The application layer protocol: Remote Sensing Communication Protocol (RSComPro)

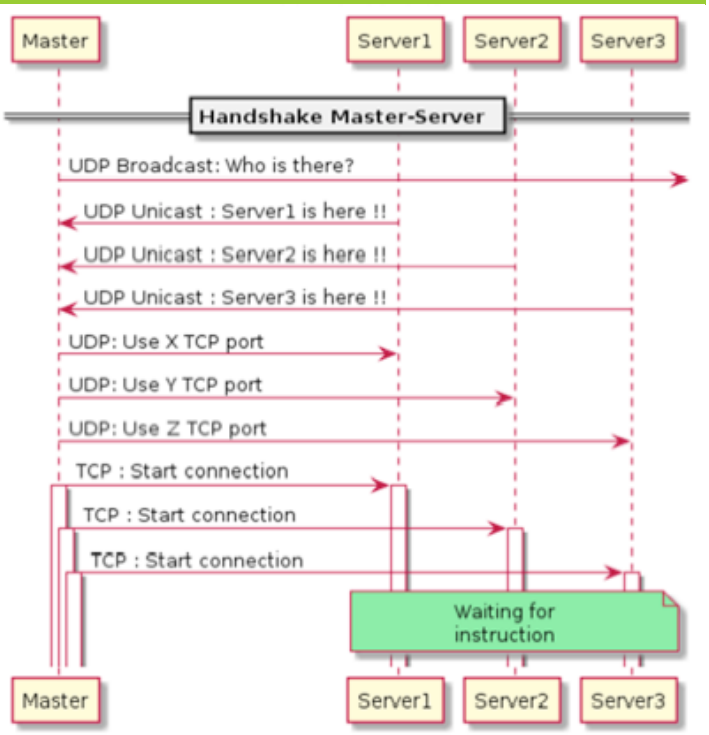
Vasiljevic, Nikola; Lea, Guillaume; Courtney, Michael; Schneemann, Jörge ; Trabucchi, Davide; Trujillo, Juan-José ; Unguran, Robert; Villa, Juan-Pablo

Publication date:
2013

Document Version
Publisher's PDF, also known as Version of record

The application layer protocol: Remote Sensing Communication Protocol (RSComPro)

DTU Vindenergi
E Rapport 2013



Nikola Vasiljević, Guillaume Lea, Michael Courtney, Jörge Schneemann, Davide Trabucchi, Juan-José Trujillo, Róbert Ungurán and Juan-Pablo Villa

DTU Wind Energy E-0017 (EN)

September 2013

DTU Vindenergi
Institut for Vindenergi

DTU external report: DTU Wind Energy E-0017 (EN)

The application layer protocol:
Remote Sensing Communication Protocol (RSComPro)
Ver. 1.0

Authors:
Nikola Vasiljević, Guillaume Lea, Michael Courtney
DTU Wind Energy

Jörge Schneemann, Davide Trabucchi, Juan-José Trujillo, Róbert Ungurán, Juan-Pablo Villa
ForWind – University of Oldenburg

Date: 1st of September 2013
ISBN number: 978-87-92896-23-0

Authors

This protocol has been developed collectively by DTU Wind Energy and ForWind University of Oldenburg. Participants on the formulation and revision of this version have been:

DTU Wind Energy

Michael Courtney

Guillaume Lea

Nikola Vasiljević §

ForWind – University of Oldenburg

Jörge Schneemann

Davide Trabucchi

Juan-José Trujillo §

Róbert Ungurán,

JuanPablo Villa

§ Editors

Acknowledgements

The development, implementation and testing of this protocol has been partly funded by the following projects

- German Joint Research Projects RAVE-OWEA No. 0327696. and RAVE-GW-Wakes No. 0325397A. These projects were funded by the German Federal Ministry for the Environment, Nature Conservation and Nuclear Safety (BMU).
- WindScanner.dk funded by the Danish Agency of Science, Technology and Innovation through gran No. 2136-08-0022
- WAUDIT is an Initial Training Network (ITN), a Marie-Curie action funded under the FP7-People program.

How to cite

In case of need of citation of this protocol you should include a text containing: "… the Remote Sensing Communication Protocol (RSComPro) Ver. 1.0 developed by DTU Wind Energy and ForWind – University of Oldenburg …"

Revision history

Revision	Comments
V 1.0	Published on 1st of September 2013

Table of contents

Introduction

The following document establishes the Remote Sensing Communication Protocol (RSComPro) to be implemented by remote sensing systems (RS system) connecting to a network coordinated by a single Master system.

The implementation of the RSComPro enables a full subordination of a RS system to a Master system. The last has as main task to control and synchronize measurement activities by all RS systems in the network. In this respect the RSComPro specifies data flow rules and formal characteristics of exchanged messages.

The first version of this protocol is the outcome of the collective work between DTU Wind Energy and ForWind University of Oldenburg. It has been developed initially for application in wind measurement by means of WindScanners (synchronized multiple lidar systems). However, it could be extended to be applied on any other type of scanning remote sensing system. Its implementation has been tested initially on systems of type WLS200S from the company Leosphere at both institutes.

The objective of this protocol is to define:

- Basic details of the software running on the RS system and the Master system
- Basic details of the network model
- Specification of handshaking process
- Basic details of error handling
- Specification of message format and syntax

Conventions and Definitions

Server: This name refers to the software running on each individual remote sensing system. This software deals with the communication and data transmission with the Master software.

Master: This name refers to the software that connects to one or more Servers and sends and receives commands and data.

LAN: Local Area Network

RS system: Remote Sensing system

TCP: Transmission Control Protocol

UDP: User Datagram Protocol

UTF: Unicode Transformation Format

VPN: Virtual Private Network

XML: Extensible Mark-up Language

Network model

In this scheme, there are 3 different entities: A **Master**, **Server(s)** (which work in a client-server architecture) and a **Network** that provides the communication layer among them.

Server

The server should have the following characteristics:

- It listens to the ephemeral UDP port = 62300
- It is able of establishing only one dedicated connection to the Master
- It is able to open a TCP port under request by the Master
- It is able to send full or chunked messages over TCP or UDP using the syntax explained in this protocol
- It is able to check integrity of sent commands over TCP or UDP. In this respect it is capable of accepting and joining chunked messages from the Master
- It is able to understand and execute instructions on the RS system sent to it over TCP or UDP
- As some commands are unicast, and others multicast, it should be able to control and discard instructions that are not valid
- It implements fully the commands in this document
- It acknowledges reception of all commands sent by the Master
- Guarantees zero data loss and data integrity of transmitted information

Master

The Master should show the following characteristics:

- It is able to scan the network for available Servers through broadcasting over UDP
- It is able to establish and control simultaneous connections to multiple Servers
- It initiates handshaking process with available Servers. It can offer the same or different port on each Server. The port range used is from 26000 to 26099
- It is responsible for keeping synchronization of the whole system
- It is able to send full or chunked messages over TCP or UDP using the syntax explained in this protocol
- It is able to check integrity of sent commands over TCP or UDP. In this respect it is capable of accepting and joining chunked messages from the Server
- To understand and execute instructions sent to it over TCP/UDP
- As some commands are unicast, and others multicast, it should be able to control and discard instructions that are not valid
- It implements fully the commands in this document
- Guarantees zero data loss and data integrity of transmitted information
- Multi-threaded software that allows different tasks to be performed at the same time (sending instructions, analyzing answers, storing data, etc.)

Network

The network that connects the different entities should present the following characteristics:

- It should work as a local network (LAN). This can be achieved by physically connecting every entity on the same LAN or by the usage of a VPN or similar, where the devices would be “virtually” placed on the same local network.
- UDP packages must be broadcastable
- No filtering/firewall should be in place for the TCP/UDP ports specified
- Preferably a Master and Server(s) should have static IP addresses

System interaction

Instruction format

The communication between the Master and the Server(s) is based on the UDP protocol for what concerns the basic commands such as those required for handshaking or stop/abort a command on emergency. On the other hand, the TCP protocol is chosen for the communication of commands, answers, data and alerts. The exchange of information is organized in packets of strings and each packet is self-contained. That means that each packet contains all the semantics needed for execution of basic tasks.

The packet format has the following characteristics:

- Applies for TCP and UDP instructions
- Follows the rules of the extensible markup language (XML) version 1.0.
- Syntax is defined in this protocol in Section "Commands over UDP" and "Commands over TCP"
- Messages are encoded in Unicode Transformation Format UTF-8
- The root element is called `packet` and has attributes which contain information of direction of information flow, traceability of data loss and system state.

```
1 <packet Client="NAME" PckNo="PckNoM.PckNoS" Cmd="CODE" Alert="CODE">
2       <xx> </xx>
3       <yy> </yy>
4       . . .
5
6       <msg></msg>
7 </packet>
```

Table 1 Definition of the XML structure of the packet

Client
Contains the name of the entity that produced the package.

Packet number (PckNo)
Contains unique identifiers created by the master (`PckNoM`) and the Server (`PckNoS`). From the Master perspective, this value identifies uniquely each packet within all the packets sent by it to any Server.. When issuing a command, the Master creates its `PckNoM` fully independently from the Server counter `PckNoS`, therefore `PckNo="PckNoM"`. In contrast a Server should respond to a command using as `PckNo="PckNoM"."PckNoS"`. This is depicted in Figure 1.

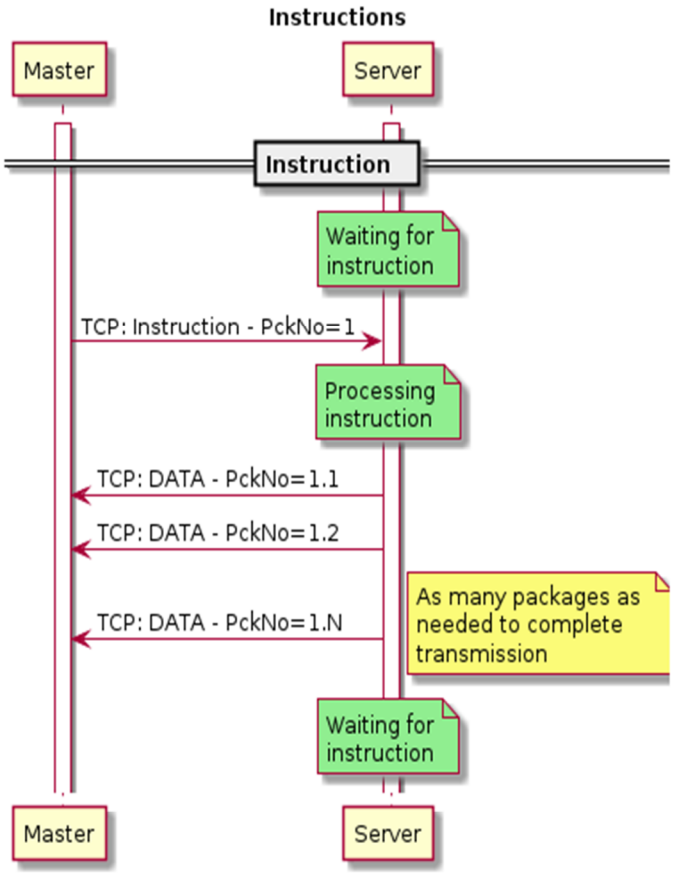


Figure 1 Sequence diagram showing construction of PckNo attribute of any packet

In this way, the original instruction from a given response can be traced back and a control can be done to check for missing packages in the answer stream.
The Packet counter in the Master should be different for UDP and TCP.

Command (Cmd)
This value identify with a code the command to which the package refers to. There are instruction and response codes which are to be used by the Master and the Servers respectively.

Alert
Either contains the **Error code** (response to a command) or the **Warning Code** (for a warning initiated from a Server (i.e. WCx -> Master)).

> **Error code** - Zero if the response of a client on the command issued by Master was done without errors, or otherwise can hold the explanation of an error that happened during the process of executing the command.
> **Warning code -** Carries the explanation of the warning sent by a Server to the Master, for example if the laser temperature is too high or low disk space.

Message (msg)

Field for transferring data. The information stored in this field depends on the command. In the following section the structure of the *msg* field relative to the commands sent by the Master and the relative answer returned by the Server is described.

Handshake procedure

Handshaking is an automated process of negotiation that dynamically sets parameters of a communication channel established between two entities before normal communication over the channel begins. An example of the initial handshake between a Master and three Servers (RS systems) can be seen in Figure 2.

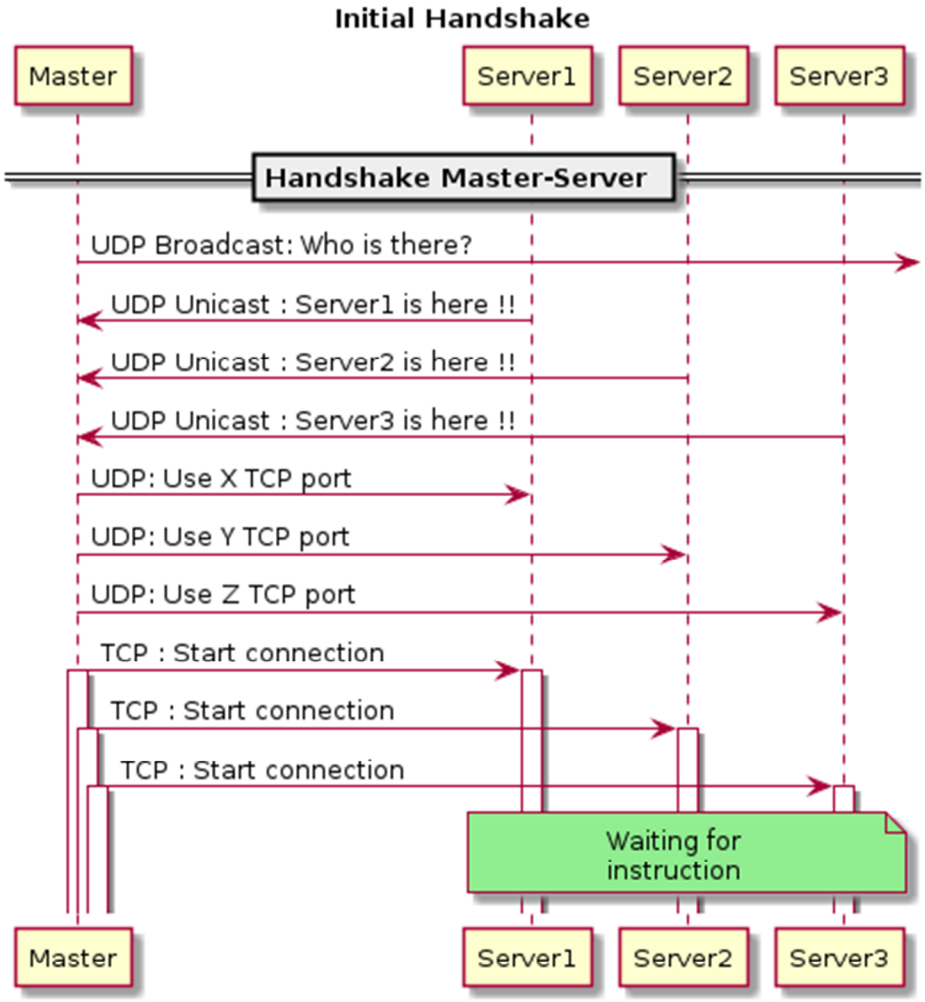


Figure 2 Sequence diagram of initial handshake between a Master and three Servers

Once the handshake is finished, the connection remains open and the Servers adopt a **Waiting for command** state.

Intructions flow

When a new command is sent, the behavior can be seen in Figure 3:

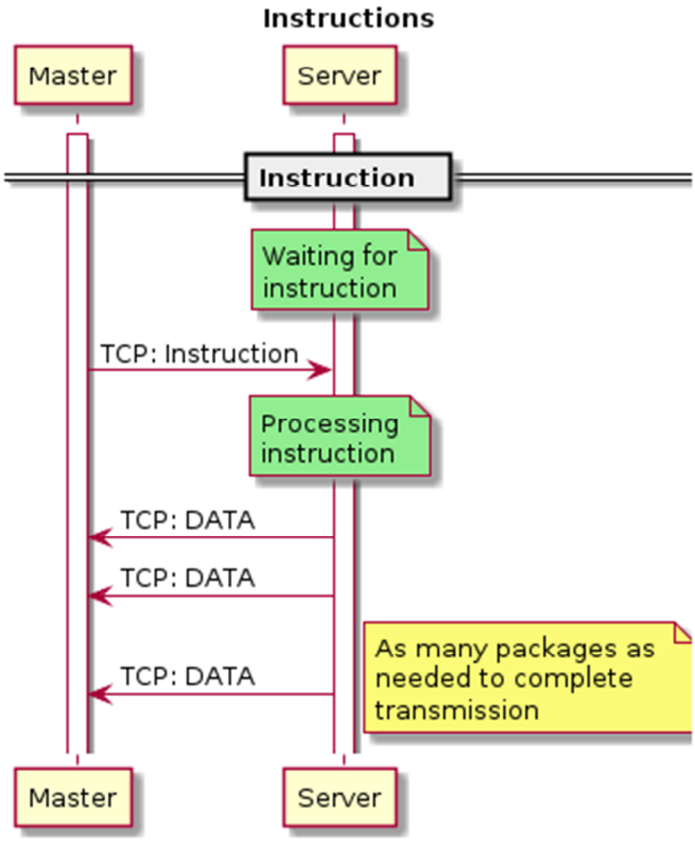


Figure 3 Sequence diagram of normal instruction flow

The Master sends the instruction and waits for the response from the Server without locking.

Error handling

The following is a typical example of a packet loss and the procedure to handle it.

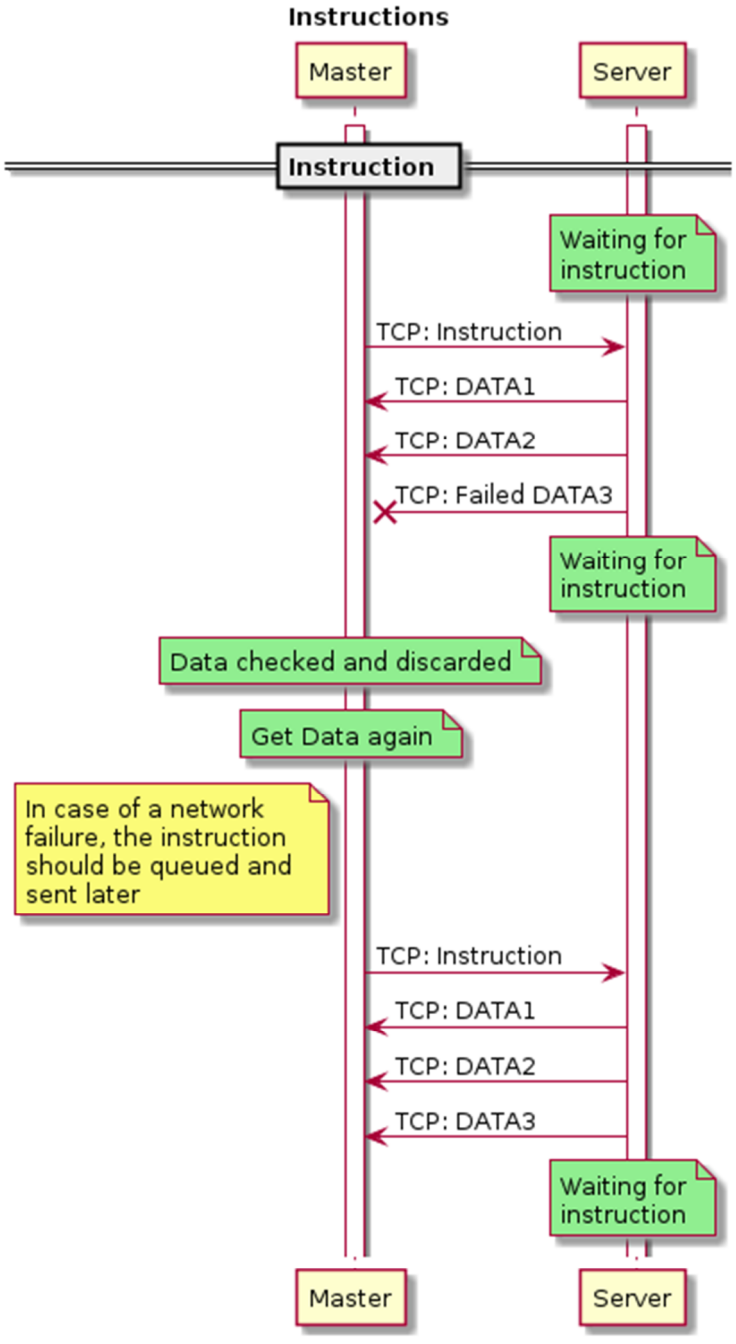


Figure 4 Sequence diagram of error handling

Case of a network failure

In the case of a Network failure during transmission of information, the protocol to follow is shown in Figure 5:

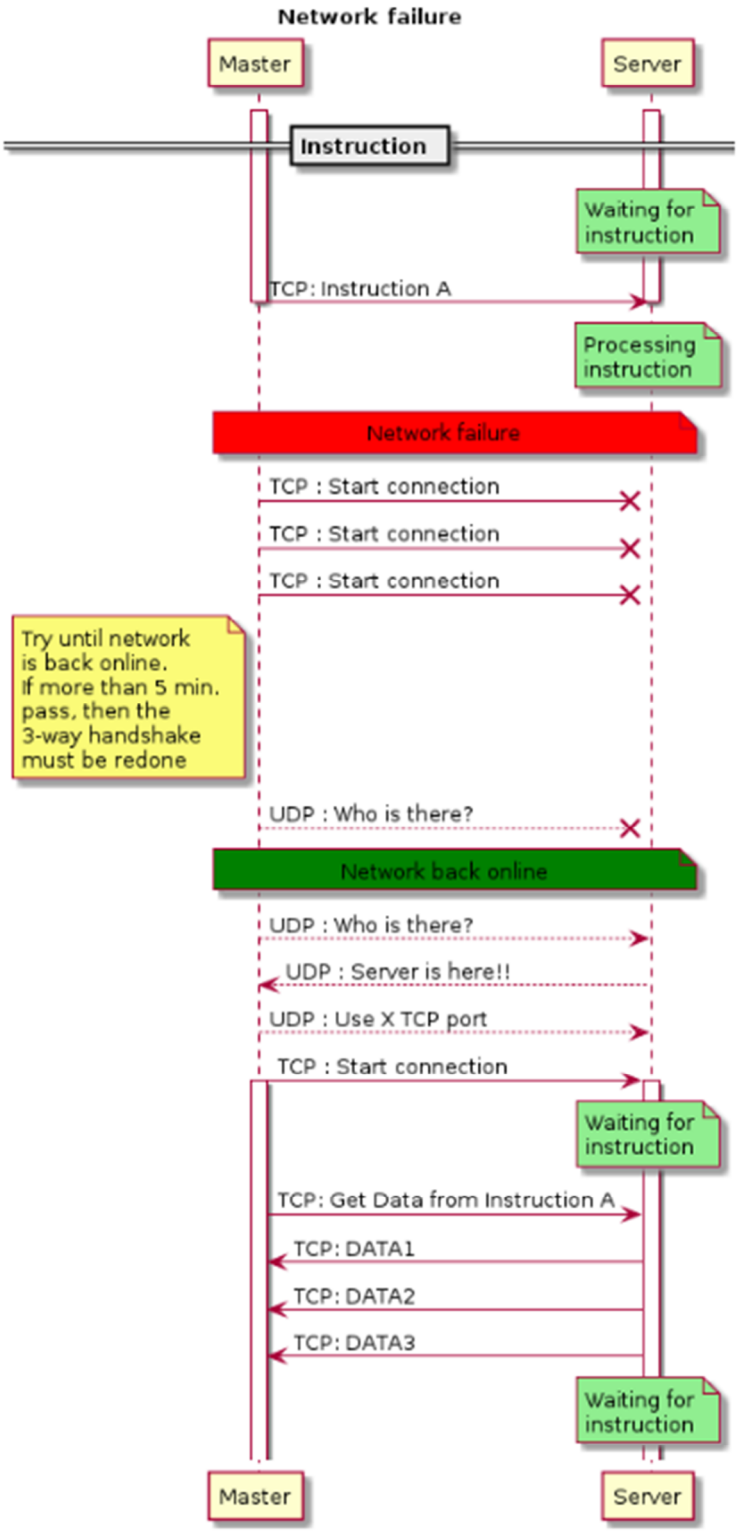


Figure 5 Sequence diagram of network failure

Unresponsive Server

In the case of an unresponsive Server, the procedure to follow is detailed in Figure 6 and Figure 7:

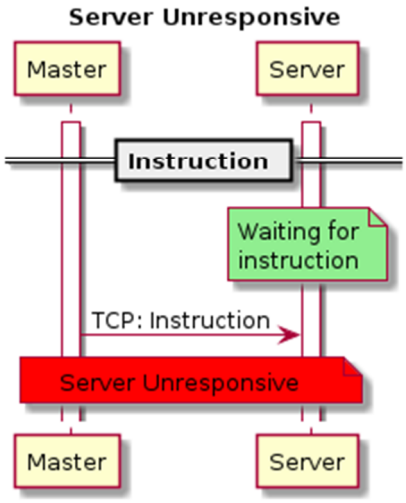


Figure 6 Sequence diagram of an unresponsive server

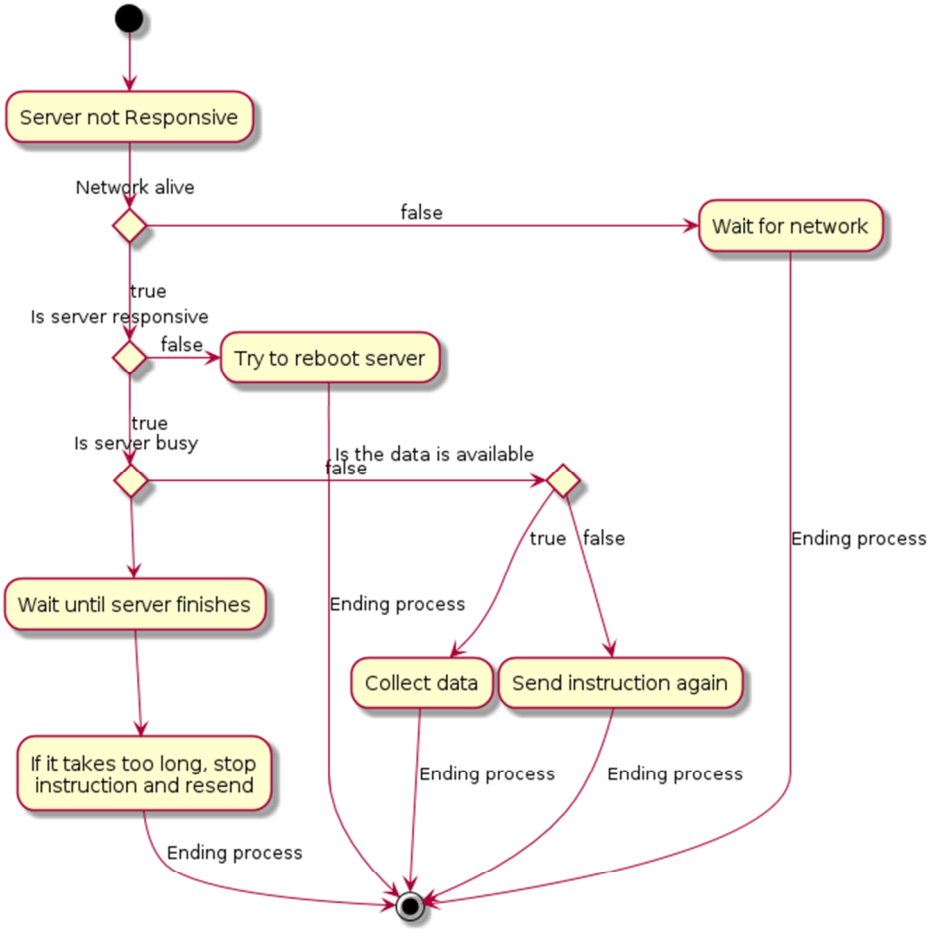


Figure 7 Activity diagram of an unresponsive Server

Wrong answer

In the case of the Master receiving an incorrect or unexpected answer from one of the Servers, the information is to be discarded and the original instruction resent (Figure 8).

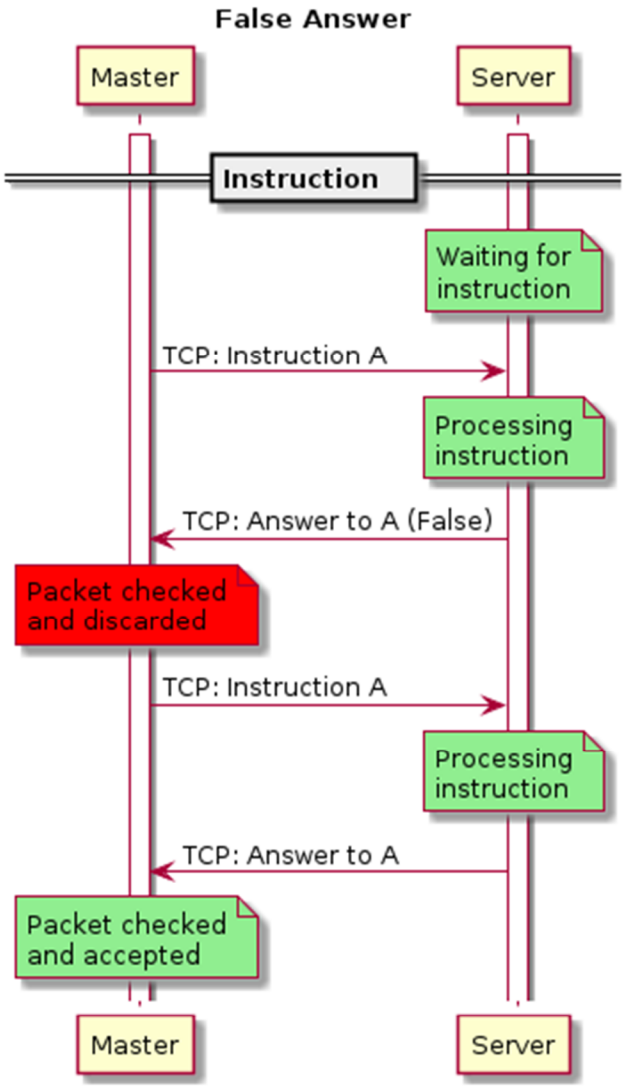


Figure 8 Sequence diagram on wrong answer from Server

Commands over UDP

Overview

The full set of commands to be sent via UDP can be seen in Table 2. All commands sent over UDP are using port number 62300.

Command	Code	Page
who is there	1100	17
abort	1200	19
unlock	1300	20
stop	1400	20
get states	1500	21
is busy	1600	22
shut down	1700	23
reset	1800	24

Table 2 Summary of UDP commands

Detailed Format of UDP packets

The UDP packets are detailed in the tables below.

Command name:	WhoIsThere?
Command code:	1100
Transport protocol:	UDP
IP routing:	Unicasting and Broadcasting
General characteristics:	
With this command (**Command 1**) user detects how many Servers are present in the network. Once a Server receives this command it responds (**Response**) back to the Master with information about its name, IP address and enquires for a TCP port. After receiving and checking the responses the Master sends back **Command 2** with the required information that the Server needs to start a TCP service (handshaking). Once the TCP connection is established the TCP commands can be sent.	
Command 1 XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="1100" Alert="0">
      <ip>Master IP address</ip>
      <port></port>
      <buffer></buffer>
      <sysid></sysid>
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SysId(?).No" Cmd="1100" Alert="0">
```

```
      <ip>Client IP address </ip>
      <port></port>
      <buffer></buffer>
      <sysid></sysid>
      <msg>Need TCP port</msg>
</packet>
```

Command 2 XML structure:

```
<packet Client="Master" PckNo="0.(No+1)" Cmd="1100" Alert="0">
      <ip>Master IP address</ip>
      <port>Available port number for Client</port>
      <buffer>Size of TCP packets in bytes</buffer>
      <sysid>System ID number (SysId(?))</sysid>
      <msg></msg>
</packet>
```

Example of the implementation of a WhoIsThere? command:

The Master with IP address 192.168.3.66 broadcasts the "WhoIsThere? command:

```
<packet Client="Master" PckNo="0.1" Cmd="1100" Alert="0">
      <ip>192.168.3.66</ip>
      <port></port>
      <buffer></buffer>
      <sysid></sysid>
      <msg></msg>
</packet>
```

Server **Košava** (IP address 192.168.3.7) replies back with the request for the TCP port number that is needed to start the process of establishing the TCP connection:

```
<packet Client="Košava" PckNo=" .1" Cmd="1100" Alert="0">
      <ip>192.168.3.7</ip>
      <port></port>
      <buffer></buffer>
      <sysid></sysid>
      <msg>Need TCP port</msg>
</packet>
```

Once the Master receives the response from **Košava** it sends the reply with necessary information that are needed for the TCP connection:

```
<packet Client="Master" PckNo="0.2" Cmd="1100" Alert="0">
      <ip>192.168.3.66</ip>
      <port>26000</port>
      <buffer>1024</buffer>
      <sysid>1</sysid>
      <msg></msg>
</packet>
```

Parameter	Description	Values possible
<ip>	IP of the sender of the message	Values within the local network
<port>	TCP port to be used for the communication	26000 / 26090
<buffer>	Size of the buffer in bytes	1024 * N (N=1 to 64)
<sysid>	System Identification number	0 / 255
<msg>	Additional Message	text

Command name:	Abort
Command code:	1200
Transport protocol:	UDP
IP routing:	Unicasting
General characteristics:	
Command **Abort** results in a sudden stop of the current operations.	
Command XML structure:	

```
1 <packet Client="Master" PckNo="0.No" Cmd="1200" Alert="0">
2       <msg></msg>
3 </packet>
```

Response XML structure:

```
1 <packet Client="Server" PckNo="SySID.No" Cmd="1200" Alert="0">
2       <msg>system locked</msg>
3 </packet>
```

Example of the implementation of an Abort command:

The Master unicasts "**Abort**" to Server **Košava**:

```
1 <packet Client="Master" PckNo="0.5" Cmd="1200" Alert="0">
2       <msg></msg>
3 </packet>
```

Sensor **Košava** acknowledges execution of the command and that it forced the sudden stop of all activities, and locked the RS system:

```
1 <packet Client="Košava" PckNo=" 1.8" Cmd="1200" Alert="0">
2       <msg>system locked</msg>
3 </packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	Unlock
Command code:	1300
Transport protocol:	UDP
IP routing:	Unicasting
General characteristics:	
Command **Unlock** unlocks a RS system for further use.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="1300" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="1300" Alert="0">
      <msg>Unlocked, system available for command</msg>
</packet>
```

Example of the implementation of an Unlock command:

The Master unicasts "**Unlock**" to Server **Košava**:

```
<packet Client="Master" PckNo="0.5" Cmd="1300" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** acknowledges execution of the command and readiness to accept commands and execute them:

```
<packet Client="Košava" PckNo=" 1.8" Cmd="1300" Alert="0">
      <msg> Unlocked, system available for command </msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	Stop
Command code:	1400
Transport protocol:	UDP
IP routing:	Unicasting
General characteristics:	
Command **Stop** results in a soft stop of the current operations.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="1400" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="1400" Alert="0">
      <msg>the current operations stopped </msg>
</packet>
```

Example of the implementation of a Stop command:

The Master unicasts "**Stop**" to Server **Košava**:

```
<packet Client="Master" PckNo="0.5" Cmd="1400" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava**acknowledges execution of the command and stop of the current measurement:

```
<packet Client="Košava" PckNo=" 1.3" Cmd="1400" Alert="0">
      <msg>the measurement stopped</msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	GetStates
Command code:	1500
Transport protocol:	UDP
IP routing:	Unicasting

General characteristics:

With the command **GetStates** the Master receives information about a RS system such as current computer time, available system RAM, available hard drive space, if it is available to perform measurements (busy element), if it is locked, the strength of the GSM signal, and the strength of the Wi-Fi signal.

Command XML structure:

```
<packet Client="Master" PckNo="0.No" Cmd="1500" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="1500" Alert="0">
      <ostime></ostime>
      <freeram></freeram>
      <freehdd></freehdd>
      <busy></busy>
      <locked></locked>
      <gsm></gsm>
      <wifi></wifi>
      <msg></msg>
```

```
10 </packet>
```

Example of the implementation of a GetStates command:

The Master unicasts "**Stop**" to Server **Košava**:

```
1 <packet Client="Master" PckNo="0.12" Cmd="1500" Alert="0">
2       <msg></msg>
3 </packet>
```

Server **Košava** replies back with all the available information, among which it indicates that it is not busy and it is unlocked:

```
1 <packet Client="Košava" PckNo=" 1.5" Cmd="1500" Alert="0">
2       <ostime>30/10/2012 16:43:00</ostime>
3       <freeram>?</freeram>
4       <freehdd>567Gb</freehdd>
5       <busy>no</busy>
6       <locked>unlocked</locked>
7       <gsm>?</gsm>
8       <wifi>?</wifi>
9       <msg></msg>
10 </packet>
```

Parameter	Description	Values possible
<ostime>	Clock time of the OS in the format DDMMYYYYThhmm	
<freeram>	Available RAM	
<freehd>	Available space on the hard-disk	
<busy>	Identifies the availability of the LIDAR Server	Integers 0 : not busy >0 : busy state code
<locked>	Identifies the status the LIDAR Server	0/1
<gsm>	Percentage of the power of the GSM signal	0 / 100 /
<wifi>	Percentage of the power of the Network signal	0 / 100 /
<msg>	Additional Message	Text

Command name:	IsBusy?
Command code:	1600
Transport protocol:	UDP
IP routing:	Unicasting
General characteristics:	
Command **IsBusy?** returns back information about the status in which a RS system is.	
Command XML structure:	

```
1 <packet Client="Master" PckNo="0.No" Cmd="1600" Alert="0">
2       <msg></msg>
```

```
3 </packet>
```

Response XML structure:

```
1 <packet Client="Server" PckNo="SySID.No" Cmd="1600" Alert="0">
2       <msg>Ready to use or Setting Instruments or Acquiring</msg>
3 </packet>
```

Example of the implementation of an IsBusy? command:

The Master unicasts "**IsBusy?**" to Server **Košava**:

```
1 <packet Client="Master" PckNo="0.5" Cmd="1600" Alert="0">
2       <msg></msg>
3 </packet>
```

Server **Košava** replies back that it is currently setting instruments (acquisition board, scanner head, pulse generator,...):

```
1 <packet Client="Košava" PckNo=" 1.3" Cmd="1600" Alert="0">
2       <msg>Setting Instruments</msg>
3 </packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	Shutdown
Command code:	1700
Transport protocol:	UDP
IP routing:	Unicasting

General characteristics:

Command **Shutdown** results in the shut down of a RS system.

Command XML structure:

```
1 <packet Client="Master" PckNo="0.No" Cmd="1700" Alert="0">
2       <msg></msg>
3 </packet>
```

Response XML structure:

```
1 <packet Client="Server" PckNo="SySID.No" Cmd="1700" Alert="0">
2       <msg>Shutting down computer in 30 seconds</msg>
3 </packet>
```

Example of the implementation of a Shutdown command:

The Master unicasts "**Shutdown**" to Server **Košava**:

```
1 <packet Client="Master" PckNo="0.5" Cmd="1700" Alert="0">
2       <msg></msg>
3 </packet>
```

Server **Košava** replies back that the shutdown will happen in 30 seconds:

```
<packet Client="Košava" PckNo=" 1.3" Cmd="1700" Alert="0">
      <msg>Shutting down computer in 30 seconds</msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	Reset
Command code:	1800
Transport protocol:	UDP
IP routing:	Unicasting
General characteristics:	
Command **Reset** results in the reset of a lidar.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="1800" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="1800" Alert="0">
      <msg>Resetting computer in 30 seconds</msg>
</packet>
```

Example of the implementation of a Reset command:

The Master unicasts "**Reset**" to Server **Košava**:

```
<packet Client="Master" PckNo="0.5" Cmd="1800" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** replies back that the reset will happen in 30 seconds:

```
<packet Client="Košava" PckNo=" 1.3" Cmd="1800" Alert="0">
      <msg>Resetting computer in 30 seconds</msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Commands over TCP

Overview

The full set of commands to be sent via TCP can be seen in Table 3. All commands sent over TCP are using port numbers from 26000 to 26099.

Table 3 Summary of TCP commands

Detailed format of TCP Packets

The TCP packets are detailed in the tables below.

Command name:	GoHome
Command code:	2100
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
When a RS system receives **GoHome** it sets moving parts to their initial positions.	
Command XML structure:	

```
1 <packet Client="Master" PckNo="0.No" Cmd="2100" Alert="0">
2       <msg></msg>
3 </packet>
```

Response XML structure:

```
1 <packet Client="Server" PckNo="SySID.No" Cmd="2100" Alert="0">
```

```
        <msg>Home Done</msg>
</packet>
```

Example of the implementation of a GoHome command:

The Master sends "**GoHome**" over TCP to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2100" Alert="0">
        <msg></msg>
</packet>
```

Server **Košava** acknowledges execution of GoHome command:

```
<packet Client="Košava" PckNo="1.3" Cmd="2100" Alert="0">
        <msg>Home Done</msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	GetGPS
Command code:	2200
Transport protocol:	TCP
IP routing:	Unicasting

General characteristics:

Command **GetGPS** results in the reply from a lidar that consists of the information provided by the internal GPS such as GPS time and date, latitude, longitude and altitude.

Command XML structure:

```
<packet Client="Master" PckNo="0.No" Cmd="2200" Alert="0">
        <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="2200" Alert="0">
        <time>time</time>
        <date>date</date>
        <lat>latitude</lat>
        <long>longitude</long>
        <alti>altitude</alti>
        <msg></msg>
</packet>
```

Example of the implementation of a GetGPS command:

The Master unicasts "**GetGPS**" to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2200" Alert="0">
        <msg></msg>
</packet>
```

Server **Košava** replies back with the information gathered from the GPS receiver:

```
<packet Client="Košava" PckNo="1.3" Cmd="2200" Alert="0">
      <time>134520.50</time>
      <date>141212</date>
      <lat>554137.8778N</lat>
      <long>120513.5359E</long>
      <alti>40.091041</alti>
      <msg></msg>
</packet>
```

Parameter	Description	Values possible
<time>	Clock time	
<date>	Date in the format DDMMYYYY	
<lat>	Latitude in UTM (North)	
<long>	Longitude in UTM (East)	
<alti>	Altitude above sea level in meters	> 0
<msg>	Additional Message	text

Command name:	GetCompass
Command code:	2300
Transport protocol:	TCP
IP routing:	Unicasting

General characteristics:

The master computer receives information from the lidar internal compass using command **GetCompass**.

Command XML structure:

```
<packet Client="Master" PckNo="0.No" Cmd="2300" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="2300" Alert="0">
      <head>heading</head>
      <pitch>pitch</pitch>
      <roll>roll</roll>
      <temp>temperature</temp>
      <msg> </msg>
</packet>
```

Example of the implementation of a GetCompass command:

The Master unicasts **GetCompass** to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2300" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** replies back with information regarding heading, pitch, roll and also temperature of the lidar:

```
<packet Client="Košava" PckNo=" 1.3" Cmd="2300" Alert="0">
      <head>98.3</head>
      <pitch>-0.6</pitch>
      <roll>177.9</roll>
      <temp>25.2</temp>
      <msg> </msg>
</packet>
```

Parameter	Description	Values possible
<head>	Heading in degrees	0 – 360
<pitch>	Pitch in degrees	0 – 360
<roll>	Roll in degrees	0 – 360
<temp>	Temperature in degrees Celsius	
<msg>	Additional Message	text

Command name:	GetConfiguration
Command code:	2400
Transport protocol:	TCP
IP routing:	Unicasting

General characteristics:

Command **GetConfiguration** returns information regarding a lidar configuration.

Command XML structure:

```
<packet Client="Master" PckNo="0.No" Cmd="2400" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="2400" Alert="0">
      <config>Configuration values</config>
      <msg></msg>
</packet>
```

Example of the implementation of a GetConfiguration command:

The Master unicasts **GetConfiguration** to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2400" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** replies back with values from the WindScanner configuration file:

```
<packet Client="Košava" PckNo="1.3" Cmd="2400" Alert="0">
      <config>[General Informations]
Version="1.0.2.4"
ID Client="RISOE 3 - WLS200S-7"
ID System="Koshava"
Localisation="RISOE"
Comments=""

[Data Paths]
Data Path="/C/Users/LEOSPHERE/Documents/Leosphere/DATA"
Pulse File Path="/C/Users/LEOSPHERE/Desktop/TOOLS/pulse_laser_400.sig"

[Devices Settings]
Digitizer.Ressource Name="PCI::INSTR0"
Digitizer.Channel="1"
Digitizer.Trigger Param.Trigger Source="External"
Digitizer.Trigger Param.Trigger Coupling="DC"
Digitizer.Trigger Param.Trigger Slope="Positive"
Digitizer.Trigger Param.Trig Level 1="3000.000000000"
Digitizer.Trigger Param.Trig Level 2="0.000000000"
Digitizer.Horizontal Param.Sampling Interval="0.000000004"
Digitizer.Horizontal Param.Delay Time="0.000000200"
Digitizer.Vertical Param.Coupling="AC 50 Ohm"
Digitizer.Vertical Param.Offset="0.000000000"
Digitizer.Vertical Param.Full Scale="0.200000000"
EDFA.COM Port="\00\00\00\04COM1"
EDFA.LD1 Current="1700"
EDFA.LD2 Current="7900"
EDFA.Version="3-Stages"
Scanner.dwDevice="0"
Compass.COM Port="\00\00\00\04COM5"
Compass.Declination="7.500000000"
Compass.Pitch angle correction="0.000000000"
Compass.Roll angle correction="-179.600000000"
Compass.Digital-compass-model="OS5X"
Compass.Temperature correction="-1.700000000"
PulseGenerator.COM Port="\00\00\00\04COM2"
PulseGenerator.DELAY="0.000000048"
PulseGenerator.LPULSE="0.000001100"
PulseGenerator.FPULSE="10000.000000000"
PulseGenerator.DAC filePath="/C/Users/LEOSPHERE/Desktop/TOOLS/pulse_400.dac"
GPS.COM Port="\00\00\00\04COM7"

[Acquisition Settings]
Pulse File Settings.Emitted Pulse Start="0"
Pulse File Settings.Emitted Pulse End="149"
Pulse File Settings.Pulse File Header Length="0"
Laser Settings.Wavelength="1543.000000000"
Laser Settings.Direction Offset="0.000000000"
Filters Settings.Homogeneity Filter="FALSE"
Filters Settings.Nb High Pass Filter Points="5"
Filters Settings.Low Pass Filter CutOff Frequency="90000000.000000000"
Filters Settings.Max Mesurable Speed="20.000000000"
Algorythm Settings.MAO Frequency="68477000.000000000"
Algorythm Settings.Nb MinPulses="1000"
Algorythm Settings.Reflected Pulse Start ="295"
Algorythm Settings.Reflected Pulse End="494"
Algorythm Settings.Alpha="2.000000000"
Algorythm Settings.FFT Window Width="128"

[Warnings Thresholds]
```

```
ID1 high="1800.000000000"
ID1 low="900.000000000"
Temperature high="45.000000000"
Temperature low="0.000000000"
PSD (fMAO)="200.000000000"
Space disk used="90.000000000"
ID2 high="7920.000000000"
ID2 low="4000.000000000"</config>
      <msg> </msg>
</packet>
```

Parameter	Description	Values possible
<config>	Configuration	text
<msg>	Additional Message	text

Command name:	GetPosition
Command code:	2600
Transport protocol:	TCP
IP routing:	Unicasting

General characteristics:

The reply on command **GetPosition** provides the current position of moving parts of a lidar.

Command XML structure:

```
<packet Client="Master" PckNo="0.No" Cmd="2600" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="2600" Alert="0">
      <azi>Azimuth position</azi>
      <ele>Elevation position</ele>
      <msg></msg>
</packet>
```

Example of the implementation of a GetPosition command:

The Master unicasts **GetPosition** to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2600" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** replies back with the scanner head azimuth and elevation position:

```
<packet Client="Košava" PckNo="1.3" Cmd="2600" Alert="0">
      <azi>22.01</azi>
      <ele>19.83</ele>
      <msg></msg>
</packet>
```


Parameter	Description	Values possible
<azi>	Azimuth in degrees	
<ele>	Elevation in degrees	
<msg>	Additional Message	text

Command name:	SetPosition
Command code:	2700
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
Command **SetPosition** results in setting of moving parts to certain positions.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="2700" Alert="0">
      <azi>Azimuth position</azi>
      <ele>Elevation position</ele>
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="2700" Alert="0">
      <msg>Position Reached</msg>
</packet>
```

Example of the implementation of a SetPosition command:

The Master unicasts **SetPosition** to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="2700" Alert="0">
      <azi>22.01</azi>
      <ele>19.83</ele>
      <msg></msg>
</packet>
```

When **Košava** reaches user defined positions it replies back about the success:

```
<packet Client="Košava" PckNo="1.3" Cmd="2700" Alert="0">
      <msg>Position Reached</msg>
</packet>
```

Parameter	Description	Values possible
<azi>	Azimuth in degrees	
<ele>	Elevation in degrees	
<msg>	Additional Message	text

Command name:	GetScenario
Command code:	2900
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
Command **GetScenario** receives the existing scenario or a list of scenarios from a lidar.	
Command XML structure:	

```
1 <packet Client="Master" PckNo="0.No" Cmd="2900" Alert="0">
2       <msg></msg>
3 </packet>
```

Response XML structure for scenario Line-Of-Sight (LOS):

```
1 <packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
2       <scn Typ="LOS" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
3             <meas Azi1="azimuth angle" Ele1="elevation angle"
Acc="accumulation time" Tm="time to move to the azimuth and elevation
angle" RG="range1;range2;. . .;rangeN">
4             </meas>
5       </scn>
6 </packet>
```

Response XML structure for scenario Plan Position Indicator (PPI):

```
1 <packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
2       <scn Typ="PPI" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
3             <meas Azi1="starting azimuth angle" Azi2="ending azimuth
angle" Ele1="elevation angle" Speed="the speed of the motion from
(Azi1,Ele1) to (Azi2,Ele2) in degrees/second" Acc="accumulation time"
RG="range1;range2;. . .;rangeN">
4             </meas>
5       </scn>
6 </packet>
```

Response XML structure for scenario Range Height Indicator (RHI):

```
1 <packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
2       <scn Typ="RHI" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
3             <meas Azi1="starting azimuth angle" Ele1="ending azimuth
angle" Ele2="elevation angle" Speed="the speed of the motion from
(Azi1,Ele1) to (Azi1,Ele2) degrees/second" Acc="accumulation time"
RG="range1;range2;. . .;rangeN">
4             </meas>
5       </scn>
6 </packet>
```

Response XML structure for scenario Doppler Beam Swinging (DBS):

```
<packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
      <scn Typ="DBS" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars">
            <meas Mod="5B or 4B" Ele1="elevation angle"
Acc="accumulation time" RG="range1;range2;. . .;rangeN">
            </meas>
      </scn>
</packet>
```

Response XML structure for scenario Velocity Azimuth Display (VAD):

```
<packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
      <scn Typ="VAD" Iter="Number of Iterations" FFTs="FFT size"
PulseL="0">
            <meas Mod="Number of points " Ele1="elevation angle"
Acc="accumulation time" RG="range1;range2;. . .;rangeN">
            </meas>
      </scn>
</packet>
```

Response XML structure for scenario Complex Trajectory (CT):

```
<packet Client="Server" PckNo="SysID.No" Cmd="2900" Alert="0">
      <scn Typ="CT" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
            <meas Azi1="azimuth angle 1" Ele1="elevation angle 1"
Acc="accumulation time 1" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2; . . .;rangeN">
            </meas>
            <meas Azi1="azimuth angle 2" Ele1="elevation angle 2"
Acc="accumulation time 2" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2; . . .;rangeN">
            </meas>
            âÀ¦
            <meas Azi1="azimuth angle N" Ele1="elevation angle N"
Acc="accumulation time N" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2;. . .;rangeN">
            </meas>
      </scn>
</packet>
```

Example of the implementation of a GetScenario command:

The Master unicasts **GetScenario** to Server **Košava** and gets back information about the existing five scenarios (LOS, PPI, RHI, DBS and CT) in one message:

```
<packet Client="Master" PckNo="0.5" Cmd="2900" Alert="0">
      <msg>Scenario Received</msg>
</packet>
```

Server **Košava** acknowledges reception of all scenarios:

```
<packet Client="Košava" PckNo="1.103" Cmd="2900" Alert="0">
      <scn Typ="LOS" Iter="100" FFTs="128" PulseL="400">
```

```
    3             <meas Azi1="45" Ele1="45" Acc="100" Tm="3000"
RG="110;120;130;140;150">
    4             </meas>
    5       </scn>
    6       <scn Typ="PPI" Iter="5" FFTs="256" PulseL="400">
    7             <meas Azi1="0" Azi2="180" Ele1="3" Speed="1.5"
Acc="500"
RG="100;200;300;400;500;600;700;800;900;1000;1100;1200;1300;1400;1500;160
0;1700;1800;1900;2000">
    8             </meas>
    9       </scn>
   10       <scn Typ="RHI" Iter="8" FFTs="64" PulseL="200">
   11             <meas Azi1="0" Ele1="0" Ele2="60" Speed="0.1"
Acc="1000" RG="100;150;200;250;300;350;400;450;500">
   12             </meas>
   13       </scn>
   14       <scn Typ="DBS" Iter="200" FFTs="128" PulseL="200">
   15             <meas Mod="5B" Ele1="80" Acc="2000"
RG="100;150;200;250;300;350;400;450;500">
   16             </meas>
   17       </scn>
   18       <scn Typ="CT" Iter="10" FFTs="128" PulseL="200">
   19             <meas Azi1="0" Ele1="0" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   20             </meas>
   21             <meas Azi1="10" Ele1="10" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   22             </meas>
   23             <meas Azi1="20" Ele1="20" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   24             </meas>
   25             <meas Azi1="30" Ele1="30" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   26             </meas>
   27             <meas Azi1="20" Ele1="20" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   28             </meas>
   29             <meas Azi1="10" Ele1="10" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   30             </meas>
   31       </scn>
   32       <msg></msg>
   33 </packet>
```

Parameter	Description	Values possible
<scn>	Scenario	
Typ	Type of scenario: LOS: Line-of-Sight PPI: Plan Position Indicator RHI: Range Height Indicator DBS: Doppler Beam Swinging VAD: Velocity Azimuth Display CT: Complex Trajectory	LOS | PPI | RHI | DBS | VAD | CT

Iter	Number of times the scenario has to be iterated	> 0
FFTs	Size of FFT	Integer : 64|128|264|512
PulseL	Pulse length (FWHM) in nano seconds	0: CW Integer : Pulsed
<meas>	Measurement setup	
Mod	Mode of DBS or VAD scenarios	5B|4B / Points respectively
Azi1	Initial or staring azimuth angle in degrees	
Azi2	Final azimuth angle in degrees	
Ele1	Initial or staring elevation angle in degrees	
Ele2	Final elevation angle in degrees	
Acc	Accumulation time in seconds	> 0
Tm	Time to move to the azimuth and elevation angle in seconds.	> 0
Speed	Speed to move from (Azi1,Ele1) to (Azi2,Ele2) in degrees per second	>0
RG	List of range gates in meters	
<msg>	Additional Message	text

Command name:	SetScenario
Command code:	3000
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
Command **SetScenarion** sets a measurement scenario or a list of measurement scenarios.	
Command XML structure for scenario Line-Of-Sight (LOS):	

```
<packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
      <scn Typ="LOS" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
            <meas Azi1="azimuth angle" Ele1="elevation angle"
Acc="accumulation time" Tm="time to move to the azimuth and elevation
angle" RG="range1;range2;âÀ¦;rangeN">
            </meas>
      </scn>
</packet>
```

Command XML structure for scenario Plan Position Indicator (PPI):

```
<packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
      <scn Typ="PPI" Iter="Number of Iterations" FFTs="FFT size"
```

```
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
    3             <meas Azi1="starting azimuth angle" Azi2="ending azimuth
angle" Ele1="elevation angle" Speed="the speed of the motion from
(Azi1,Ele1) to (Azi2,Ele1) in degrees/second" Acc="accumulation time"
RG="range1;range2;. . .;rangeN">
    4             </meas>
    5       </scn>
    6 </packet>
```

Command XML structure for scenario Range Height Indicator (RHI):

```
    1 <packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
    2       <scn Typ="RHI" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
    3             <meas Azi1="starting azimuth angle" Ele1="ending azimuth
angle" Ele2="elevation angle" Speed="the speed of the motion from
(Azi1,Ele1) to (Azi1,Ele2) degrees/second" Acc="accumulation time"
RG="range1;range2;. . .;rangeN">
    4             </meas>
    5       </scn>
    6 </packet>
```

Command XML structure for scenario Doppler Beam Swinging (DBS):

```
    1 <packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
    2       <scn Typ="DBS" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars">
    3             <meas Mod="5B or 4B" Ele1="elevation angle"
Acc="accumulation time" RG="range1;range2;. . .;rangeN">
    4             </meas>
    5       </scn>
    6 </packet>
```

Command XML structure for scenario Velocity Azimuth Display (VAD):

```
    1 <packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
    2       <scn Typ="VAD" Iter="Number of Iterations" FFTs="FFT size"
PulseL="0">
    3             <meas Mod="Number of points " Ele1="elevation angle"
Acc="accumulation time" RG="range1;range2;. . .;rangeN">
    4             </meas>
    5       </scn>
    6 </packet>
```

Command XML structure for scenario Complex Trajectory (CT):

```
    1 <packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
    2       <scn Typ="CT" Iter="Number of Iterations" FFTs="FFT size"
PulseL="Pulse length for pulsed lidars or 0 for CW lidars">
    3             <meas Azi1="azimuth angle 1" Ele1="elevation angle 1"
Acc="accumulation time 1" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2;âÀ¦;rangeN">
    4             </meas>
    5             <meas Azi1="azimuth angle 2" Ele1="elevation angle 2"
Acc="accumulation time 2" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2;âÀ¦;rangeN">
    6             </meas>
    7             âÀ¦
    8             <meas Azi1="azimuth angle N" Ele1="elevation angle N"
```

```
Acc="accumulation time N" Tm="the time to reach the position (Azi1,Ele1)"
RG="range1;range2;âÀ¦;rangeN">
    9             </meas>
   10       </scn>
   11 </packet>
```

Response XML structure:

```
    1 <packet Client="Server" PckNo="SySID.No" Cmd="2900" Alert="0">
    2       <msg>Scenario Received</msg>
    3 </packet>
```

Example of the implementation of a SetScenario command:

The Master unicasts **SetScenario** to Server **Košava** and sets five different scenarios (LOS, PPI, RHI, DBS and CT) in one goal:

```
    1 <packet Client="Master" PckNo="0.5" Cmd="3000" Alert="0">
    2       <scn Typ="LOS" Iter="100" FFTs="128" PulseL="400">
    3             <meas Azi1="45" Ele1="45" Acc="100" Tm="3000"
RG="110;120;130;140;150">
    4             </meas>
    5       </scn>
    6       <scn Typ="PPI" Iter="5" FFTs="256" PulseL="400">
    7             <meas Azi1="0" Azi2="180" Ele1="3" Speed="1.5"
Acc="500"
RG="100;200;300;400;500;600;700;800;900;1000;1100;1200;1300;1400;1500;160
0;1700;1800;1900;2000">
    8             </meas>
    9       </scn>
   10       <scn Typ="RHI" Iter="8" FFTs="64" PulseL="200">
   11             <meas Azi1="0" Ele1="0" Ele2="60" Speed="0.1"
Acc="1000" RG="100;150;200;250;300;350;400;450;500">
   12             </meas>
   13       </scn>
   14       <scn Typ="DBS" Iter="200" FFTs="128" PulseL="200">
   15             <meas Mod="5B" Ele1="80" Acc="2000"
RG="100;150;200;250;300;350;400;450;500">
   16             </meas>
   17       </scn>
   18       <scn Typ="CT" Iter="10" FFTs="128" PulseL="200">
   19             <meas Azi1="0" Ele1="0" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   20             </meas>
   21             <meas Azi1="10" Ele1="10" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   22             </meas>
   23             <meas Azi1="20" Ele1="20" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   24             </meas>
   25             <meas Azi1="30" Ele1="30" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   26             </meas>
   27             <meas Azi1="20" Ele1="20" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   28             </meas>
   29             <meas Azi1="10" Ele1="10" Acc="2000" Tm="1000"
RG="111;222;333;444;555">
   30             </meas>
```

```
31        </scn>
32        <msg></msg>
33 </packet>
```

Server **Košava** acknowledges reception of all scenarios:

```
1 <packet Client="Košava" PckNo="1.103" Cmd="3000" Alert="0">
2       <msg>Scenario Received</msg>
3 </packet>
```

Parameter	Description	Values possible
<scn>	Scenario	
Typ	Type of scenario: LOS: Line-of-Sight PPI: Plan Position Indicator RHI: Range Height Indicator DBS: Doppler Beam Swinging VAD: Velocity Azimuth Display CT: Complex Trajectory	LOS | PPI | RHI | DBS | VAD | CT
Iter	Number of times the scenario has to be iterated	> 0
FFTs	Size of FFT	Integer : 64|128|264|512
PulseL	Pulse length (FWHM) in nano seconds	0: CW Integer : Pulsed
<meas>	Measurement setup	
Mod	Mode of DBS or VAD scenarios	5B|4B / Points respectively
Azi1	Initial or staring azimuth angle in degrees	
Azi2	Final azimuth angle in degrees	
Ele1	Initial or staring elevation angle in degrees	
Ele2	Final elevation angle in degrees	
Acc	Accumulation time in seconds	> 0
Tm	Time to move to the azimuth and elevetaion angle in seconds.	> 0
Speed	Speed to move from (Azi1,Ele1) to (Azi2,Ele2) in degrees per second	>0
RG	List of range gates in meters	
<msg>	Additional Message	text
Parameter	Description	Values possible
<scn>	Scenario	

<meas>	Measurement setup	
<msg>	Additional Message	text

Command name:	Measure
Command code:	3100
Transport protocol:	TCP
IP routing:	Unicasting, Multicasting and Broadcasting
General characteristics:	
Command **Measure** sends the time when measurement should start.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="3100" Alert="0">
      <stime>HH:MM:SS</stime>
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="3100" Alert="0">
      <msg>Measurement Started</msg>
</packet>
```

Example of the implementation of a Measure command:

The master computer unicast "**Measure**" consisting of the time a scenario or the list of scenarios should start on the WindScanner **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="3100" Alert="0">
      <msg>03:40:00</msg>
</packet>
```

Once the time has been reached and measurements started WindScanner **Košava** responds back:

```
<packet Client="Košava" PckNo="1.3" Cmd="3100" Alert="0">
      <msg>Measurement Started</msg>
</packet>
```

Parameter	Description	Values possible
stime	UTC time of lidar system when a scenario should begin. Format HH:mm:ss.fff	
<msg>	Additional Message	text

Command name:	GetData
Command code:	3200
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
There is only response to command **GetData** and it results in the instantaneous data stream of measurements, at the moment they are done they are sent to the master computer.	
Command XML structure:	
Not defined yet!	
Response XML structure:	

```
 1 <packet Client="Server" PckNo="SySID.No" Cmd="3100" Alert="0">
 2       <points Nb="No of points in message">
 3             <point Id="Measurement Point Id" ScnId="Scenarion Id"
Tstamp="YYYY/MM/DD HH:MM:SS.mmm" Values="azimuth; elevation; range1; radial
speed 1; CNR 1; Dispersion 1; range 2; radial speed 2; CNR 2; Dispersion
2;âÀ¦; range N; radial speed N; CNR N; Dispersion N">
 4             </point>
 5 . . .
 6             <point Id="Measurement Point Id" ScnId="Scenario Id"
Tstamp="YYYY/MM/DD HH:MM:SS.mmm" Values="azimuth; elevation; range1; radial
speed 1; CNR 1; Dispersion 1; range 2; radial speed 2; CNR 2; Dispersion
2;âÀ¦; range N; radial speed N; CNR N; Dispersion N">
 7             </point>
 8       </points>
 9  <msg></msg>
10 </packet>
```

Example of the implementation of a GetData command:
Server **Košava** streams data during the measurement scenario 0 that consists of only one measurement point, and currently the same measurement point has been measured for the 9th time:

```
 1 <packet Client="Košava" PckNo="1.3" Cmd="3200" Alert="0">
 2       <points Nb="1">
 3             <point Id="9"  ScnId="0" Tstamp="2012/12/14
13:53:51.519" Values="10.000;10.000;111;-8.399;-16.553;1.268;222;-9.636;-
15.279;0.963;333;-8.919;-14.507;1.076;444;-10.244;-14.624;1.683;555;-
10.181;-14.376;0.844">
 4             </point>
 5       </points>
 6  <msg></msg>
 7 </packet>
```

Parameter	Description	Values possible
<points>	UTC time of lidar system when a scenario should begin. Format	

	HH:MM:SS.fff	
Nb		
<point>		
Id	Measurement Point Id	
ScnId	Scenario Id	
Tstamp	YYYY/MM/DD HH:MM:SS.fff	
Values	azimuth; elevation; range1; radial speed 1; CNR 1; Dispersion 1; range 2; radial speed 2; CNR 2; Dispersion 2; . . .; range N; radial speed N; CNR N; Dispersion N	
<msg>	Additional Message	text

Command name:	Wipe
Command code:	3300
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
Command **Wipe** executes cleaning procedure of the optical component(s).	
Command XML structure:	

```
1 <packet Client="Master" PckNo="0.No" Cmd="3300" Alert="0">
2       <msg></msg>
3 </packet>
```

Response XML structure:

```
1 <packet Client="Server" PckNo="SySID.No" Cmd="3300" Alert="0">
2       <msg>Wipe Done</msg>
3 </packet>
```

Example of the implementation of a Wipe command:

The Master unicasts **Wipe** to Server **Košava**:

```
1 <packet Client="Master" PckNo="0.1" Cmd="3300" Alert="0">
2       <msg></msg>
3 </packet>
```

Server **Košava** acknowledges execution of Wipe command:

```
1 <packet Client="Košava" PckNo="1.3" Cmd="3300" Alert="0">
2       <msg>Wipe Done</msg>
3 </packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text

Command name:	GetCapabilities
Command code:	3400
Transport protocol:	TCP
IP routing:	Unicasting
General characteristics:	
Command **GetCapabilities** retrieves the capabilities of a lidar.	
Command XML structure:	

```
<packet Client="Master" PckNo="0.No" Cmd="3400" Alert="0">
      <msg></msg>
</packet>
```

Response XML structure:

```
<packet Client="Server" PckNo="SySID.No" Cmd="3400" Alert="0">
      <msg>Everything is possible</msg>
</packet>
```

Example of the implementation of a GetCapabilities command:

The Master unicasts **GetCapabilities** over TCP to Server **Košava**:

```
<packet Client="Master" PckNo="0.1" Cmd="3400" Alert="0">
      <msg></msg>
</packet>
```

Server **Košava** replies back with values:

```
<packet Client="Košava" PckNo="1.3" Cmd="3400" Alert="0">
      <msg>Everything is possible</msg>
</packet>
```

Parameter	Description	Values possible
<msg>	Additional Message	text