - A reference to the layering of TCP/IP. TCP/IP consist of the network layer at the bottom of the stack, then the internet layer, then the transport layer, and finally the application layer.

MAC Address - The physical address of a device on an internet. It is also referred to as an Ethernet address, hardware address, or PHY address.

Port - A logical address of a service/protocol that is available on a particular computer.

TCP - Transmission Control Protocol. One of the two main protocols used in the transport layer of the IP stack. TCP is a connection oriented protocol that guarantees delivery of data.

TCP/IP - The generic name given to the suite of services and applications that are used for communicating over a local LAN or the Internet. TCP is the better known transport protocol and IP is the better known internet layer protocol.

TFTP - Trivial File Transfer Protocol. An application level protocol that is used to transfer files using UDP. It is typically used to download an image to a diskless remote host during the bootstrap process.

UDP - User Datagram Protocol. One of the two main protocols used in the transport layer of the IP stack. UDP is a connectionless oriented protocol that does not guarantee delivery of data.

Service - A process that accepts connections from other processes, typically called client processes, either on the same computer or a remote computer.

Socket - An access mechanism or descriptor that provides an endpoint for communication.

Socket Address - The combination of the IP address, protocol (TCP or UDP) and port number on a computer that defines the complete and unique address of a socket on a computer.

'Well known' IFSF heartbeat port - The UDP port to be used by all IFSF compliant devices having been assigned by the Internet Assigned Numbers Authority (IANA) as '3486'.

WINS - Windows Internet Name Service. A Microsoft Windows service that dynamically registers NetBIOS names on a Windows network and provides of resolution of names to IP addresses.

3 Introduction

This document describes the transport of IFSF application messages using the TCP/IP protocol suite. Detail on the IFSF messages is described in the IFSF STANDARD FORECOURT PROTOCOL PART II COMMUNICATION SPECIFICATION.

4 Guidelines for Implementation

4.1 Block Cutting

With TCP/IP there is no need for the IFSF application to perform block cutting. Although TCP/IP can transport any size message, it is recommended that single message sizes for the dispenser and other embedded applications be restricted to a maximum of 228 bytes. This reduces the buffering requirements for such applications

4.2 Security

As with any networking environment, security measures should be implemented in line with the results of risk assessment for a given installation. Details are dependant on the network installation and hence our outside the scope of this document. However the following items should be considered.

4.2.1 Access

Network access should be managed, clearly identifying sources, and any associated risks with these sources. Suitable access controls, such as passwords, dial-back etc, need to be in place to ensure network security.

4.2.2 Firewall

Any network accessible from unsecured sources (i.e. Internet) should provide adequate access protection using for example firewalls.

4.2.3 Authentication

Where sensitive network messages are routed over unsecured connections, an authentication mechanism should be used. This ensures that the end points of the connection can guarantee the source of the message is genuine.

4.3 IP Implementation

It is recommended that an IP stack be selected that does not buffer small messages. If this were to occur, it could delay message sending.

It is not recommended the application change the IP quality of service flags etc such as Service Type. These may not be managed in the same way by different network routers/IP stacks. The quality of service offered by TCP/IP will easily be equal to that supported by LON.

All IP implementations must meet applicable RFC's

As IP is a streaming protocol, it may not be immediately obvious where one IFSF message ends and another begins. It will be the responsibility of the implementation to detect the beginning and end of IFSF messages and correctly delivery them to the IFSF application. It is not permissable to add any extra information to the IFSF message to help deliniate one message from another. Neither is it permissable to frame the IFSF message with additional information for the purpose of delineating the beginning and end of a message. The recommended way to determine an IFSF message boundary is to use the IFSF message length field. The IFSF message length field is in the same position in all IFSF messages (block cutting not supported). This requires that the implementation keep synchronized with the messages coming to it and at any time it detects that there is confusion about the beginning and/or end of a message it should go into a recovery mode where it forces the sending host to retransmit a message in its entirety. This recovery mode may consist of not forwarding any questionable messages to the local application, thereby creating a timeout condition at the sending host. The sending host should detect the timeout and resend any messages that it has not received a response for.

5 IFSF over TCP/IP - services and options

The TCP/IP protocol suite offers many services and each of those services has options to enable their efficient use within the environment they are used. To support IFSF over TCP/IP only a few of these services are required and many of the options they offer are not needed. For several of the services within the suite the options are no longer required since they were developed when networking hardware and computers were less powerful and took longer to process the frames.
A minimum IP stack to implement IFSF over TCP/IP includes the following:

- IP
- ARP
- ICMP
- TCP
- UDP
- DHCP (client or server depending on device)

These services are the basic ones required allowing one machine to communicate with another. It should be pointed out that a device could implement BOOTP client instead of DHCP client, but it would be taking a chance that the DHCP server on site supports BOOTP clients, which is not required. It is advisable that all equipment tied to the LAN at a site allows for manual programming of network information in the case that there is not a compatible address server on site.
As an additional note it may be necessary for an equipment installer to have access to the controller on-site to set up download information for the device being installed.

Additional services that may be considered are:

- TFTP
- Domain Name System (DNS)

These additional services add the ability to do application downloads at boot time and the ability to resolve names to addresses. Name resolution has some advantages to on-site communications, but adds more to off-site communications.

6 IFSF over TCP/IP - Architecture

An example of architecture for an IFSF device with a TCP/IP interface is shown below.

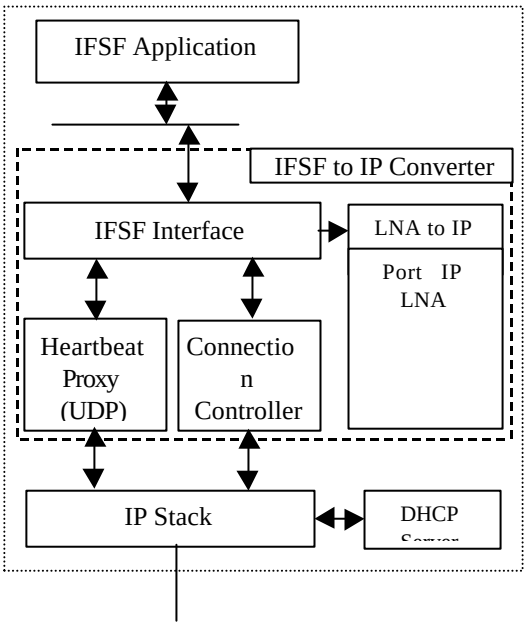
There are four main components

6.1 The IFSF application

The IFSF application is as described in the respective IFSF specifications. It is important to note that the application will remain the same whether the communication transport is LON or TCP/IP.

6.2 IP Stack

The IP stack is the interface to the network. It implements the various IP protocols and provides services to manage connections, resolving IP addresses, etc. Protocol stacks are available as off-the-shelf commodities, which can readily be purchased. The detailed operation of this component is outside the scope of this document, it is described extensively elsewhere.



6.3 DHCP server

The DHCP Server is used to distribute IP addresses to all the IP devices on a network. It may be part of an IFSF device, or it may be a separate device. There must only be one DHCP server on the network.

6.4 IFSF to IP Converter

The IFSF to IP converter module (hereafter referred to as the IIPC) has the responsibility to look like an IFSF interface to the local IFSF application, accepting all IFSF messages and placing them in IP datagrams to send to a remote device over the local LAN. The module has three main objectives - to send and receive heartbeats via the heartbeat proxy, keep a list of all active connections on the LAN, and package up all data and control messages into TCP streams for the LAN.

The IIPC module consists of 3 functional blocks

6.4.1 IFSF interface

This module is responsible for the interface between the IFSF application and other IP communication services. It will maintain a table of all LNA's and their corresponding IP/port addresses (a combination of IP address, protocol and port number corresponds to the socket address). This module will route all heartbeat messages to the **heartbeat proxy** and

all other messages to the **connection controller**. This module will receive heartbeats from other devices, add the LNA, socket address to the table (if not already in the table), and then send the IFSF heartbeat to all applications the interface is hosting.

6.4.2 Heartbeat proxy

This module is responsible for packaging local application heartbeat messages and broadcasting them using UDP datagrams. It is from here the UDP datagram is broadcast to the 'well known' IFSF heartbeat port. Incoming heartbeat messages come through this module, and are sent through the **IFSF interface**. The proxy will also send the heartbeats it receives from a local device to other locally hosted devices.

An IFSF heartbeat contains the LNA and a device status bit. To be effective on the IP network this message needs to have augmented to it the IP address of the host of the local IFSF application and the port number on the local host that a remote device uses to connect to the local IFSF application. When a remote device receives this message it will strip off the IP and port number information, record the LNA of the sending device, and pass the IFSF heartbeat message on to the IFSF application in the standard IFSF protocol format. The remote device will take the data from the received message and make an entry in a table that maps the IP and port address to the LNA of a device that has announced itself to the network, which we will call the LNA to IP mapping table.

Each time a heartbeat message is received by the IIPC it will reset a timer for that remote device. The purpose of the timer is to notify the IIPC when a heartbeat has not been received for a period of time. When the IIPC gets that notice it assumes the remote device has gone off-line and removes it from the LNA to IP mapping table, sends a connection closed message to the remote device, and closes any local connections associated with the device other than the main service connection. The next time a heartbeat or TCP connection request comes in from the remote device then a new entry will be made in the LNA to IP mapping table and the timer is started again.

6.4.3 Connection controller

This module is responsible for managing the TCP connections. Any IFSF message other than a heartbeat message is handled by this interface.

All data and control messages will be wrapped in TCP and sent to the appropriate address. The appropriate address is determined by taking the LNA information from the IFSF message and finding the corresponding socket address from the LNA to IP mapping table. The receiving station will accept the message and strip off the TCP wrapper, passing the IFSF message on to the device application.

Sending an IFSF message.
If an application hosted by this interface sends an IFSF message, the connection controller will check if there is a TCP connection to the required IFSF device. If not, a request to set up a connection will be sent to the socket address hosting the IFSF application. This application will acknowledge the request and return the port number to be used for this communication session. From now on until the connection is broken, all communications (except heartbeats) between these applications will be handled using this socket address.

Receiving an IFSF message.
When a TCP connection request is received, the connection controller will select the next unique port number (i.e. one that is not in use by any current connections hosted by this controller), and return this port number as the one to use for this connection. This port number will be held in a table to identify to the connection controller which IFSF application is hosted by this port.

7 Sequences for IFSF over TCP/IP communication.

7.1 Initial start-up preparation – before any IFSF communications

1. DHCP server has to be set up with its own IP address and the range of IP addresses to be leased to clients
2. All other devices need to have their node numbers set-up as in the LNA address.

7.2 Communication initiation

1. Independently, each TCP/IP stack will request IP address from the DHCP server using UDP. Optionally other information such as file name for software downloads may be supplied at this time. It is possible that some devices, or a whole site, may want to use static addresses. If this is the case then each device must have the ability to program in the networking information at the device and accommodations made with the DHCP server as required. It is strongly recommended that static IP addressing not be used.
2. The **heartbeat proxy** will set up the 'well known' port at the IP stack so that it can receive incoming heartbeat messages.
3. Each application will send a heartbeat to the **IFSF Interface**. On receiving the heartbeat the **IFSF Interface** will:

    i. Register the application (associate this IFSF application communication channel with the IFSF LNA).
    ii. Use the IP stack to get a socket address via the TCP interface.
    iii. Enter into its LNA/IP table the IFSF LNA address and the socket address. Each hosted IFSF application will be allocated a unique port address. Once this step is completed for all hosted IFSF applications, a table will exist identifying IFSF LNA's to socket addresses.
    iv. A heartbeat message will be broadcast via UDP to the 'well known' heartbeat port, containing the socket address assigned for this application. This will be repeated for each IFSF application hosted by this **IFSF Interface**.

Now all the configuration housekeeping tasks have been completed to allow all hosted IFSF applications to send/receive both heartbeats and explicit messages.

7.3 Communication operation

7.3.1 Heartbeats

The **IFSF interface** will route heartbeat messages from each application that it is hosting to the **heartbeat proxy**. These messages will then be broadcast using UDP datagrams
All incoming heartbeats will be examined and entries, where needed, will be made to the LNA/IP table. The heartbeats will then be passed up to all IFSF applications hosted by this interface.

7.3.2 Explicit messages

On receiving an explicit IFSF message from an application, the connection controller will check if a connection to the required device has been set up, and if so, send the message to the associated socket address. If no, it will get the socket address from the LNA/IP table, set up a connection, and send the message. Incoming messages will be examined, and based on the socket address, routed to the appropriate application.

8 IP Service options with the minimum services.

This section discusses parameters associated with each IP service, along with guidance as to its usage.

8.1 IP

IP (Internet Protocol) has several options (Type of service and IP options) that have to do with directing routes and delivery of data. However these options only pertain to frames that must pass through routers for delivery to the destination. If a frame is staying within the local network then these options add little to no value to the delivery of the message.

8.2 ARP

The Address Resolution Protocol is a mechanism to distribute IP addresses. It is not used for IFSF message transfers, so there are no options for IFSF to set-up.

8.3 ICMP

The Internet Control Message Protocol is an error reporting mechanism to help diagnose network problems. It is not used for IFSF message transfers, so there are no options for IFSF to set-up.

8.4 TCP

The size of the window should be set to max out at the size of the receive buffer of the local host. The receive buffer of the local host should be able to handle multiple messages to make better use of the medium. It will be important that the supplier of the equipment pick an IP stack that has a good window control heuristic. A good heuristic will have mechanisms built in to keep data flowing without overflowing the buffer, causing retransmissions. It will also avoid sending segments on the link that are so short they cause an inefficient use of bandwidth. See RFC 1106 for other precautions on window management.

8.5 UDP

The User Datagram Protocol has no message options to set up

8.6 DHCP

DHCP has no message options to be concerned with, however it has optional information that can be delivered to a client when it is requesting an IP address. Of these the following should be standard:

- The new IP address
- The default router/gateway address
- Subnet mask

There are some optional pieces of information to share if applicable to the site:

- DNS server address
- Boot file name
- TFTP server name

8.7 TFTP

There are no options with TFTP. The TFTP server should have sufficient time outs to allow a device to be slow in delivering the acknowledgement to a message. If the time out at the server is too short it is possible to corrupt the image being loaded in the memory of the device.

9 Networking and Name Resolution

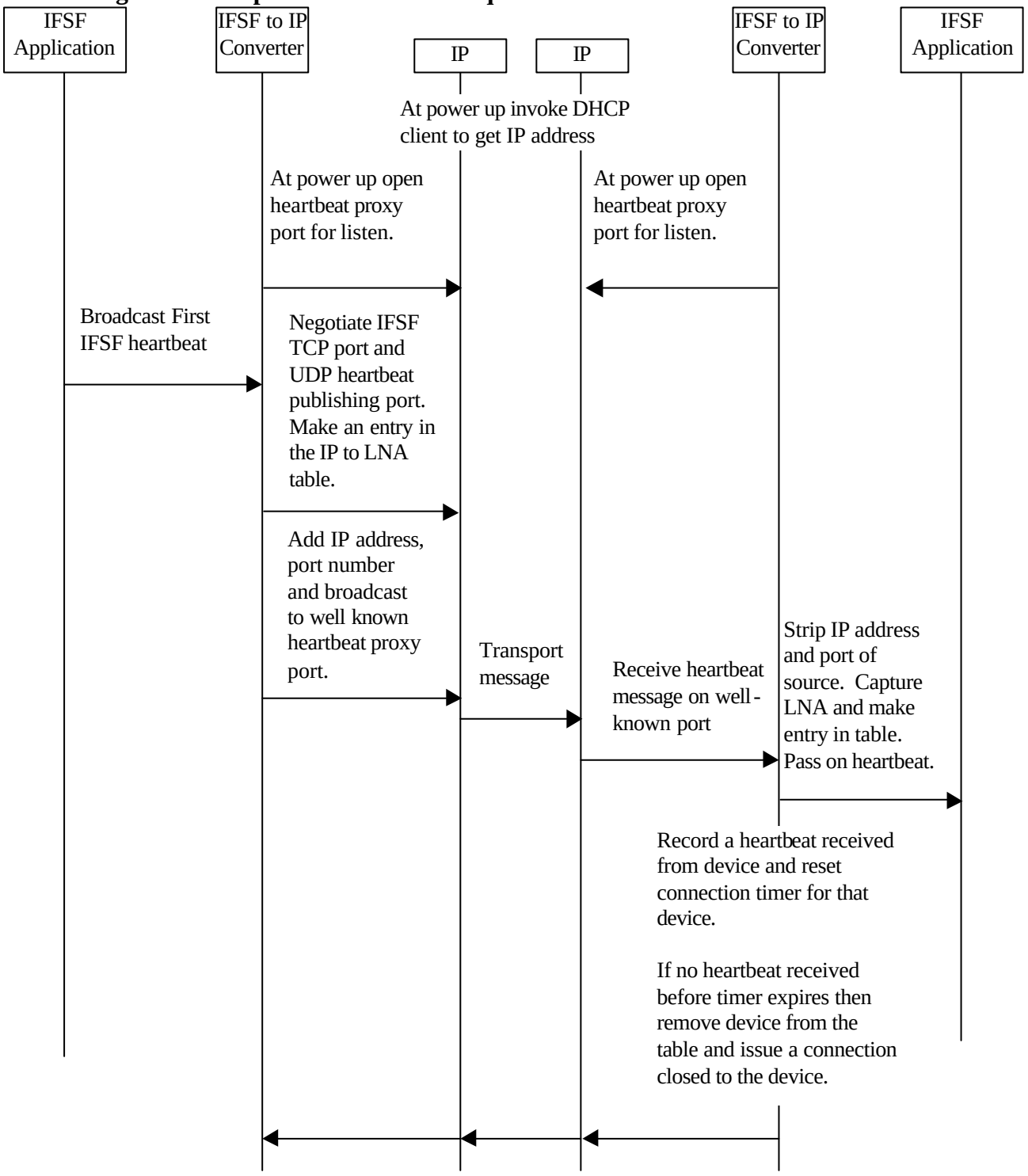
Between IFSF devices the method of resolving where to send a message have been detailed. For a device to connect to a non-IFSF computer for on-site or off-site communications requires the implementation of some type of name resolution mechanism. This is usually accomplished through a "host" file or DNS or both. In a strictly NT environment WINS is used, but WINS is limited to NT machines only. In an environment where different types of OS's are in use then DNS and the host file are the only choices for interoperability.

To use a host file host name to IP address mapping must be manually placed in the host file. Any change in the network environment must be reflected in each host file on each machine in the network. For small networks this is an easy task, the larger the network the more difficult it gets. DNS is the answer for the medium to large networks. The negative is the set-up work involved to set-up a DNS server. Once in place there is a single point that has to be administered not every machine on the network.

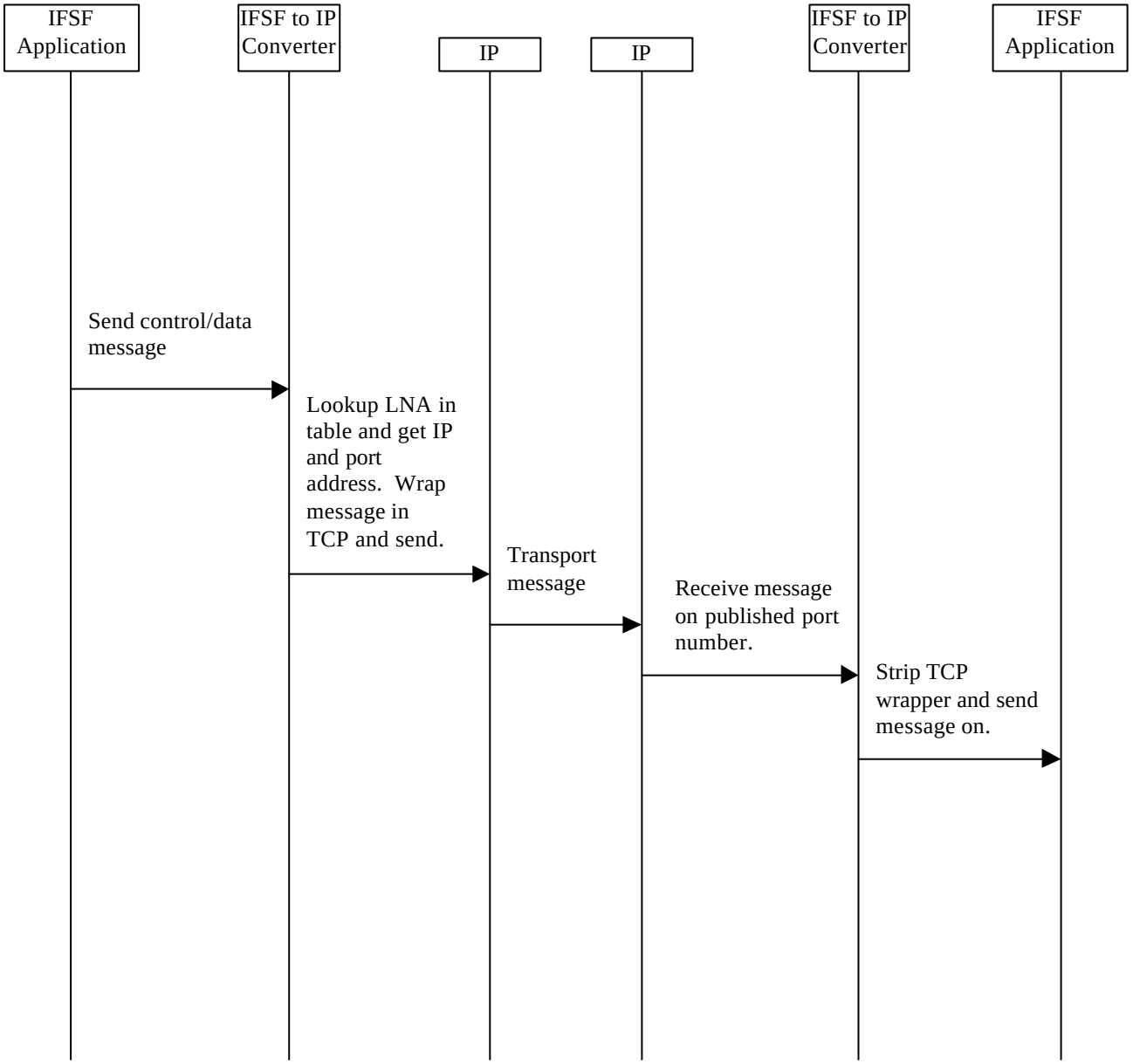
10 Sequence Diagrams

The following sequence diagrams give an overview of the functions of the IIPC.

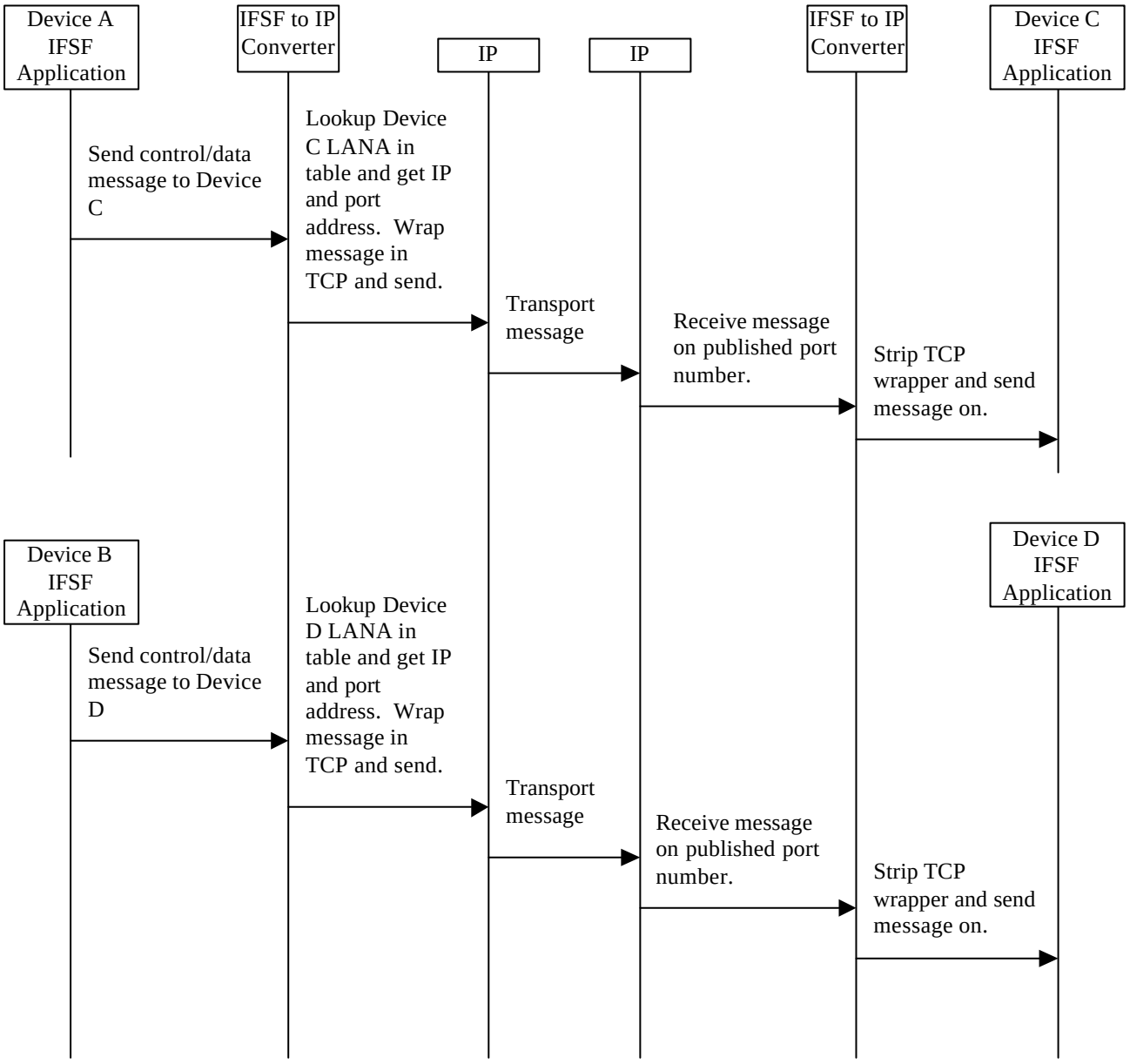
10.1 Figure 1 Startup and Initialization Sequence



10.2 Figure 2 Sending a TCP Message



10.3 Figure 3 Two IFSF Applications on One Host Communicating with Remote IFSF Devices



10.4 Detailed Examples

This section illustrates some typical examples of how IFSF over TCP/IP communication works. Here is detailed how the IP, Port and IFSF LNA could be implemented.

This example is for one type of architecture, it is not meant to imply this is the only architecture.

The following examples show the message handling inside the dispenser/cardreader.

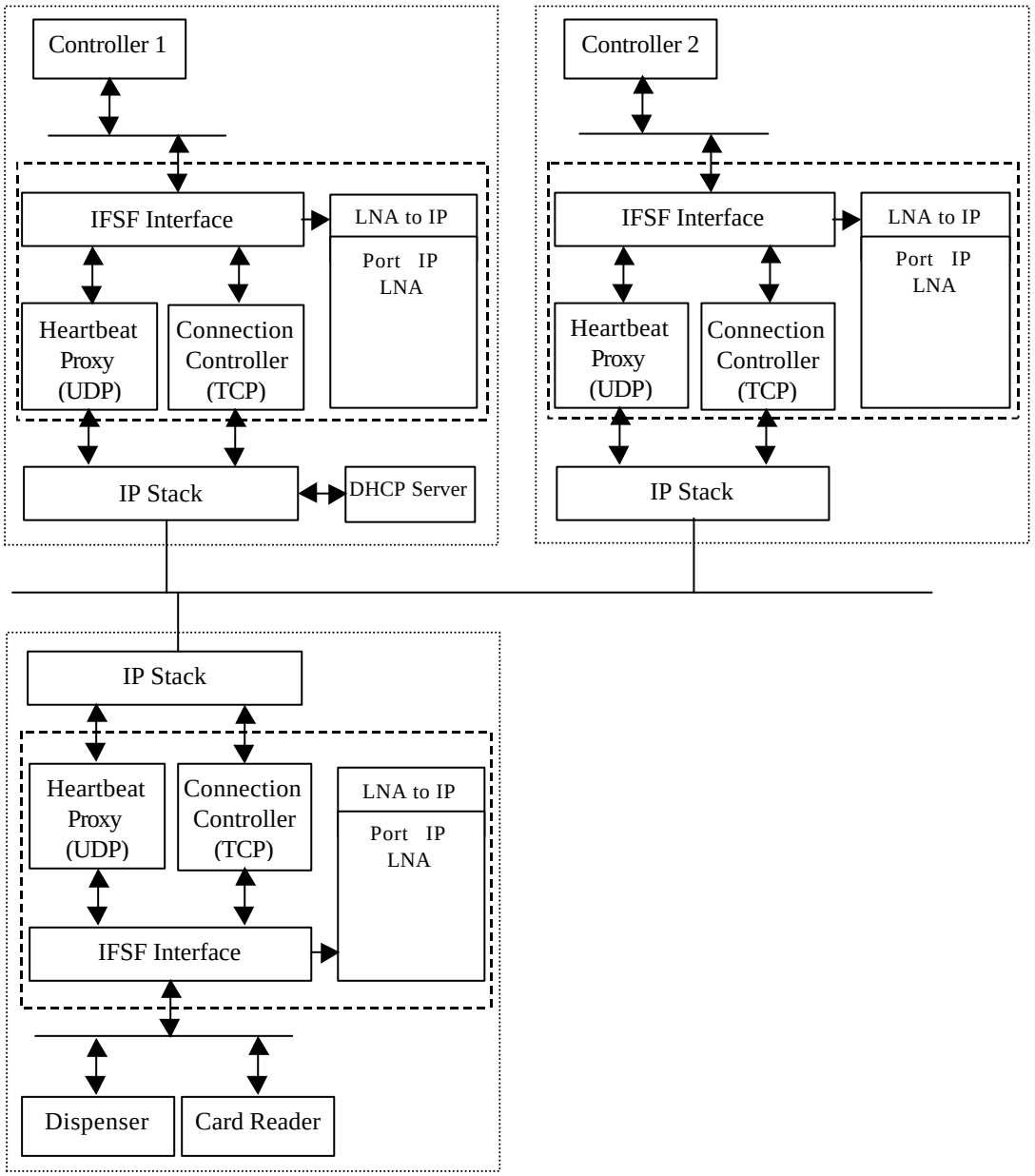
The IP, port and LNA values used are for example only.

The sequences show the establishment of connection through to a number of different communication scenarios.

10.4.1 Configuration used in the following example

This example uses a forecourt dispenser with integral card reader, controlled by one of two controlling devices. The dispenser has two independent IFSF applications, one controlling the dispenser, the other controlling the card reader.

The IFSF Protocol Converter (IIPC) is shown bound by the innermost dotted line. This is the collection of applications responsible for interfacing the IFSF application, with the protocol stack



10.4.2 Establishing Heartbeats

This example shows the establishment of heartbeats from power up.

IFSF Application		IIPC		Network
Dispenser Application Heartbeat	→	Ignored		
Card Read Application Heartbeat	→	Ignored		
		Broadcast DHCP Request	→	DHCP Request
		Receives reply from DHCP Server Allocated IP address for Dispenser/Card reader is 192.1.1.1	←	DHCP reply
		Heartbeat Proxy opens up UDP port for sending heartbeats and listening for remote heartbeats.		
		Connection Controller opens up TCP port for listening for connection requests for each local IFSF device. Entries are made in the local LNA to IP mapping table for each local application.		
Dispenser Application Heartbeat	→	Heartbeat Proxy sends broadcast UDP message • Source IP = 192.1.1.1 • Destination IP = 255.255.255.255 • Source Port = TCP listen socket • Destination Port = IFSF H/B port • Data = Dispenser LNA (01 01)	→	UDP Broadcast
Cardreader Application Heartbeat	→	Heartbeat Proxy sends broadcast UDP message • Source IP = 192.1.1.1 • Destination IP 255.255.255.255 • Source Port - TCP listen socket • Destination Port = IFSF H/B port • Data = Cardreader LNA (05 01)	→	UDP Broadcast
Receives Controller 1 Heartbeat	←	Heartbeat Proxy receives UDP broadcast on IFSF H/B port • Extracts Source IP and Port for Controller 1 (192.1.1.21) • Extracts LNA for Controller 1 (02 21) • Enters LNA/IP data for Controller 1 into LNA to IP Map	←	UDP heartbeat from Controller 1
Receives Controller 2 Heartbeat	←	Heartbeat Proxy receives UDP bradcast on IFSF H/B port • Extracts Source IP and Port for Controller 2 (192.1.1.22) • Extracts LNA for Controller 2 (02 22) • Enters LNA/IP data for Controller 2 into LNA to IP Map	←	UDP heartbeat from Controller 2

10.4.3 Simple message transfer

This section shows controller 1 sending a command to dispenser 1

IFSF Application		IIPC		Network
IFSF dispenser application sends an IFSF read message to Controller 1	→	IIPC determines that there is no TCP connection established between Controller 1 and dispenser application • It first assigns a port that will identify the dispenser application for this connection request (1111) • Connection controller sends TCP message requesting a connection • Source IP = 192.1.1.1 • Destination IP = 192.1.1.21 • Source Port = 1111 • Destination Port = from LNA to IP table	→	TCP
		• Message received acknowledging connection, and identifiyng controller port for this connection to be • 2222 • IIPC adds this port address into the correct LNA to IP map entry	←	TCP
		IIPC sends TCP message containing IFSF message • Source IP = 192.1.1.1 • Destination IP = 192.1.1.21 • Source Port = 1111 • Destination Port = 2222 • IFSF application message	→	TCP
IFSF application receives reply to first read message		Response to IFSF read message • Source IP = 192.1.1.21 • Destination IP = 192.1.1.1 • Source Port = 2222 • Destination port = 1111 • IFSF Application message	←	TCP
IFSF dispenser application sends a second IFSF read message to Controller 1	→	IIPC determines that there is a TCP establised connection between Controller 1 and dispenser application IIPC sends TCP message containing IFSF message • Source IP = 192.1.1.1 • Destination IP = 192.1.1.21 • Source Port = 1111 • Destination Port = 2222 • IFSF application message	→	TCP
Dispenser IFSF application receives reply to second read message	←	Response to IFSF read message • Source IP = 192.1.1.21 • Destination IP = 192.1.1.1 • Source Port = 2222 • Destination port = 1111 • IFSF Application message	←	TCP

10.4.4 Two controllers sending commands to one device

This example shows controller 1 and controller 2 both sending commands to dispenser 1

IFSF Application		IIPC		Network
Dispenser IFSF application receives command from controller 1		Receives TCP message from controller 1 • Source IP = 192.1.1.21 • Destination IP = 192.1.1.1 • Source Port = 2222 • Destination port = 1111 • IFSF Application message From the IP/Port address, the IIPC recognises a current connection is already established, and uses this to send the IFSF message to the dispenser application	←	TCP
		Receives TCP request from controller 2 for a connection and port • Source IP = 192.1.1.22 • Destination IP = 192.1.1.1 • Source Port 3333 • Destination Port 2223 IIPC finds an unused port, and accepts the connection using this port. Adds this port in the LNA to IP map	←	TCP
		Replies accepting connection • Source IP = 192.1.1.1 • Destination IP 192.1.1.22 • Source Port = 2223 • Destination Port = 3333	→	TCP
Dispenser IFSF application receives command from controller 2	←	Receives TCP message from controller 2 • Source IP = 192.1.1.22 • Destination IP 192.1.1.1 • Source Port 3333 • Destination Port 2223 From the IP/Port address, the IIPC recognises a current connection is already established, and uses this to send the IFSF message to the dispenser application	←	TCP

11 Appendix 1
Minimum Number of TCP-IP Sockets Required for IFSF Device Types

Device Type	Minimum Number of TCP/IP Connections/Sockets	Comment
Dispenser	12+1	Based on: 8 POS/SC devices connecting 2 Tank Level Gauges 2 Copt/BOS or other devices == 12 Total number of devices that might want to connect. +1= Additional 'well known' port allowing other devices to connect to Control Device.
Control Device (SC/POS/BOS)	X+1	X=Number of connections dictated by number of IFSF TCP/IP devices to be controlled. +1= Additional 'well known' port allowing other devices to connect to Control Device.
Price Pole	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
Tank Level Gauge	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
BNA	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
COPT	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
Pin Pad	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
Printer	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
Card Reader	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.
Car Wash	8+1	+1= Additional 'well known' port allowing other devices to connect to Control Device.

Other IFSF Devices	8+1	Unless additional Requirements are defined 8+1 is the standard minimum requirement for other devices.

12 Appendix 2
IFSF over TCP/IP Implementation Using Socket API

12.1 Terms and abbreviations

IP	**I**nternet **P**rotocol
IP ADDRESS	Internet address (four bytes, usually written in dot notation, e.g. 192.168.2.12)
TCP	**T**ransfer **C**ontrol **P**rotocol
UDP	**U**ser **D**atagram **P**rotocol
TCP/IP	The family of protocols including IP, TCP and UDP
LNA	IFSF **L**ogical **N**ode **A**ddress. The LNA consists of the Subnet and Node
LANO	LNA of the **O**riginator
LNAR	LNA of the **R**ecipient
IPC	**I**nter**P**rocess **C**ommunication. The IPC is used to transfer data between different processes (possibly running on different computers – i.e. network transparently). For example a PIPE in Unix or a Window Message in MS Windows.

12.2 Introduction

This Appendix describes a possible implementation of the IFSF TCP/IP Communication Standard using Socket API in Unix (Linux OS) and/or MS Windows 9x/ME/2000 environment.

The Socket API was originally designed for the Unix platform and was adopted by the MS Windows 9x/ME/2000 platform. It means that from the application point of view the definition of the Socket API functions is the same for both the platforms. Consequently, the applications implemented using Socket API on different hosts, and running under different operating systems – Unix and Windows – are able to communicate. It allows describing the Socket API independently of the running platform.

The IFSF implementation described below consists of two main layers (see *Figure 1*).

1. The IFSF lower layer performs the (LON, TCP/IP) independent services for the higher layer. The services performed by lower layer are reception and transmission of the IFSF heartbeat messages and the IFSF messages.
2. The IFSF higher layer can implement either an IFSF forecourt device (according to the appropriate IFSF device standard) or an IFSF controller device. The IFSF higher-level modules have to be implemented according to the IFSF Communication Specification [1], and in case of the forecourt device according to the appropriate forecourt device standard (e.g. IFSF Dispenser application [2]).

The interface between the two layers mentioned above enables transmission and reception of the IFSF messages and the IFSF heartbeat messages. It is defined by [1].

The higher layers have been defined by an IFSF Communication Specification [1], and IFSF Forecourt Device Standards ([2],…). The IFSF lower layer is described here, and – especially – the implementation of the **IFSF TCP/IP Gate** module. The brief description of each module of the IFSF lower layer is also presented here to make the integration of the IFSF TCP/IP Gate module to the entire system clear.

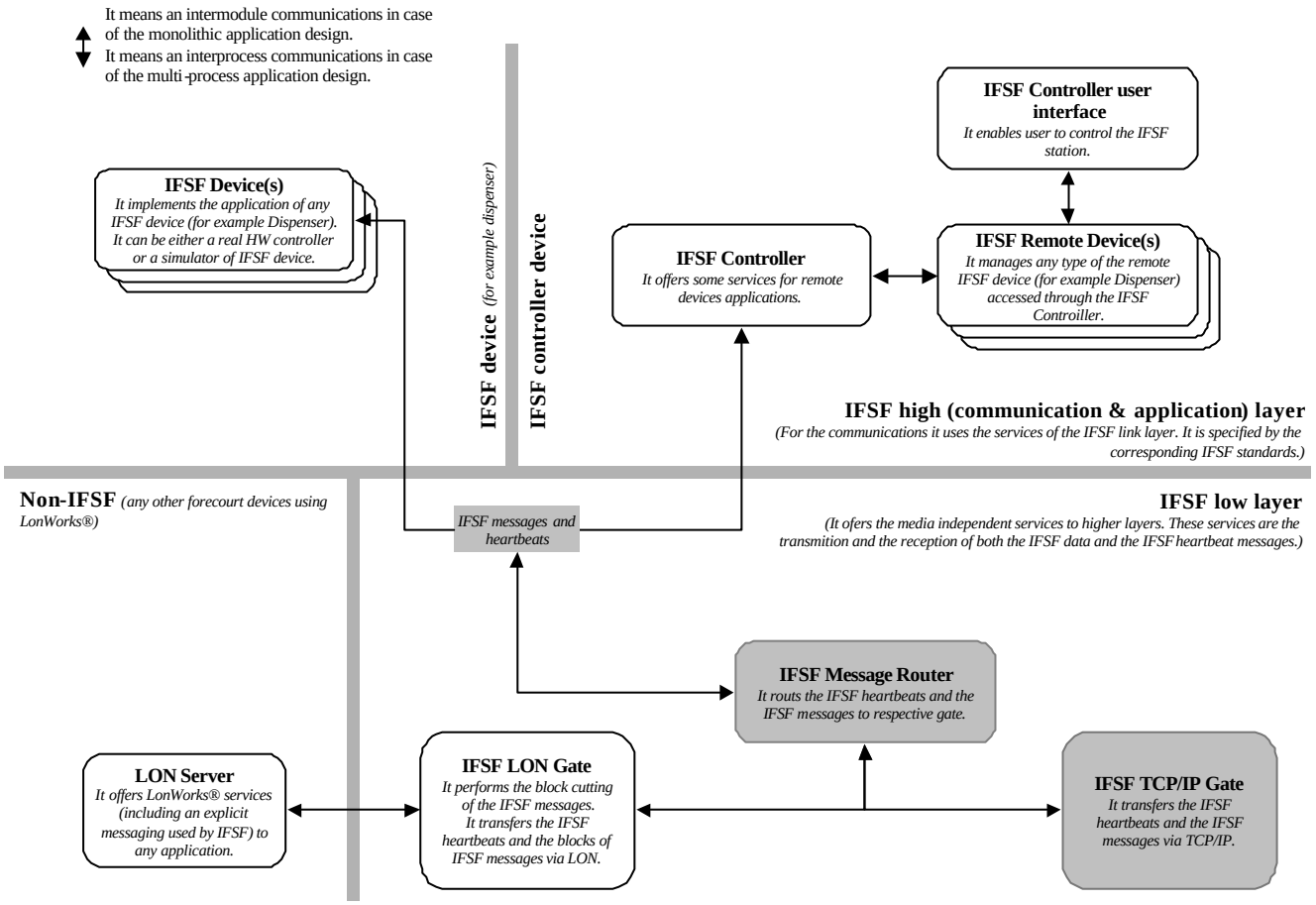


Figure 1: The complete structure of the IFSF application

Note:

The lower layer was implemented to enable not only the combination of LON and TCP/IP IFSF device, but also the non-IFSF devices connected to the LON bus. I.e. the lower layer module offers the communication services to IFSF forecourt device applications (LON and/or TCP/IP), IFSF controller device(s) applications, and the general LonWorks® control applications. As the interface to the services is the InterProcess Communication, the application **modules of different vendors can co-operate**.

12.3 IFSF lower layer

The modularity of the implementation (see *Figure 1*) allows the higher layers independence from the actual communication media (LON, TCP/IP). It even supports the communication media combinations.

The main part of the lower layer is the **IFSF Message Router** module. This module offers one and only one interface, which is defined by the IFSF Communication Specification [1] and which allows the transmition and the reception of both the IFSF messages and the IFSF heartbeats. This interface is shared by the IFSF higher layer and by one or more Gate Modules.

Note:
The IFSF messages going via the interface of the **IFSF Message Router** module are prefixed by one byte of the Max_Block_Length information, which is needed by the **IFSF LON Gate** module to perform the block cutting.

Now there are three Gate modules defined by means of the IFSF standards:

1. The **IFSF LON Gate** module and
2. The **IFSF TCP/IP Gate** module.
3. From the point of view of the **IFSF Message Router** module the IFSF higher layer is also a “Gate” module.

12.3.1 IFSF Message Router module

The **IFSF Message router** module (see *Figure 2*) uses the following rules when routing messages.

1. The IFSF heartbeat incoming to the **IFSF Message Router** module is routed to all connected “Gate” modules excluding the originating one. The **List of Gates** (see *Figure 2*) is used to do that. The IFSF heartbeat received is also used to maintain the **Address Table**.
2. The IFSF message incoming to the **IFSF Message Router** module is routed to the “Gate” module defined in the **Address Table** by its LNAR. If there is no “Gate” module defined to correspond with LNAR of the message, the message is dropped.

Note: *An IFSF device which does not send IFSF heartbeats can not be accessed by IFSF messages as the **IFSF Message Router** module does not know how to route them.*

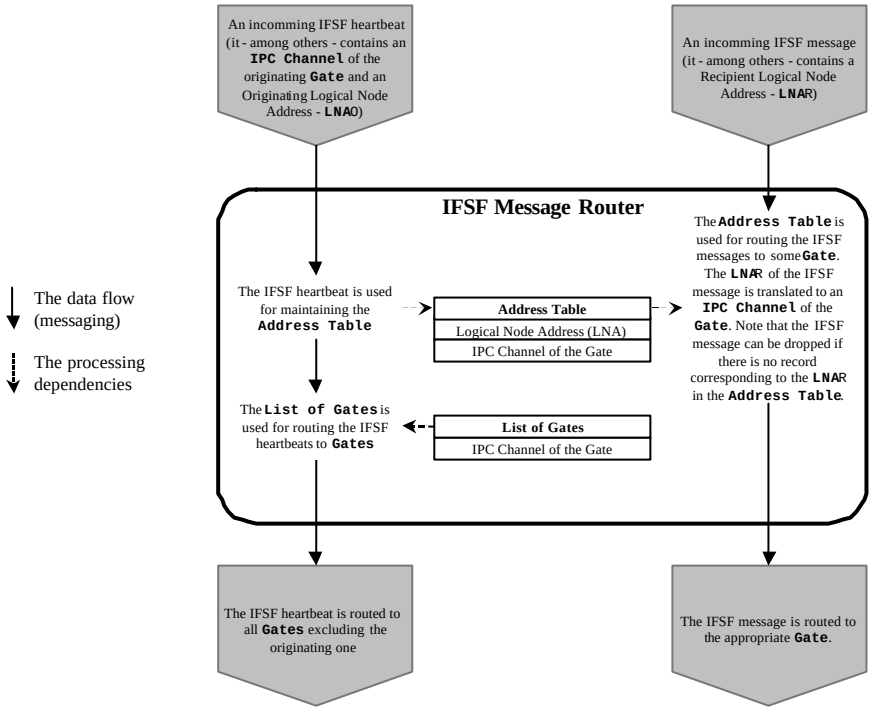


Figure 2: The structure of the IFSF Message Router module

12.3.2 IFSF TCP/IP Gate module

The **IFSF TCP/IP Gate** module transfers the IFSF messages and the IFSF heartbeats using the TCP/IP protocol family. Actually the IFSF messages are transferred using the TCP protocol and the IFSF heartbeats are transferred using the UDP protocol.

The TCP/UDP protocols are accessed through the Socket **A**pplication **P**rogramming **I**nterface (**Socket API**).

The implementation allows creating two slightly different architectures of the TCP connections of the IFSF devices – see below.

The well-known port has to be defined for the UDP socket through which the IFSF heartbeat messages are transferred. In the paragraphs below the name used for the well-known port is **`HB_PORT`**.

12.3.3 Socket API

The **Socket API** was designed to unify the TCP/IP protocol family interface in Unix environment. This API has become a standard and has been accepted by the MS Windows 9x/ME/2000.

The socket API is a set of constants, structures and functions. The most common definitions are mentioned below in this paragraph. The description is not comprehensive, for historical and technical details see, please [3]. For the programming reference see, please [4] or [5]. Finally, the description below uses the native programming language of the Socket API, which is "C".

12.3.3.1 Socket address structure

The socket API **`sockaddr`** structure varies depending on the protocol selected. The default definition is following:

```
struct sockaddr {
  u_short sa_family;    // a related family of protocols
  char    sa_data[14];  // an address
};
```

The structure below is used with the TCP/IP protocols family. Note that all values should be stored in the network byte order.

```
struct in_addr {
  u_long s_addr;            // an IP address value
};

struct sockaddr_in {
  short   sin_family;       // AF_INET value
  u_short sin_port;         // a port number
  struct  in_addr sin_addr; // an IP address
  char    sin_zero[8];      // an unused area should contain 0
};
```

12.3.3.2 Function socket

The socket API **`socket`** function creates a socket.

```
int socket(
  int af,
  int type,
```

```
  int protocol
);
```

Parameters:

- **af** – an address family specification, it should be **AF_INET**to work with an IP protocols,
- **type** – a socket type specification, it should be either **SOCK_STREAM** to create a TCP socket or **SOCK_DGRAM** to create an UDP socket,
- **protocol** – a protocol to be used with the specified address family, it should be **0** to select the default protocol.

Returned values:

If no error occurs, function **socket** returns a descriptor referencing the new socket. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.3 Function bind

The socket API **bind** function associates a local address with a socket.

```
int bind(
  int s,
  const struct sockaddr *name,
  int namelen
);
```

Parameters:

- **s** – a descriptor identifying an unbound socket,
- **name** – an address to assign to the socket from the **sockaddr** structure,
- **namelen** – a length of the value in the **name** parameter.

Returned values:

If no error occurs, **bind** returns zero. Otherwise, a value of **INVALID_SOCKET** is returned.

Note:

Providing that the **AF_INET** address family is used, the port number (see 12.3.3.1. *Socket address structure*) value **0** instructs the **bind** function to select a first free port automatically.

12.3.3.4 Function listen

The socket API **listen** function enables a socket to the state where it is listening for an incoming connection request(s).

```
int listen(
  int s,
  int backlog
);
```

Parameters:

- **s** – a descriptor identifying a bound, unconnected socket,
- **backlog** – maximum length of the queue of pending connections.

Returned values:

If no error occurs, **listen** returns zero. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.5 Function accept

The socket API **accept** function can accept the incoming connection request on a socket.

```
int accept(
  int s,
  struct sockaddr *addr,
  int *addrlen
);
```

Parameters:

- **s** – a descriptor identifying a socket that has been placed in a listening state with the **listen** function; the connection is actually made for the socket that is returned by **accept**,
- **addr** – an optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer; the exact format of the **addr** parameter is determined by the address family that was established when the socket was created,
- **addrlen** – an optional pointer to an integer that contains the length of **addr**.

Returned values:

If no error occurs, **accept** returns a descriptor for the new socket. This returned value is a handle for the socket on which the actual connection is made. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.6 Function connect

The socket API **connect** function establishes a connection to a specified socket.

```
int connect(
  int s,
  const struct sockaddr *name,
  int namelen
);
```

Parameters:

- **s** – a descriptor identifying an unconnected socket,

- **name** – a name of the socket to which the connection should be established,
- **addrlen** – a length of **name**.

Returned values:

If no error occurs, **connect** returns zero. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.7 Function send

The socket API **send** function sends data through a connected socket.

```
int send(
  int s,
  const char *buf,
  int len,
  int flags
);
```

Parameters:

- **s** – a descriptor identifying a connected socket,
- **buf** – a buffer of the outgoing data,
- **len** – a length of data in **buf**,
- **flags** – a flag specifying the way in which the call is made.

Returned values:

If no error occurs, **send** returns the total number of bytes sent, which can be less than the number indicated by **len** for nonblocking sockets. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.8 Function recv

The socket API **recv** function receives data from a connected socket.

```
int recv(
  int s,
  char *buf,
  int len,
  int flags
);
```

Parameters:

- **s** – a descriptor identifying a connected socket,
- **buf** – a buffer for the incomming data,
- **len** – a length of **buf**,
- **flags** – a flag specifying the way in which the call is made.

Returned values:

If no error occurs, **recv** returns the number of bytes received. If the connection has been gracefully closed, the returned value is zero. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.9 Function sendto

The socket API **sendto** function sends data on a specific destination.

```
int sendto(
  int s,
  const char *buf,
  int len,
  int flags,
  const struct sockaddr *to,
  int tolen
);
```

Parameters:

- **s** – a descriptor identifying a (possibly connected) socket,
- **buf** – a buffer of the outgoing data,
- **len** – a length of data in **buf**,
- **flags** – a flag specifying the way in which the call is made,
- **to** – pointer to the address of the target socket,
- **tolen** – size of the address in **to**.

Returned values:

If no error occurs, **sendto** returns the total number of bytes sent, which can be less than the number indicated by **len**. Otherwise, a value of **INVALID_SOCKET** is returned.

12.3.3.10 Function recvfrom

The socket API **recvfrom** function receives datagram and stores the source address.

```
int recvfrom(
  int s,
  char *buf,
  int len,
  int flags
  struct sockaddr *from,
  int *fromlen
);
```

Parameters:

- **s** – a descriptor identifying a bound socket,

- **`buf`** – a buffer for the incomming data,
- **`len`** – a length of **`buf`**,
- **`flags`** – a flag specifying the way in which the call is made,
- **`from`** – optional pointer to a buffer that will hold the source address upon return,
- **`fromlen`** – optional pointer to the size of the **`from`** buffer.

Returned values:

If no error occurs, **`recvfrom`** returns the number of bytes received. Otherwise, a value of **`INVALID_SOCKET`** is returned.

12.3.4 TCP/IP Connection Architectures

The implementation enables using the two different architectures for the TCP protocol (used for the transfer of the IFSF messages), which is connection oriented – see Figure 3.

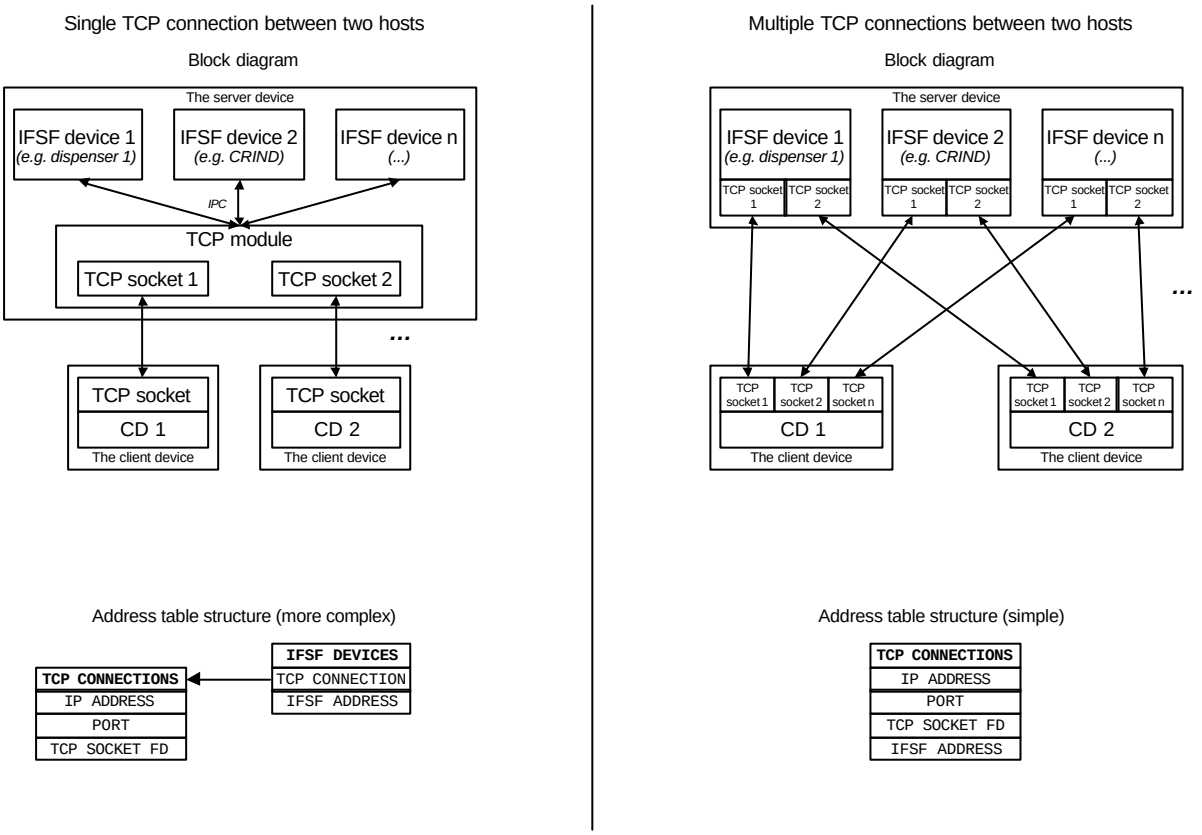


Figure 3: The two possible architectures of the TCP connections created among IFSF devices

The basic part of the TCP/IP Gate module is an **ADDRESS TABLE**. The address table is used to translate the LNAR of the outgoing IFSF message to the TCP connection. The structure of the address table varies depending on the architecture.

12.3.4.1 Single TCP connection between two hosts

In this architecture the only one TCP connection exists between two hosts (regardless the number of the IFSF devices located on each host). Consequently, the address table has to be more complex in

this case. In fact, the Address Table structure is a simple relational database consisting of the two tables:

- The **IFSF DEVICES** table and
- The **IFSF TCP CONNECTIONS** table.

The **TCP CONNECTIONS** table fields (see the left side of the Figure 3):

1. **IP ADDRESS** – contains an IP address of the peer host.
2. **PORT** – contains the port number of the application running on the peer host.
3. **TCP SOCKET FD** – contains the file descriptor (handle) of the local TCP socket connected to the peer host.

The **IFSF DEVICES** table fields (see the left side of the Figure 3):

1. **TCP CONNECTION** – it is the reference to the TCP connection, which is to be used to transfer the IFSF messages for particular IFSF recipient device.
2. **IFSF ADDRESS** – contains the IFSF address (subnet, node) of the particular IFSF recipient device.

12.3.4.2 Multiple TCP connections between two hosts

The structure of the address table in this architecture is quite simple, as there exists a dedicated TCP connection between each pair of the IFSF devices, which need to transfer the IFSF messages to each other.

The **TCP CONNECTION** table fields (see the right side of the Figure 3):

1. **IP ADDRESS** – contains an IP address of the peer host.
2. **PORT** – contains the port of particular recipient IFSF device application running on the peer host.
3. **TCP SOCKET FD** – contains the file descriptor (handle) of the local TCP socket connected to the recipient IFSF device application running on the peer host.
4. **IFSF ADDRESS** – contains the IFSF address (subnet, node) of the particular IFSF recipient device.

12.3.5 TCP/IP overhead of IFSF messages

To assure the IFSF functionality using the TCP/IP protocols the standard IFSF heartbeat message has to be extended by aditional data (see *Figure 4*). All the data are transferred in the network byte order.

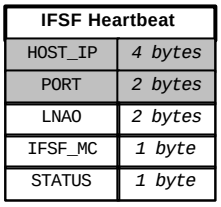


Figure 4: The TCP/IP overhead of the IFSF heartbeat

12.3.5.1 TCP/IP overhead of IFSF heartbeat

The IFSF heartbeat over TCP/IP is a fixed length message of the 10 bytes, which consists of two parts (see *Figure 4*).

The first part is the TCP/IP overhead:

1. **`HOST_IP`** – it is the IP address of the host where the originator IFSF application is running.**`PORT`** – it contains the port number of the **`SOCKET_SERVER`** TCP Socket, where the IFSF device TCP/IP module is listening for the connection requests – see **`listen`** function above.

The second part is inherited from the valid IFSF Communication Specification [1]:

1. **`LNAO`** – it is the originator IFSF Subnet, Node.
2. **`IFSF_MC`** – it is the IFSF message code.**`STATUS`** – it is the IFSF device status.

12.3.6 TCP/IP Gate module

The structure of the **IFSF TCP/IP Gate** module (see *Figure 5*) is built upon the first TCP/IP architecture (see the paragraph 12.3.4.1) as it consumes less system resources.

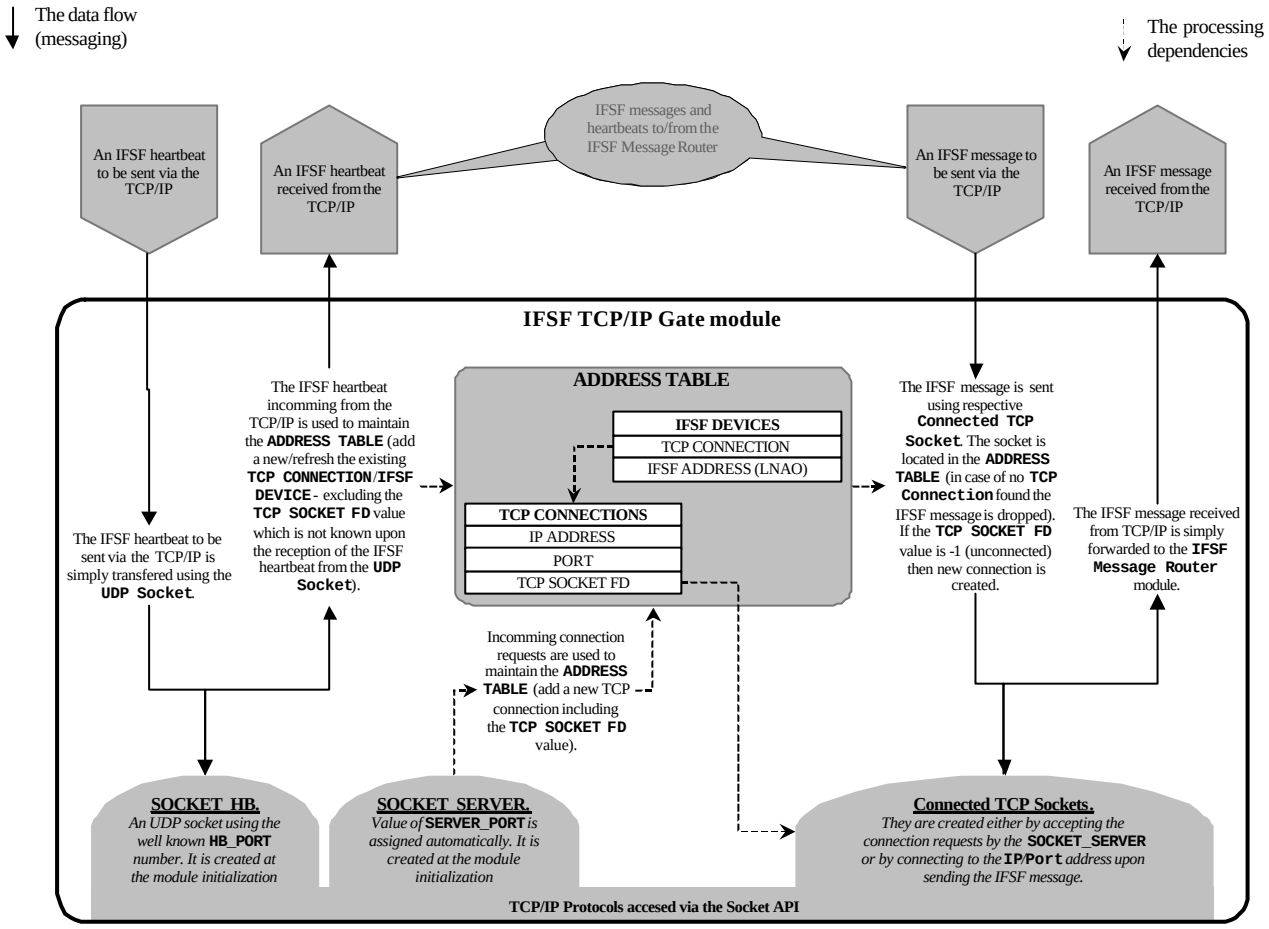


Figure 5: The detailed structure of the IFSF TCP/IP Gate module

The **IFSF TCP/IP Gate** module processes the following events:

1. The outgoing IFSF heartbeat message (to be sent to the TCP/IP).
2. The incomming IFSF heartbeat message (received from the TCP/IP).
3. The outgoing IFSF message (to be sent to the TCP/IP).
4. The incomming IFSF message (received from the TCP/IP).
5. The connection requests (received from the TCP/IP).

The following chapters describe how each of the events is processed.

12.3.6.1 Outgoing IFSF heartbeat message

The outgoing IFSF heartbeat message is, firstly, extended by the TCP/IP overhead (see paragraph TCP/IP overhead of IFSF heartbeat) to create the datagram. The datagram is then simply broadcast using the **SOCKET_HB** UDP socket (which is bound to well known **HB_PORT**).

12.3.6.2 Incomming IFSF heartbeat message

Every datagram received by the **SOCKET_HB** UDP socket consists of two parts:

- The TCP/IP overhead (see 12.3.5.1 TCP/IP overhead of IFSF heartbeat) and
- the IFSF heartbeat message.

The following information can be extracted from each datagram:

- The **IP ADDRESS** of the datagram originator,
- the **PORT** of the TCP server (listening for connection requests) socket of the datagram originator and
- the **LNAO**.

All of these parts of information are used to update the **ADDRESS TABLE** (i.e. the tables **TCP CONNECTIONS**, **IFSF DEVICES**).
The IFSF heartbeat message is then forwarded to the **IFSF Message Router** module.

12.3.6.3 Outgoing IFSF message

The outgoing IFSF message is processed in the following steps:

1. The **LNAR** of the IFSF message is used to locate the record in the **IFSF DEVICES** table. If there is no such record then the IFSF message is dropped (and possibly reported as undelivered).
2. The **TCP CONNECTION** field of the **IFSF DEVICES** table record is used to locate the record in the **TCP_CONNECTIONS** table.
3. The **TCP SOCKET FD** field of the **TCP CONNECTIONS** table record is explored. If the field does not contain the valid descriptor of the connected TCP socket (i.e. the connection is not active yet) then a new connection is created (using the **IP ADDRESS** and the **PORT** fields of the **TCP CONNECTIONS** table record) and stored in the field. In case the creation of the connection fails the IFSF message is dropped (and possibly reported as undelivered).

12.3.6.4 Incomming IFSF message

The IFSF message received from the connected TCP socket is simply forwarded to the **IFSF Message Router** module.

The following information can be extracted from each IFSF message incomming from TCP/IP:

- The **TCP SOCKET FD** and
- the **LNAO**.

Both of those parts of information are used to update the **ADDRESS TABLE** (**TCP CONNECTIONS**, **IFSF DEVICES**) as necessary.

12.3.6.5 Connection request

Each incomming connection request (received by the **SOCKET_SERVER**) is accepted and used to update the **TCP CONNECTIONS** table.

12.4 Example of the Startup

The following table shows the startup example of the IFSF via TCP/IP devices.

Controller device (CD)			**Dispenser**		
Verbal description	Socket API	TCP/IP	Verbal description	Socket API	TCP/IP
The IFSF CD starts up. - In case of using DHCP for IP address assignment the startup of the DHCP client is performed and it waits for the IP address dynamic assignment. In case of using the constant IP address the DHCP client will not be started and IP address is known. - IFSF TCP/IP Gate Module (**GM** below) creates the `TCP CONNECTIONS`/`IFSF DEVICES` address structures. The structures are empty now.		7.1 7.2, 1.	The IFSF dispenser is off now.		
- GM creates the `SOCKET_HB` UDP socket on the well-known `HB_PORT` for the reception/transmission of the HBs.	`socket()` `bind()`	7.2, 2.			

- GM creates the `SOCKET_SERVER` TCP socket on whatever port number and starts listening to connection requests.	`socket()`
	`bind()`
	`listen()`				

- GM reads the values of the own IP address from local host and of the `SERVER_PORT` from the `SOCKET_SERVER`.		7.2, 3., i. and iii.			
- CD starts the transmission of the HBs. GM sends HBs via `SOCKET_HB` to the broadcast address `<network broadcast>:HB_PORT`. Each HB contains the `IP` address and `SERVER_PORT` port number.	`sendto()`	7.2, 3., iv.			

CD tries to start the communication with the Dispenser repeatedly. - CD knows LNA addresses of the IFSF devices, which are to be controlled – implicit range of LNA (subnet, node) addresses or a Station Map. It includes the Dispenser address. - GM looks to the `TCP CONNECTIONS/IFSF DEVICES` structures to find out `IP`, `PORT` address of the Dispenser. - The structures are empty till now so it is not possible to connect to dispenser.			The IFSF dispenser is still off.		

			The IFSF Dispenser starts up.
			- In case of using DHCP for IP address assigment the startup of the DHCP client is performed and it waits for the IP address dynamic assignment. In case of using the constant IP address the DHCP client will not be started and IP address is known.
			- IFSF TCP/IP Gate Module (**GM** below) creates the `TCP CONNECTIONS/IFSF DEVICES` address structures. The structures are empty now.
			- GM creates the `SOCKET_HB` UDP socket on the well-known `HB_PORT` for the reception/transmission of the HBs.
			- GM creates the `SOCKET_SERVER` TCP socket on whatever port number and starts listening to connection requests.
				`socket()`
				`bind()`
				
				`socket()`
				`bind()`
				`listen()`
					7.1
					
					7.2, 1.
					
					7.2, 2.

			- GM reads the values of the own IP address from local host and of the SERVER_PORT from the SOCKET_SERVER.		7.2, 3., i. and iii.
			- Dispenser starts the transmission of the HBs. GM sends HBs via SOCKET_HB to the broadcast address <network broadcast>:HB_PORT. Each HB contains the IP address and SERVER_PORT port number.	sendto()	7.2., 3., iv.
GM of the CD receives the HB. - GM extracts the IP address, port and LNAO from the HB. - GM updates its TCP CONNECTIONS/IFSF DEVICES structures.	recvfrom()	7.2, 3., iv. 7.2, 3., iii.	GM of the Dispenser receives the HB. - GM extracts the IP address, port and LNAO from the HB. - GM updates its TCP CONNECTIONS/IFSF DEVICES structures.	recvfrom()	7.2, 3., iv. 7.2, 3., iii.
CD tries to start the communication with the Dispenser repeatedly. - GM looks to its TCP CONNECTIONS/IFSF DEVICES structures to find out IP, PORT address of the Dispenser. - Structures are not empty now so the GM knows the IP, PORT address of the Dispenser.					
- GM creates new TCP socket (on whatever port) and connects it to the IP, PORT of the Dispenser. - The connection between the Dispenser and the CD is established now. All the next data communication (the IFSF messages) will be performed according to the existing application standards ([1], [2], …).	socket() bind() connect() send() recv()	7.2, 3., ii. 7.3.2	GM received (via the SOCKET_SERVER) and accepted the connection request sent from CD. - The new socket has been created. It represents the endpoint of the connection with CD. - The connection between the Dispenser and the CD is established now.	accept() send() recv()	7.2, 3., ii. 7.3.2