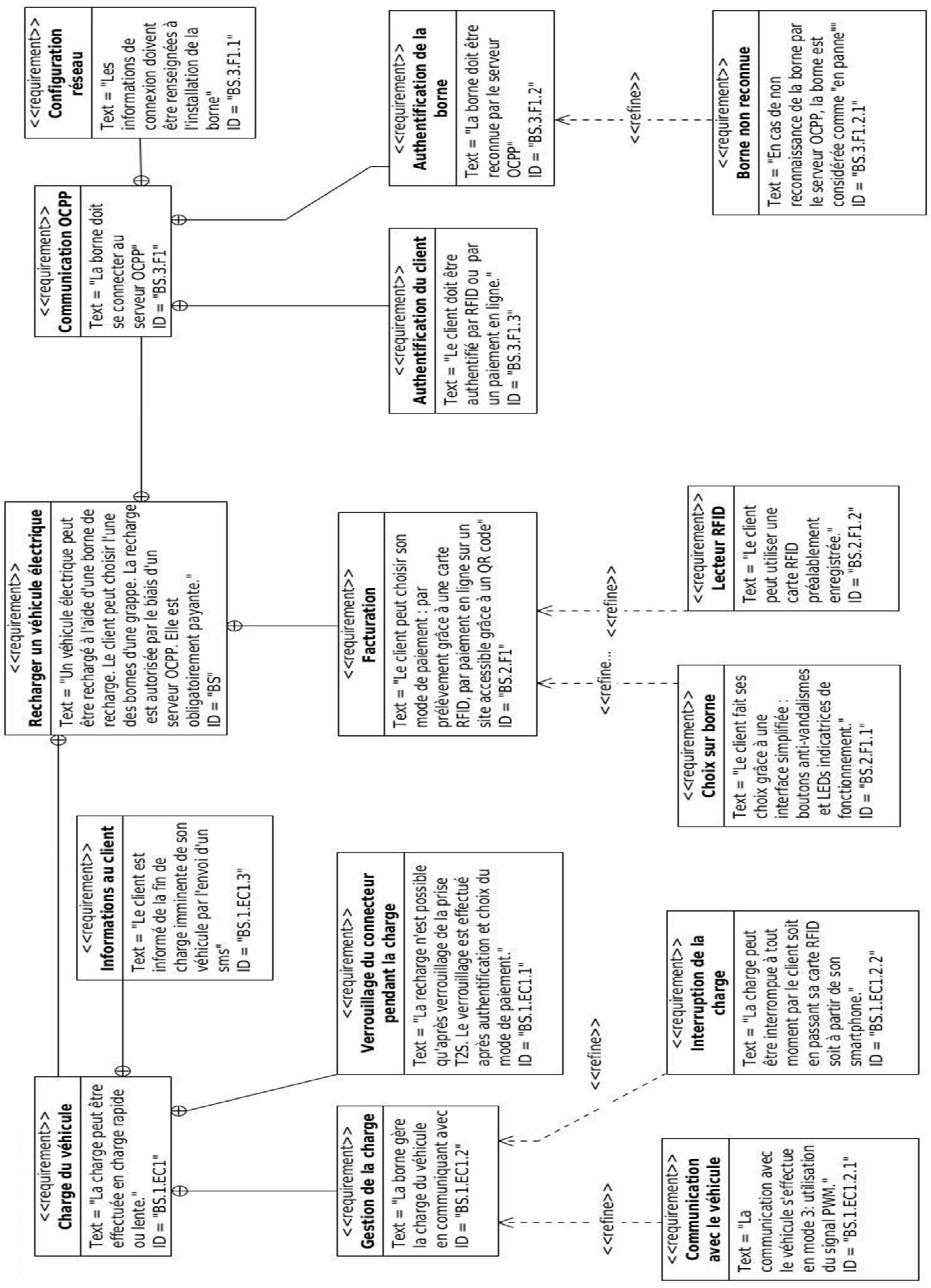
DOCUMENTATION

DOCUMENTATION PP1 : Diagramme des exigences



<<requirement>> Configuration réseau
Text = "Les informations de connexion doivent être renseignées à l'installation de la borne" ID = "BS.3.F1.1"
<<requirement>> Authentification de la borne
Text = "La borne doit être reconnue par le serveur OCPP" ID = "BS.3.F1.2"
<<requirement>> Borne non reconnue
Text = "En cas de non reconnaissance de la borne par le serveur OCPP, la borne est considérée comme "en panne"" ID = "BS.3.F1.2.1"
<<refine>>
<<requirement>> Communication OCPP
Text = "La borne doit se connecter au serveur OCPP" ID = "BS.3.F1"
<<requirement>> Authentification du client
Text = "Le client doit être authentifié par RFID ou par un paiement en ligne." ID = "BS.3.F1.3"
<<requirement>> Recharger un véhicule électrique
Text = "Un véhicule électrique peut être rechargé à l'aide d'une borne de recharge. Le client peut choisir l'une des bornes d'une grappe. La recharge est autorisée par le biais d'un serveur OCPP. Elle est obligatoirement payante." ID = "BS"
<<requirement>> Facturation
Text = "Le client peut choisir son mode de paiement : par prélèvement grâce à une carte RFID, par paiement en ligne sur un site accessible grâce à un QR code" ID = "BS.2.F1"
<<requirement>> Lecteur RFID
Text = "Le client peut utiliser une carte RFID préalablement enregistrée." ID = "BS.2.F1.2"
<<refine>>
<<refine...
<<requirement>> Choix sur borne
Text = "Le client fait ses choix grâce à une interface simplifiée : boutons anti-vandalismes et LEDs indicatrices de fonctionnement." ID = "BS.2.F1.1"
<<requirement>> Informations au client
Text = "Le client est informé de la fin de charge imminente de son véhicule par l'envoi d'un sms" ID = "BS.1.EC1.3"
<<requirement>> Verrouillage du connecteur pendant la charge
Text = "La recharge n'est possible qu'après verrouillage de la prise T2S. Le verrouillage est effectué après authentification et choix du mode de paiement." ID = "BS.1.EC1.1"
<<requirement>> Interruption de la charge
Text = "La charge peut être interrompue à tout moment par le client soit en passant sa carte RFID soit à partir de son smartphone." ID = "BS.1.EC1.2.2"
<<refine>>
<<requirement>> Charge du véhicule
Text = "La charge peut être effectuée en charge rapide ou lente." ID = "BS.1.EC1"
<<requirement>> Gestion de la charge
Text = "La borne gère la charge du véhicule en communiquant avec" ID = "BS.1.EC1.2"
<<refine>>
<<requirement>> Communication avec le véhicule
Text = "La communication avec le véhicule s'effectue en mode 3: utilisation du signal PWM." ID = "BS.1.EC1.2.1"

DOCUMENTATION PP2 : Lecteurs RFID

INVEO

RFID IND Modbus-Mif

General features

The reader is equipped with an RS485 port supporting Modbus RTU protocol and a USB port used for configuration and testing of the module.
The device has two relay outputs and two inputs.

Technical data:
Supply voltage:12-24VDC
Power supply: 40mA (12V)

Transponders:
Supported transponder standard: Mifare
Carrier frequency: 13,56 MHz
Reading distance to 10cm (depending on the type of transponder used)

Communication:
1 RS485 port – modbus RTU
1 USB port to configuration

Inputs/Outputs
2 relay outputs 1A@30VDC
2 inputs

Enclosure:
IP Rating: IP65

PEPPERL+FUCHS

IPH-FP-V1

Technical data		
General specifications		
Operating frequency		125 kHz
Transfer rate		2 kBit/s
Sensing range		
Read distance		0 ... 100 mm
Write distance		0 ... 80 mm
Width		max. 80 mm
Operating distance		maximum: 100 mm
UL File Number		E87056
Functional safety related parameters		
$MTTF_d$		710 a
Mission Time (T_M)		10 a
Diagnostic Coverage (DC)		0 %
Indicators/operating means		
LED green/yellow		green: power on green flashing: read/write attempt performed yellow: data carrier detected
Electrical specifications		
Power consumption	P_0	≤ 1.2 W
Supply		from the IDENTControl
Ambient conditions		
Ambient temperature		-25 ... 70 °C (-13 ... 158 °F)
Storage temperature		-40 ... 85 °C (-40 ... 185 °F)
Mechanical specifications		
Degree of protection		IP67
Connection		M12 x 1 connector
Material		
Housing		PBT
Base		diecast aluminum
Encapsulation compound		Fermadur
Installation		
Distance between two heads		Multiplex on: ≥ 100 mm Multiplex off: ≥ 550 mm
Mass		approx. 380 g
Compliance with standards and directives		
Directive conformity		
R&TTE Directive 1995/5/EC		EN 301489-1 V1.8.1 (2008-04), EN 301489-3 V1.4.1 (2002-08), EN 300330-2 V1.3.1 (2006-04), EN 60950-1:2006

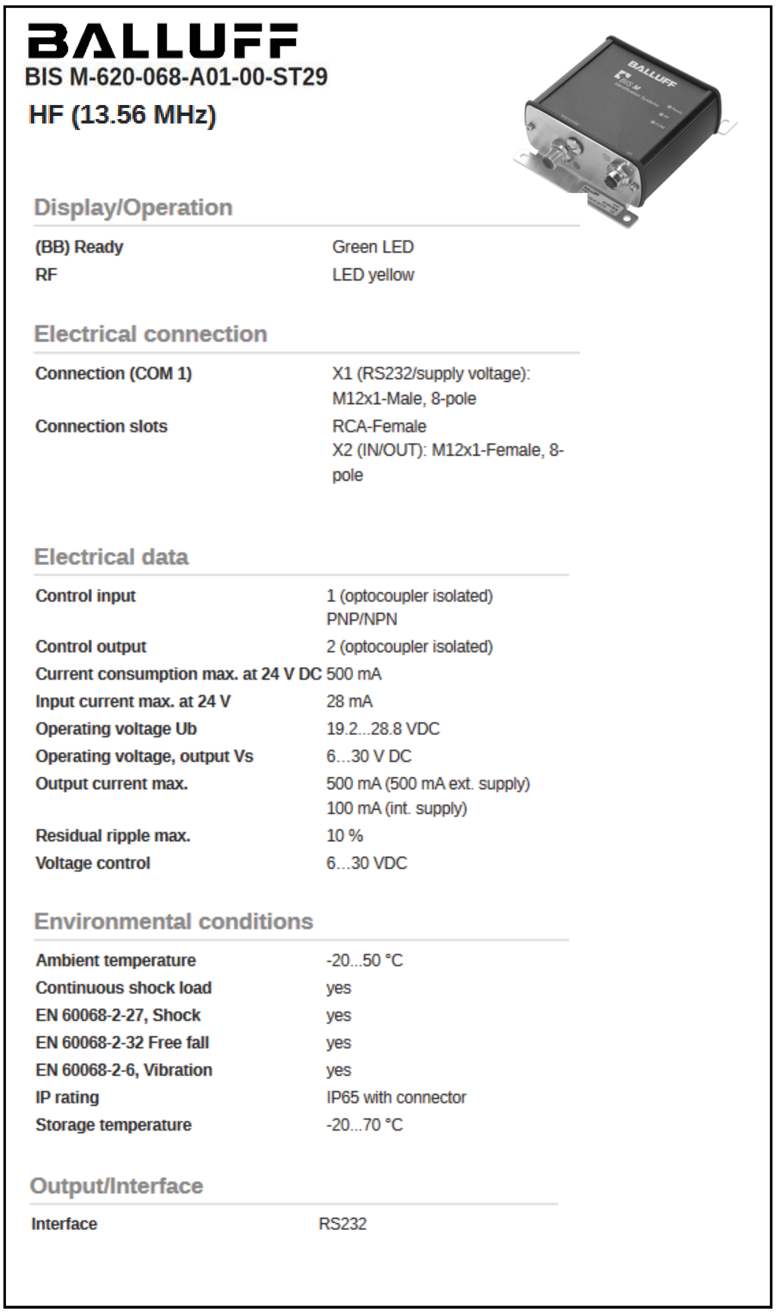
Telemecanique OsiSense XG series Smart Antenna

Characteristics		XGCS4901201 - format 40	XGCS8901201 - format 80	XGCS490B201	XGCS49LB201
Temperature	Operation	-25…+70°C (-13…158°F)		-40…+70°C (-40…158°F)	
	Storage	-40…+85 °C (-40…+185°F)		-40…+85 °C (-40…+185°F)	
Degree of protection		IP65 in accordance with IEC60529			IP69K on front faceIP65 on back part
Vibration resistance EN 60068.2.27 EN 60068.2.6		2 mm (0.078 in) from 5 to 29.5 Hz / 7 g (7 gn) from 29.5 to 150 Hz 30 g (30 gn) / 11 ms			
Resistance to mechanical shocks		IK04 according to EN 50102		IK04 according to EN 50102	
Standards / Certifications		CE, cULus, EN 300330-1/2, EN 301489-01/03, FCC Part 15 IC			
Immunity to disturbances		Resistance to electrostatic discharges, radiated electromagnetic fields, fast transients, electrical surges, conducted and induced interference and power frequency magnetic field according to IEC 61000/EN 55022.			
Unit dimensions		40x40x15 mm (1.57x1.57x0.59 in)	80x80x26 mm (3.15x3.15x1.02 in)	40x40x15 mm (1.57x1.57x0.59 in) Mounting ø22 mm (0.87 in)	
RFID frequency		13.56 MHz			
Type of associated tags		Standardized ISO 15693 and ISO 14443 tags Automatic detection of the tag type			
Nominal sensing distance (according to the associated tag)		18…70 mm (0.70…2.75 in)	20…100 mm (0.78…3.94 in)	10…70 mm (0.39…2.75 in)	
Nominal power supply		24 Vdc PELV			
Power supply voltage limits		19.2…29 V ripple included			
Power consumption		< 60 mA			
Serial links	Type	RS485			
	Protocol	Modbus RTU			
	Speed	9600…115 200 Bauds: Automatic detection			
Display		1 dual color LED for network communication 1 dual color LED for RFID communication (Tag present, Smart Antenna/tag dialog)			
Lights		-			2 Multicolor lights (7 colors)
Conne		5-way male M12 connector for connection to the communication network and power supply			
Tightening torque for the mounting		< 1 Nm (8.85 lbf-in)	< 3 Nm (26.55 lbf-in)	< 2.2 Nm (19.5 lbf-in)	

PEPPERL+FUCHS IPH-FP-V1

Technical data

General specifications		
Operating frequency		13.56 MHz
Transfer rate		26 kBit/s
Sensing range		
Read distance		0 ... 130 mm
Write distance		0 ... 130 mm
Width		max. 100 mm
UL File Number		E87056
Functional safety related parameters		
$MTTF_d$		680 a
Mission Time (T_M)		10 a
Diagnostic Coverage (DC)		0 %
Indicators/operating means		
LED red/green		Green: power on Flashing green: IO-Link communication Flashing red/green: IO-Link communication interrupted
LED blue/yellow		Blue: Write/read attempt performed Yellow: Read/write tag detected
Electrical specifications		
Rated operating voltage	U_e	20 ... 30 V DC , ripple 10 $\%_{SS}$
Power consumption	P_0	≤ 2 W
Interface		
Interface type		IO-Link
Protocol		IO-Link V1.1
Cycle time		min. 4 ms
Mode		COM 3 (230.4 kBaud)
Process data witdh		32 Byte
SIO mode support		no
Directive conformity		
Electromagnetic compatibility		
Directive 2014/30/EU		EN 61000-6-2:2005 EN 61000-6-4:2007
Radio and telecommunication terminal equipment		
Directive 2014/53/EU		EN 301489-1 V1.9.2:2011 EN 301489-3 V1.6.1:2013 EN 300330 V2.1.1:2017 EN 62368-1:2014+AC:2015 EN 50364:2010



BALLUFF

BIS M-620-068-A01-00-ST29

HF (13.56 MHz)

Display/Operation

(BB) Ready	Green LED
RF	LED yellow

Electrical connection

Connection (COM 1)	X1 (RS232/supply voltage): M12x1-Male, 8-pole
Connection slots	RCA-Female X2 (IN/OUT): M12x1-Female, 8-pole

Electrical data

Control input	1 (optocoupler isolated) PNP/NPN
Control output	2 (optocoupler isolated)
Current consumption max. at 24 V DC	500 mA
Input current max. at 24 V	28 mA
Operating voltage Ub	19.2...28.8 VDC
Operating voltage, output Vs	6...30 V DC
Output current max.	500 mA (500 mA ext. supply) 100 mA (int. supply)
Residual ripple max.	10 %
Voltage control	6...30 VDC

Environmental conditions

Ambient temperature	-20...50 °C
Continuous shock load	yes
EN 60068-2-27, Shock	yes
EN 60068-2-32 Free fall	yes
EN 60068-2-6, Vibration	yes
IP rating	IP65 with connector
Storage temperature	-20...70 °C

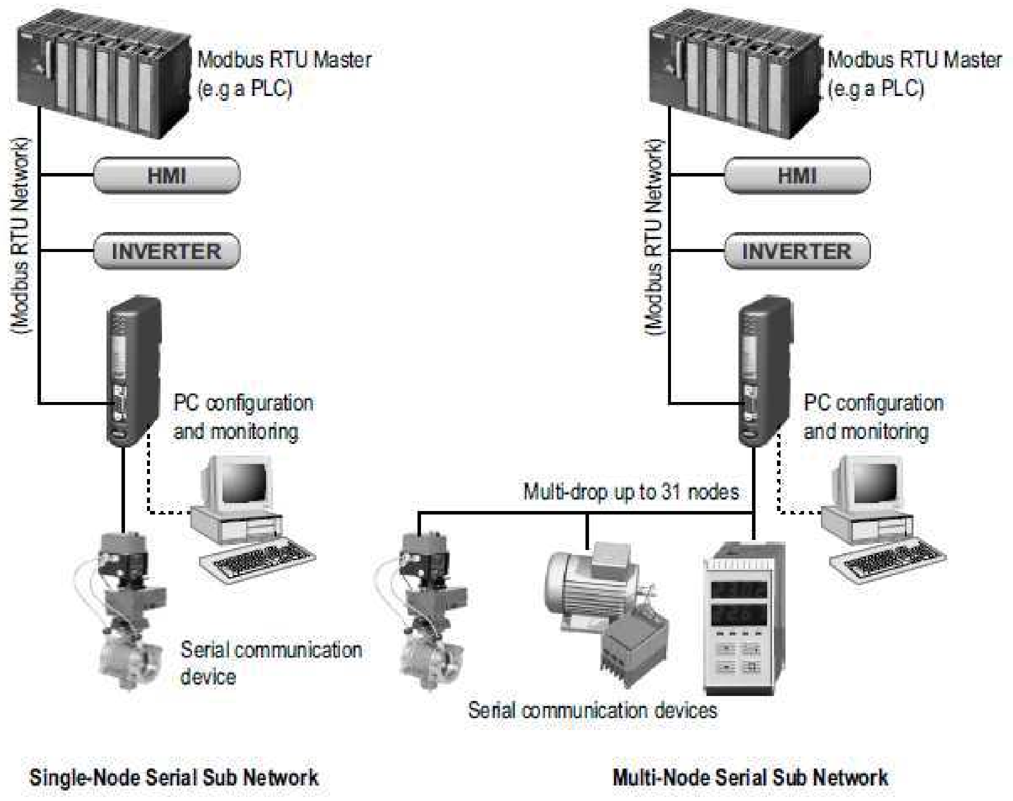
Output/Interface

Interface	RS232

DOCUMENTATION PP3 : AnyBus Communicator

About the Anybus Communicator for Modbus RTU

The Anybus Communicator for Modbus RTU acts as a gateway between virtually any serial application protocol and a Modbus RTU-based network. Integration of industrial devices is enabled without loss of functionality, control and reliability, both when retro-fitting to existing equipment as well as when setting up new installations.


Modbus RTU Master
(e.g a PLC)
(Modbus RTU Network)
HMI
INVERTER
PC configuration and monitoring
Serial communication device
Single-Node Serial Sub Network
Modbus RTU Master
(e.g a PLC)
(Modbus RTU Network)
HMI
INVERTER
PC configuration and monitoring
Multi-drop up to 31 nodes
Serial communication devices
Multi-Node Serial Sub Network

Subnetwork

The gateway can address up to 31 nodes, and supports the following physical standards:

- RS-232
- RS-422
- RS-485

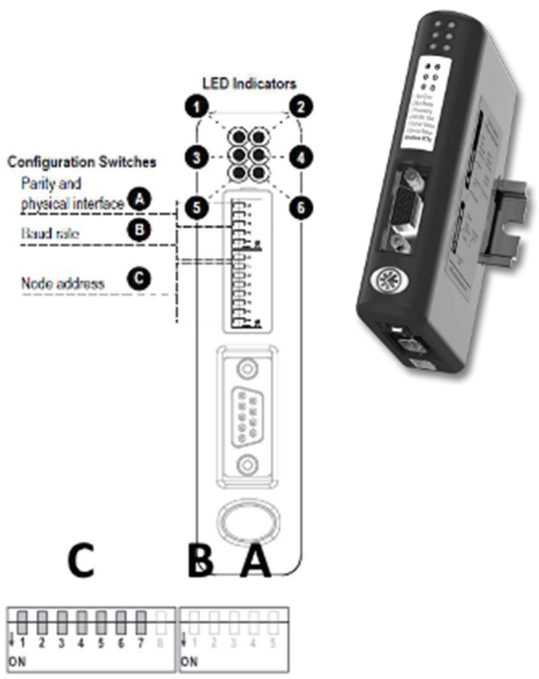
Modbus RTU Interface

Modbus RTU connectivity is provided through patented Anybus technology; a proven industrial communication solution used all over the world by leading manufacturers of industrial automation products.

- Galvanically isolated bus interface
- Coil and Register access
- RS-232 or RS-485 operation
- On-board configuration switches
- 1200... 57600bps operation

Passerelle ModBus

Configuration Switches



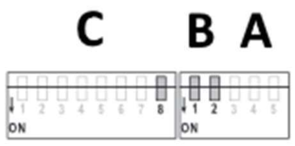
The configuration switches determines the basic communication settings for the Modbus interface. Normally, these switches are covered by a plastic hatch. When removing the hatch, avoid touching the circuit boards and components. If tools are used to open the hatch, use caution.

Note that these settings cannot be changed during runtime, i.e. the gateway must be restarted in order for any changes to have effect.

Node Address

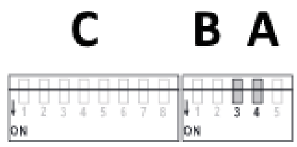
Node Address	Sw. 1	Sw. 2	Sw. 3	Sw. 4	Sw. 5	Sw. 6	Sw. 7
(reserved)	OFF	OFF	OFF	OFF	OFF	OFF	OFF
1	OFF	OFF	OFF	OFF	OFF	OFF	ON
2	OFF	OFF	OFF	OFF	OFF	ON	OF
...	...	...	...	...	...	...	...
126	ON	ON	ON	ON	ON	ON	OFF
127	ON	ON	ON	ON	ON	ON	ON

Baudrate Configuration



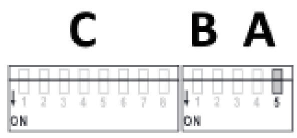
Baudrate	Sw. 8	Sw. 1	Sw. 2
(reserved)	OFF	OFF	OFF
1200 bps	OFF	OFF	ON
2400 bps	OFF	ON	OFF
4800 bps	OFF	ON	ON
9600 bps	ON	OFF	OFF
19200 bps (standard)	ON	OFF	ON
38400 bps	ON	ON	OFF
57600 bps	ON	ON	ON

Parity & Stop Bits



Parity	Sw. 3	Sw. 4
(reserved)	OFF	OFF
No parity, 2 stop bits	OFF	ON
Even parity, 1 stop bit	ON	OFF
Odd parity, 1 stop bit	ON	ON

Physical Interface



Interface Type	Sw. 5
RS-485	OFF
RS-232	ON

DOCUMENTATION PP4 : « *ModBus over Serial link* »

MODBUS Data Link Layer

MODBUS Master / Slaves protocol principle

The MODBUS Serial Line protocol is a Master-Slaves protocol. Only one master (at the same time) is connected to the bus, and one or several (247 maximum number) slaves nodes are also connected to the same serial bus. A MODBUS communication is always initiated by the master. The slave nodes will never transmit data without receiving a request from the master node. The slave nodes will never communicate with each other. The master node initiates only one MODBUS transaction at the same time.

The master node issues a MODBUS request to the slave nodes in two modes :

➔ In **unicast mode**, the master addresses an individual slave. After receiving and processing the request, the slave returns a message (a 'reply') to the master .

In that mode, a MODBUS transaction consists of 2 messages : a request from the master, and a reply from the slave.

Each slave must have an unique address (from 1 to 247) so that it can be addressed independently from other nodes.

➔ In **broadcast mode**, the master can send a request to all slaves.

No response is returned to broadcast requests sent by the master. The broadcast requests are necessarily writing commands. **All devices must accept the broadcast for writing function.** The address 0 is reserved to identify a broadcast exchange.

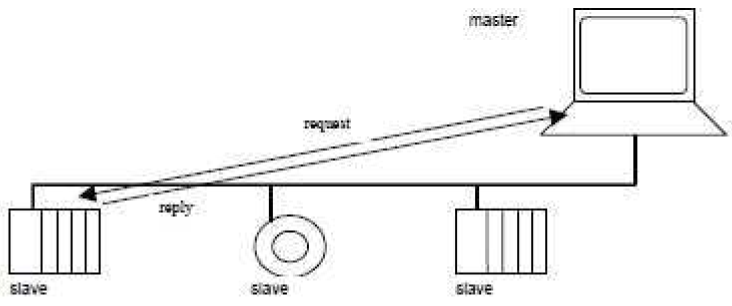


Figure 3: Unicast mode

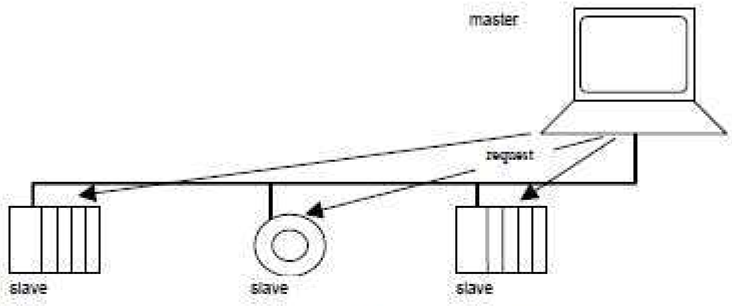
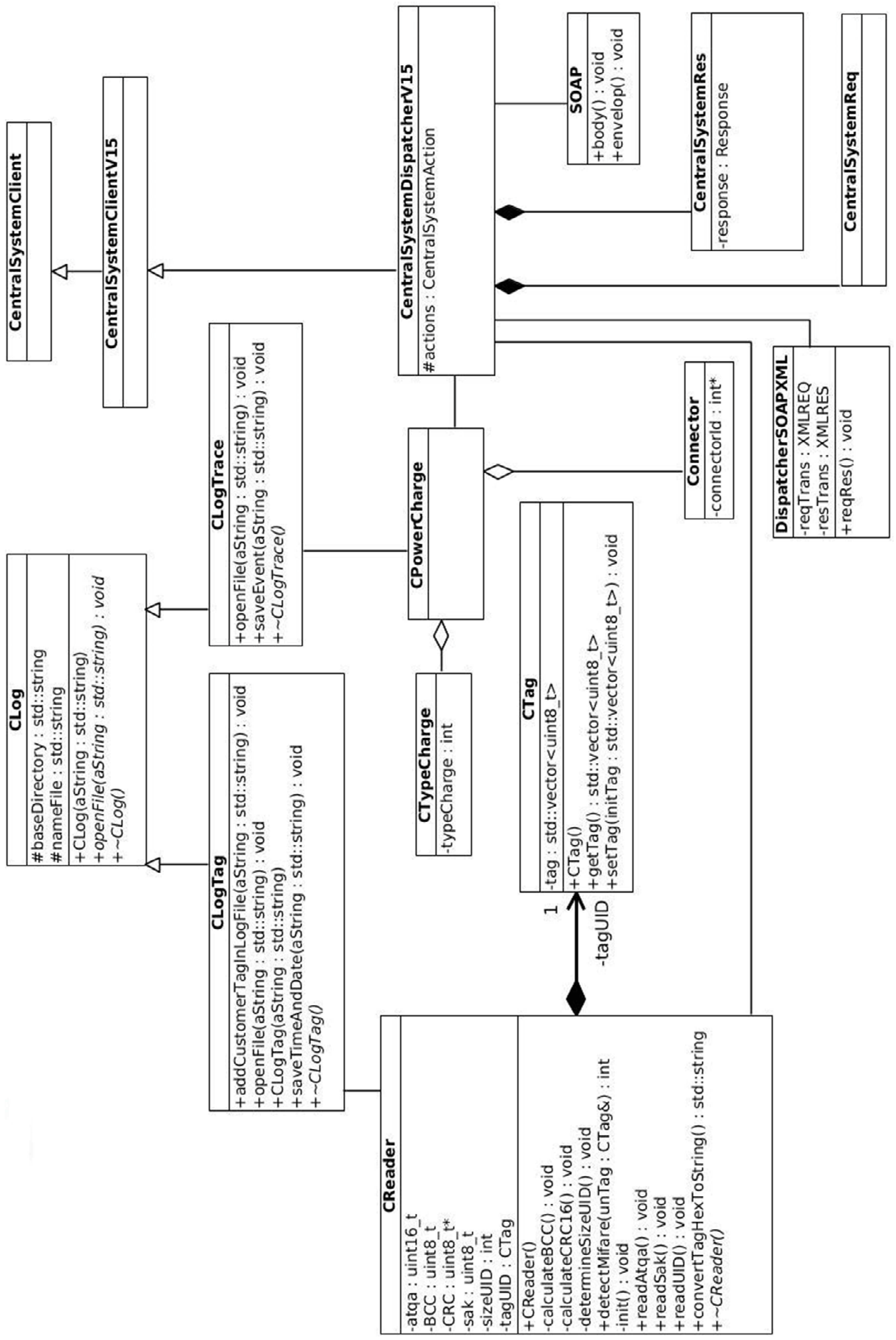


Figure 4: Broadcast mode

DOCUMENTATION PP5 : Diagramme de classes (Borne)



CentralSystemClient
CentralSystemClientV15
CentralSystemDispatcherV15
#actions : CentralSystemAction
SOAP
+body() : void
+envelop() : void
CentralSystemRes
-response : Response
CentralSystemReq
CLogTrace
+openFile(aString : std::string) : void
+saveEvent(aString : std::string) : void
+~CLogTrace()
CLog
#baseDirectory : std::string
#nameFile : std::string
+CLog(aString : std::string)
+openFile(aString : std::string) : void
+~CLog()
CPowerCharge
Connector
-connectorId : int*
DispatcherSOAPXML
-reqTrans : XMLREQ
-resTrans : XMLRES
+reqRes() : void
CTypeCharge
-typeCharge : int
CTag
-tag : std::vector<uint8_t>
+CTag()
+getTag() : std::vector<uint8_t>
+setTag(initTag : std::vector<uint8_t>) : void
CLogTag
+addCustomerTagInLogFile(aString : std::string) : void
+openFile(aString : std::string) : void
+CLogTag(aString : std::string)
+saveTimeAndDate(aString : std::string) : void
+~CLogTag()
1
-tagUID
CReader
-atqa : uint16_t
-BCC : uint8_t
-CRC : uint8_t*
-sak : uint8_t
-sizeUID : int
-tagUID : CTag
+CReader()
-calculateBCC() : void
-calculateCRC16() : void
-determineSizeUID() : void
+detectMifare(unTag : CTag&) : int
-init() : void
+readAtqa() : void
+readSak() : void
+readUID() : void
+convertTagHexToString() : std::string
+~CReader()

DOCUMENTATION PP6 : ATQA Coding of NXP Contactless Card ICs

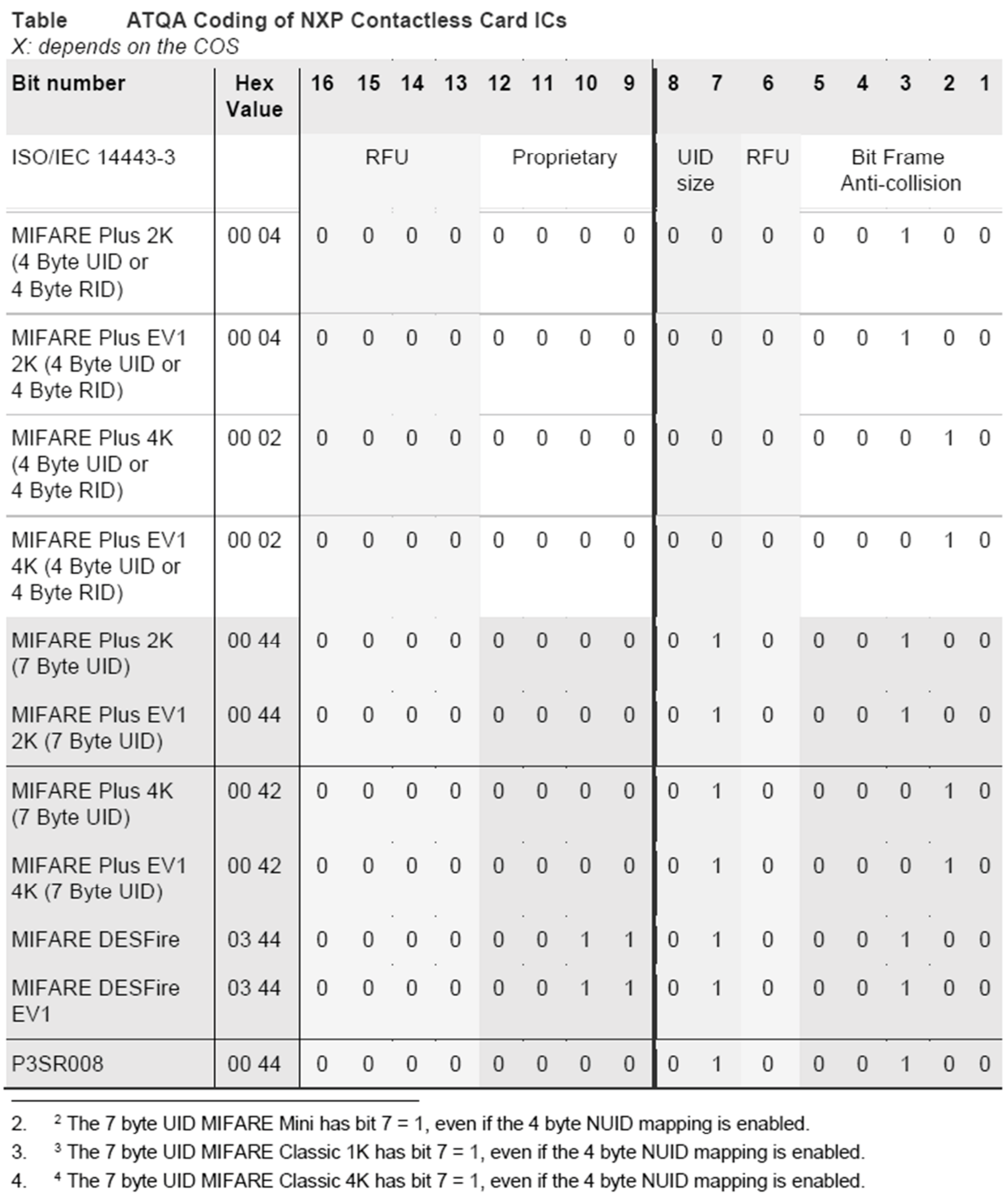


Table ATQA Coding of NXP Contactless Card ICs
X: depends on the COS

Bit number	**Hex Value**	**16**	**15**	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
ISO/IEC 14443-3		RFU				Proprietary				UID size		RFU	Bit Frame Anti-collision				
MIFARE Plus 2K (4 Byte UID or 4 Byte RID)	00 04	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
MIFARE Plus EV1 2K (4 Byte UID or 4 Byte RID)	00 04	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
MIFARE Plus 4K (4 Byte UID or 4 Byte RID)	00 02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
MIFARE Plus EV1 4K (4 Byte UID or 4 Byte RID)	00 02	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
MIFARE Plus 2K (7 Byte UID)	00 44	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
MIFARE Plus EV1 2K (7 Byte UID)	00 44	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0
MIFARE Plus 4K (7 Byte UID)	00 42	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
MIFARE Plus EV1 4K (7 Byte UID)	00 42	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
MIFARE DESFire	03 44	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0
MIFARE DESFire EV1	03 44	0	0	0	0	0	0	1	1	0	1	0	0	0	1	0	0
P3SR008	00 44	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0

2. [2] The 7 byte UID MIFARE Mini has bit 7 = 1, even if the 4 byte NUID mapping is enabled.
3. [3] The 7 byte UID MIFARE Classic 1K has bit 7 = 1, even if the 4 byte NUID mapping is enabled.
4. [4] The 7 byte UID MIFARE Classic 4K has bit 7 = 1, even if the 4 byte NUID mapping is enabled.

DOCUMENTATION PP7 : Principales requêtes SQL

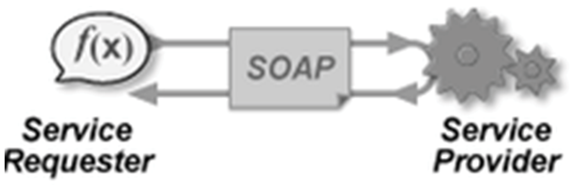
Utiliser (rendre active) une base de données existante :	`USE nom_de_la_base;`
Créer une base de données :	`CREATE DATABASE nom_de_la_base;`
Supprimer une base de données	`DROP DATABASE nom_de_la_base;`
Créer une table dans la base de données active:	`CREATE TABLE nomTable (id INT NOT NULL AUTO_INCREMENT, champ1 DOUBLE, champ2 VARCHAR, champ3 TIMESTAMP NOT NULL, …, PRIMARY KEY(id)) ;`
Lister la structure d'une table :	`DESCRIBE nomTable;`
Sélectionner toutes les informations de la table :	`SELECT * FROM nomTable ;`
Sélectionner seulement les informations d'un champ	`SELECT nomChamp FROM nomTable ;`
Sélectionner tous les champs de la table nomTable correspondant à deux critères.	`SELECT * FROM nomTable WHERE nomChamp1 = ‘poste’ AND nomChamp3 < 12 ;`
Sélectionner sur plusieurs tables (jointure) `nomTable1.nomChamp1 est clé primaire.` `nomTable2.nomChamp4 est une clé étrangère vers nomTable1.`	`SELECT * FROM nomTable1, nomTable2 WHERE nom_table1.nomChamp1 = nom_table2.nomChamp4 ;`
Écrire une nouvelle entrée dans une table de BDD	`INSERT INTO nomTable( champ1, champ2) VALUES( 'valeur1', 'valeur2') ;`
Modifier les informations de l'entrée dont le champ id = 51	`UPDATE nomTable SET nomChamp1=10, valeur2=32 WHERE id=51 ;`
Ajouter des nouveaux champs (colonnes) dans une table	`ALTER TABLE nomTable ADD champ1 DOUBLE, ADD champ2 BOOLEAN DEFAULT FALSE ;`

Les mots en gras dans la colonne de droite sont des mots réservés par le langage SQL.

DOCUMENTATION PP8 : SOAP

SOAP (*from Wikipedia*)

SOAP (abbreviation for **Simple Object Access Protocol**) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. Its purpose is to provide extensibility, neutrality and independence. It uses XML Information Set for its message format, and relies on application layer protocols, most often Hypertext Transfer Protocol (HTTP) or Simple Mail Transfer Protocol (SMTP), for message negotiation and transmission.



SOAP allows processes running on disparate operating systems (such as Windows and Linux) to communicate using Extensible Markup Language (XML). Since Web protocols like HTTP are installed and running on all operating systems, SOAP allows clients to invoke web services and receive responses independent of language and platforms.

Characteristics

SOAP provides the Messaging Protocol layer of a web services protocol stack for web services. It is an XML-based protocol consisting of three parts:

- an envelope, which defines the message structure and how to process it
- a set of encoding rules for expressing instances of application-defined datatypes
- a convention for representing procedure calls and responses

SOAP has three major characteristics:

1. *extensibility* (security and WS-Addressing are among the extensions under development)
2. *neutrality* (SOAP can operate over any protocol such as HTTP, SMTP, TCP, UDP, or JMS)
3. *independence* (SOAP allows for any programming model)

As an example of what SOAP procedures can do, an application can send a SOAP request to a server that has web services enabled—such as a real-estate price database—with the parameters for a search. The server then returns a SOAP response (an XML-formatted document with the resulting data), e.g., prices, location, features. Since the generated data comes in a standardized machine-parsable format, the requesting application can then integrate it directly.

The SOAP architecture consists of several layers of specifications for:

- message format
- Message Exchange Patterns (MEP)
- underlying transport protocol bindings
- message processing models
- protocol extensibility

SOAP evolved as a successor of XML-RPC, though it borrows its transport and interaction neutrality from Web Service Addressing and the envelope/header/body from elsewhere (probably from WDDX).

SOAP terminology

SOAP specification can be broadly defined to be consisting of the following 3 conceptual components: protocol concepts, encapsulation concepts and network concepts.

Data encapsulation concepts

- **SOAP message**: Represents the information being exchanged between 2 SOAP nodes.
- **SOAP envelope** : As per its name, it is the enclosing element of an XML message identifying it as a SOAP message.
- **SOAP header block**: A SOAP header can contain more than one of these blocks, each being a discrete computational block within the header. In general, the SOAP *role* information is used to target nodes on the path. A header block is said to be targeted at a SOAP node if the SOAP role for

the header block is the name of a role in which the SOAP node operates. (ex: A SOAP header block with role attribute as *ultimateReceiver* is targeted only at the destination node which has this role. A header with a role attribute as *next* is targeted at each intermediary as well as the destination node.)

- **SOAP header** : A collection of one or more header blocks targeted at each SOAP receiver.
- **SOAP body** : Contains the body of the message intended for the SOAP receiver. The interpretation and processing of SOAP body is defined by header blocks.
- **SOAP fault**: In case a SOAP node fails to process a SOAP message, it adds the fault information to the SOAP fault element. This element is contained within the SOAP body as a child element.

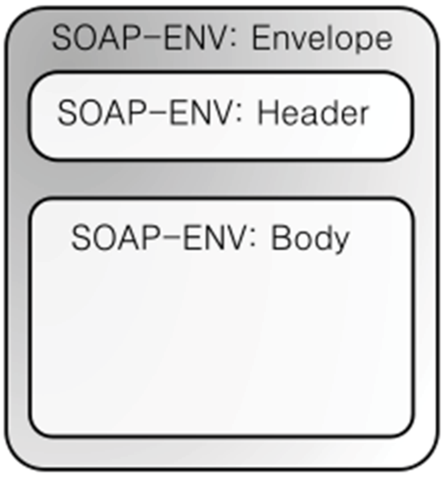
Message sender and receiver concepts

- **SOAP sender**: The node that transmits a SOAP message.
- **SOAP receiver** : The node receiving a SOAP message. (Could be an intermediary or the destination node.)
- **SOAP message path** : The path consisting of all the nodes that the SOAP message traversed to reach the destination node.
- **Initial SOAP sender**: This is the node which originated the SOAP message to be transmitted. This is the root of the SOAP message path.
- **SOAP intermediary**: All the nodes in between the SOAP originator and the intended SOAP destination. It processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate SOAP receiver.
- **Ultimate SOAP receiver**: The destination receiver of the SOAP message. This node is responsible for processing the message body and any header blocks targeted at it.

Specification

The SOAP specification defines the messaging framework, which consists of:

- The ***SOAP processing model*** defining the rules for processing a SOAP message
- The ***SOAP extensibility model*** defining the concepts of SOAP features and SOAP modules
- The ***SOAP underlying protocol binding*** framework describing the rules for defining a binding to an underlying protocol that can be used for exchanging SOAP messages between SOAP nodes
- The ***SOAP message construct*** defining the structure of a SOAP message



1 SOAP Structure

SOAP building blocks

A SOAP message is an ordinary XML document containing the following elements:

Element	Description	Required
Envelope	Identifies the XML document as a SOAP message.	Yes
Header	Contains header information.	No
Body	Contains call, and response information.	Yes
Fault	Provides information about errors that occurred while processing the message.	No

Transport methods

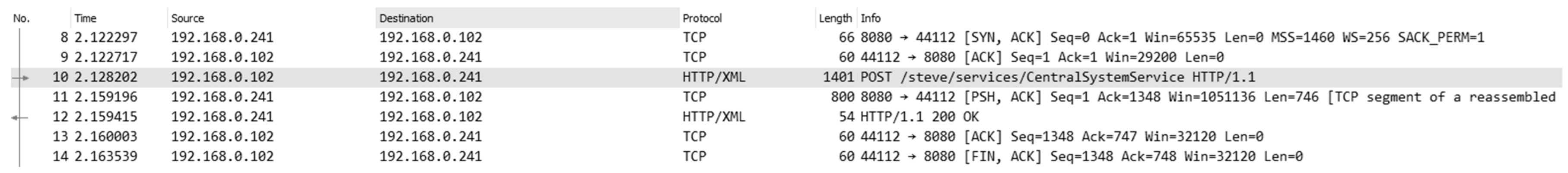
Both SMTP and HTTP are valid application layer protocols used as transport for SOAP, but HTTP has gained wider acceptance as it works well with today's internet infrastructure; specifically, HTTP works well with network firewalls. SOAP may also be used over HTTPS (which is the same protocol as HTTP at the application level, but uses an encrypted transport protocol underneath) with either simple or mutual authentication; this is the advocated WS-I method to provide web service security as stated in the WS-I Basic Profile

This is a major advantage over other distributed protocols like GIOP/IIOP or DCOM, which are normally filtered by firewalls. SOAP over AMQP is yet another possibility that some implementations support. SOAP also has an advantage over DCOM that it is unaffected by security rights configured on the machines that require knowledge of both transmitting and receiving nodes. This lets SOAP be loosely coupled in a way that is not possible with DCOM. There is also the SOAP-over-UDP OASIS standard.

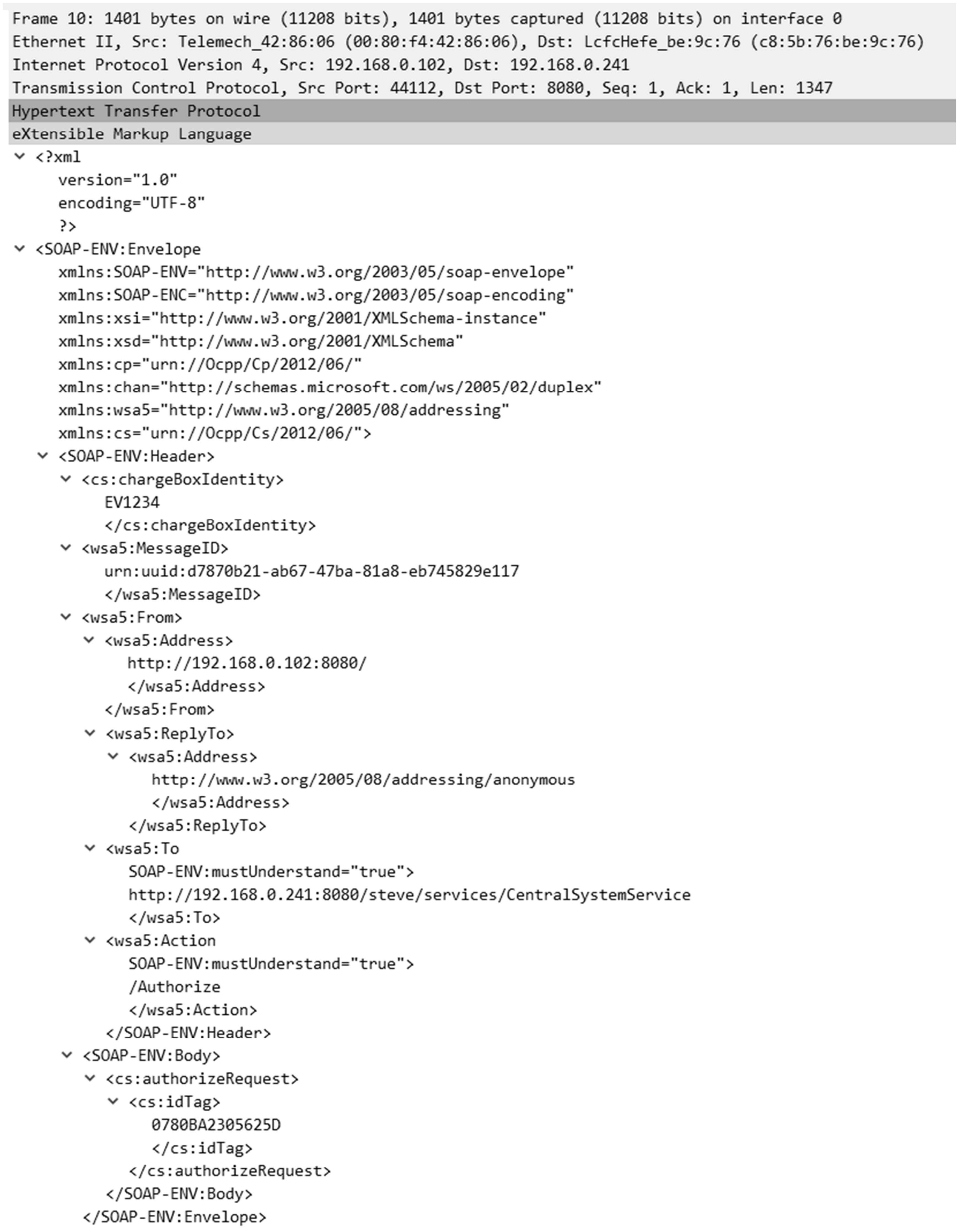
Example message (encapsulated in HTTP)

```
  POST /InStock HTTP/1.1
  Host: www.example.org
  Content-Type: application/soap+xml; charset=utf-8
  Content-Length: 299
  SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

  <?xml version="1.0"?>
  <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:m="http://www.example.org">
    <soap:Header>
    </soap:Header>
    <soap:Body>
      <m:GetStockPrice>
        <m:StockName>GOOG</m:StockName>
      </m:GetStockPrice>
    </soap:Body>
  </soap:Envelope>
```

Dialogue 1 : Borne -> Supervision

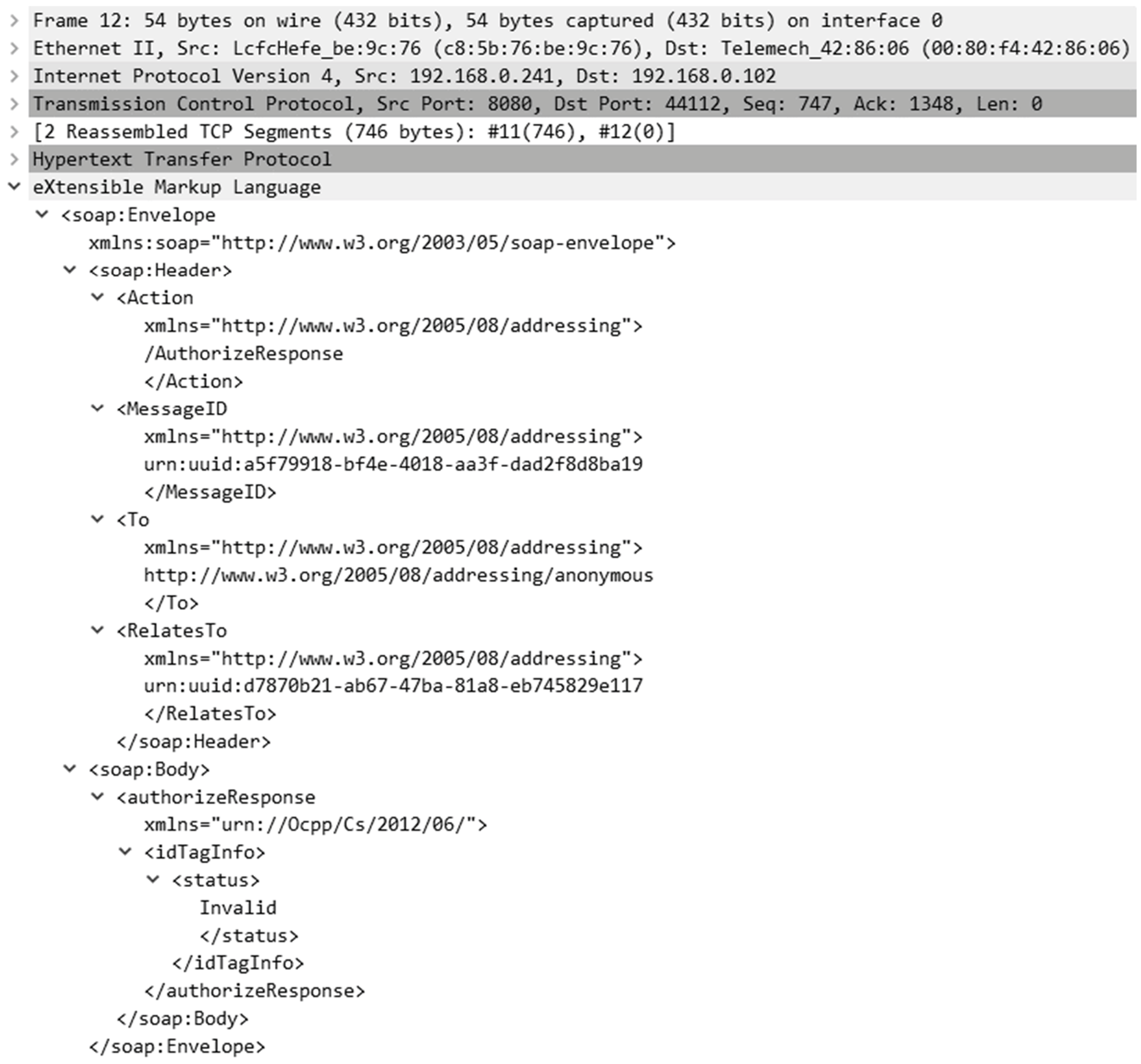


```
No.  Time      Source         Destination    Protocol  Length Info
  8 2.122297  192.168.0.241  192.168.0.102  TCP         66 8080 → 44112 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM=1
  9 2.122717  192.168.0.102  192.168.0.241  TCP         60 44112 → 8080 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 10 2.128202  192.168.0.102  192.168.0.241  HTTP/XML  1401 POST /steve/services/CentralSystemService HTTP/1.1
 11 2.159196  192.168.0.241  192.168.0.102  TCP        800 8080 → 44112 [PSH, ACK] Seq=1 Ack=1348 Win=1051136 Len=746 [TCP segment of a reassembled
 12 2.159415  192.168.0.241  192.168.0.102  HTTP/XML    54 HTTP/1.1 200 OK
 13 2.160003  192.168.0.102  192.168.0.241  TCP         60 44112 → 8080 [ACK] Seq=1348 Ack=747 Win=32120 Len=0
 14 2.163539  192.168.0.102  192.168.0.241  TCP         60 44112 → 8080 [FIN, ACK] Seq=1348 Ack=748 Win=32120 Len=0
```



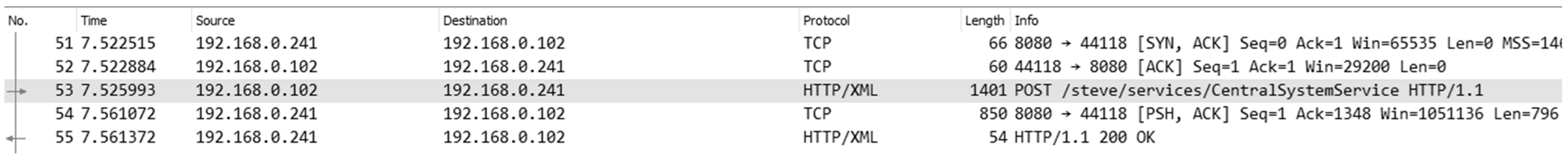
```
Frame 10: 1401 bytes on wire (11208 bits), 1401 bytes captured (11208 bits) on interface 0
Ethernet II, Src: Telemech_42:86:06 (00:80:f4:42:86:06), Dst: LcfcHefe_be:9c:76 (c8:5b:76:be:9c:76)
Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.241
Transmission Control Protocol, Src Port: 44112, Dst Port: 8080, Seq: 1, Ack: 1, Len: 1347
Hypertext Transfer Protocol
eXtensible Markup Language
<?xml
    version="1.0"
    encoding="UTF-8"
    ?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
    xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-encoding"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:cp="urn://Ocpp/Cp/2012/06/"
    xmlns:chan="http://schemas.microsoft.com/ws/2005/02/duplex"
    xmlns:wsa5="http://www.w3.org/2005/08/addressing"
    xmlns:cs="urn://Ocpp/Cs/2012/06/">
  <SOAP-ENV:Header>
    <cs:chargeBoxIdentity>
        EV1234
        </cs:chargeBoxIdentity>
    <wsa5:MessageID>
        urn:uuid:d7870b21-ab67-47ba-81a8-eb745829e117
        </wsa5:MessageID>
    <wsa5:From>
      <wsa5:Address>
          http://192.168.0.102:8080/
          </wsa5:Address>
        </wsa5:From>
      <wsa5:ReplyTo>
        <wsa5:Address>
            http://www.w3.org/2005/08/addressing/anonymous
            </wsa5:Address>
          </wsa5:ReplyTo>
      <wsa5:To
          SOAP-ENV:mustUnderstand="true">
          http://192.168.0.241:8080/steve/services/CentralSystemService
          </wsa5:To>
      <wsa5:Action
          SOAP-ENV:mustUnderstand="true">
          /Authorize
          </wsa5:Action>
        </SOAP-ENV:Header>
    <SOAP-ENV:Body>
      <cs:authorizeRequest>
        <cs:idTag>
            0780BA2305625D
            </cs:idTag>
          </cs:authorizeRequest>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

Dialogue 1 : Supervision -> Borne

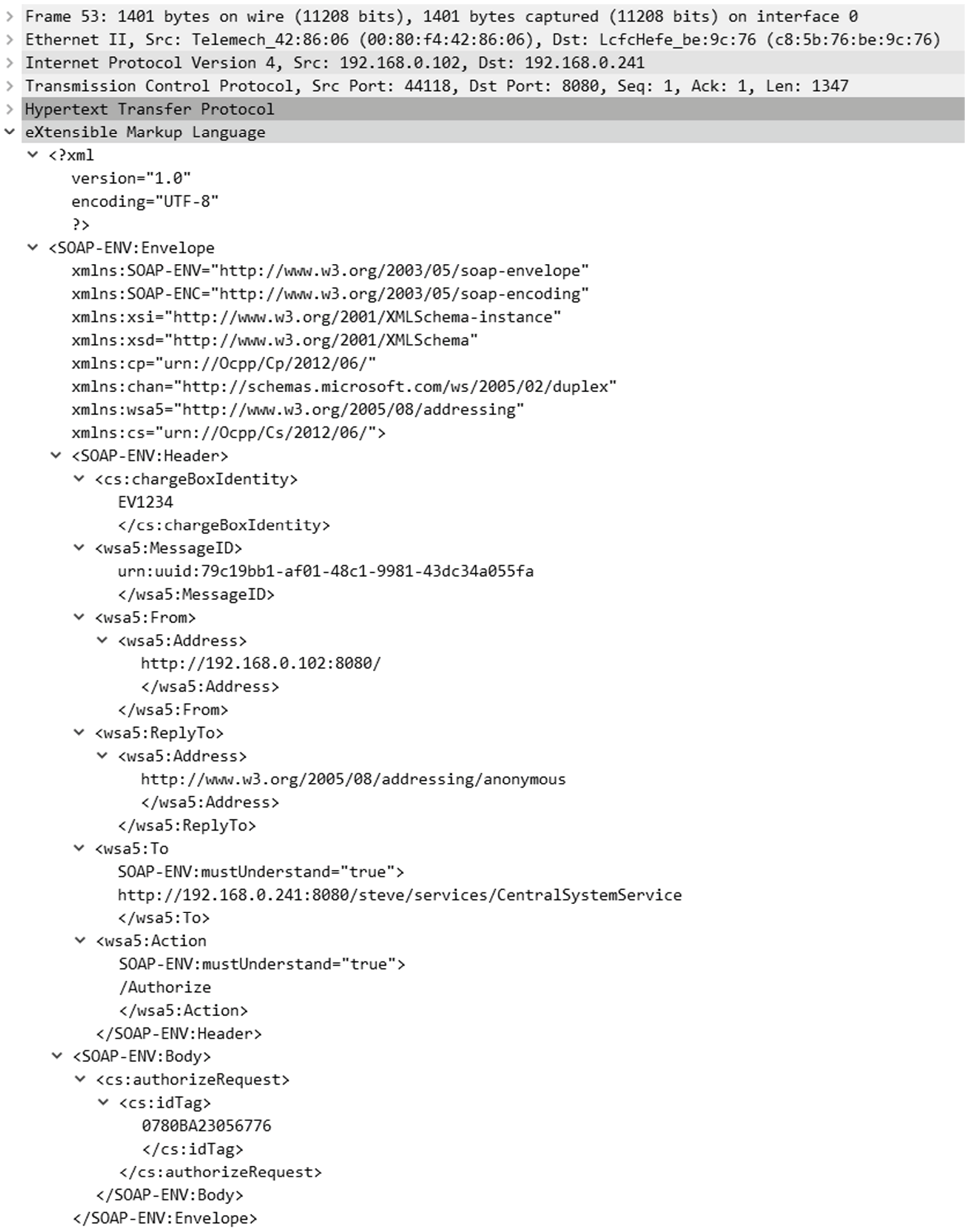


```
> Frame 12: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
> Ethernet II, Src: LcfcHefe_be:9c:76 (c8:5b:76:be:9c:76), Dst: Telemech_42:86:06 (00:80:f4:42:86:06)
> Internet Protocol Version 4, Src: 192.168.0.241, Dst: 192.168.0.102
> Transmission Control Protocol, Src Port: 8080, Dst Port: 44112, Seq: 747, Ack: 1348, Len: 0
> [2 Reassembled TCP Segments (746 bytes): #11(746), #12(0)]
> Hypertext Transfer Protocol
v eXtensible Markup Language
  v <soap:Envelope
      xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    v <soap:Header>
      v <Action
          xmlns="http://www.w3.org/2005/08/addressing">
          /AuthorizeResponse
          </Action>
      v <MessageID
          xmlns="http://www.w3.org/2005/08/addressing">
          urn:uuid:a5f79918-bf4e-4018-aa3f-dad2f8d8ba19
          </MessageID>
      v <To
          xmlns="http://www.w3.org/2005/08/addressing">
          http://www.w3.org/2005/08/addressing/anonymous
          </To>
      v <RelatesTo
          xmlns="http://www.w3.org/2005/08/addressing">
          urn:uuid:d7870b21-ab67-47ba-81a8-eb745829e117
          </RelatesTo>
        </soap:Header>
    v <soap:Body>
      v <authorizeResponse
          xmlns="urn://Ocpp/Cs/2012/06/">
        v <idTagInfo>
          v <status>
              Invalid
              </status>
            </idTagInfo>
          </authorizeResponse>
        </soap:Body>
    </soap:Envelope>
```

Dialogue 2 : Borne -> Supervision

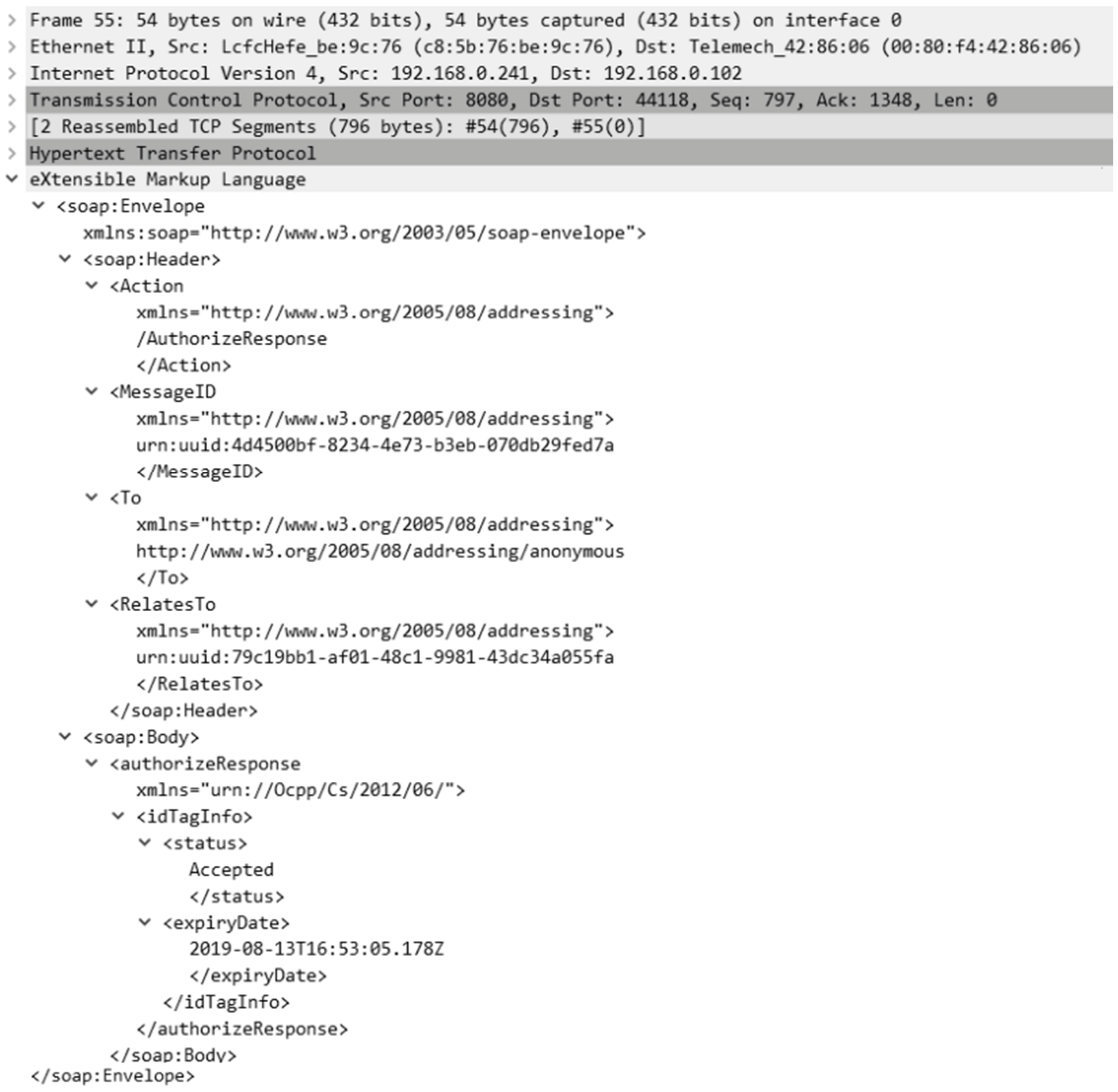


```
No.     Time       Source          Destination     Protocol  Length Info
     51 7.522515   192.168.0.241   192.168.0.102   TCP           66 8080 → 44118 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=14
     52 7.522884   192.168.0.102   192.168.0.241   TCP           60 44118 → 8080 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 →   53 7.525993   192.168.0.102   192.168.0.241   HTTP/XML    1401 POST /steve/services/CentralSystemService HTTP/1.1
     54 7.561072   192.168.0.241   192.168.0.102   TCP          850 8080 → 44118 [PSH, ACK] Seq=1 Ack=1348 Win=1051136 Len=796
 ←   55 7.561372   192.168.0.241   192.168.0.102   HTTP/XML      54 HTTP/1.1 200 OK
```



```
> Frame 53: 1401 bytes on wire (11208 bits), 1401 bytes captured (11208 bits) on interface 0
> Ethernet II, Src: Telemech_42:86:06 (00:80:f4:42:86:06), Dst: LcfcHefe_be:9c:76 (c8:5b:76:be:9c:76)
> Internet Protocol Version 4, Src: 192.168.0.102, Dst: 192.168.0.241
> Transmission Control Protocol, Src Port: 44118, Dst Port: 8080, Seq: 1, Ack: 1, Len: 1347
> Hypertext Transfer Protocol
v eXtensible Markup Language
  v <?xml
      version="1.0"
      encoding="UTF-8"
      ?>
  v <SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://www.w3.org/2003/05/soap-envelope"
      xmlns:SOAP-ENC="http://www.w3.org/2003/05/soap-encoding"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:cp="urn://Ocpp/Cp/2012/06/"
      xmlns:chan="http://schemas.microsoft.com/ws/2005/02/duplex"
      xmlns:wsa5="http://www.w3.org/2005/08/addressing"
      xmlns:cs="urn://Ocpp/Cs/2012/06/">
    v <SOAP-ENV:Header>
      v <cs:chargeBoxIdentity>
          EV1234
          </cs:chargeBoxIdentity>
      v <wsa5:MessageID>
          urn:uuid:79c19bb1-af01-48c1-9981-43dc34a055fa
          </wsa5:MessageID>
      v <wsa5:From>
        v <wsa5:Address>
            http://192.168.0.102:8080/
            </wsa5:Address>
          </wsa5:From>
      v <wsa5:ReplyTo>
        v <wsa5:Address>
            http://www.w3.org/2005/08/addressing/anonymous
            </wsa5:Address>
          </wsa5:ReplyTo>
      v <wsa5:To
          SOAP-ENV:mustUnderstand="true">
          http://192.168.0.241:8080/steve/services/CentralSystemService
          </wsa5:To>
      v <wsa5:Action
          SOAP-ENV:mustUnderstand="true">
          /Authorize
          </wsa5:Action>
        </SOAP-ENV:Header>
    v <SOAP-ENV:Body>
      v <cs:authorizeRequest>
        v <cs:idTag>
            0780BA23056776
            </cs:idTag>
          </cs:authorizeRequest>
        </SOAP-ENV:Body>
      </SOAP-ENV:Envelope>
```

Dialogue 2 : Supervision -> Borne

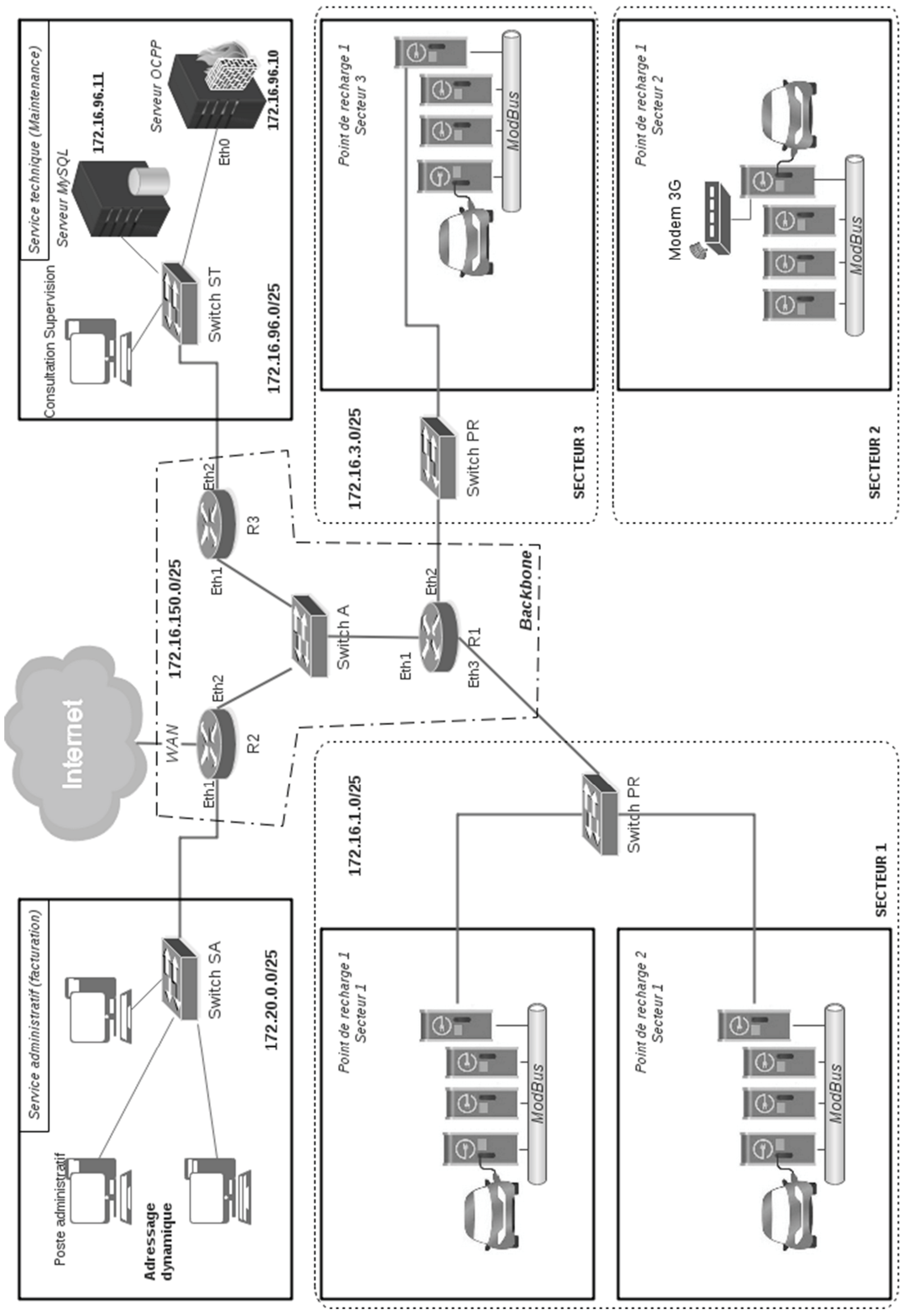


```
> Frame 55: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
> Ethernet II, Src: LcfcHefe_be:9c:76 (c8:5b:76:be:9c:76), Dst: Telemech_42:86:06 (00:80:f4:42:86:06)
> Internet Protocol Version 4, Src: 192.168.0.241, Dst: 192.168.0.102
> Transmission Control Protocol, Src Port: 8080, Dst Port: 44118, Seq: 797, Ack: 1348, Len: 0
> [2 Reassembled TCP Segments (796 bytes): #54(796), #55(0)]
> Hypertext Transfer Protocol
v eXtensible Markup Language
  v <soap:Envelope
        xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
    v <soap:Header>
      v <Action
          xmlns="http://www.w3.org/2005/08/addressing">
          /AuthorizeResponse
          </Action>
      v <MessageID
          xmlns="http://www.w3.org/2005/08/addressing">
          urn:uuid:4d4500bf-8234-4e73-b3eb-070db29fed7a
          </MessageID>
      v <To
          xmlns="http://www.w3.org/2005/08/addressing">
          http://www.w3.org/2005/08/addressing/anonymous
          </To>
      v <RelatesTo
          xmlns="http://www.w3.org/2005/08/addressing">
          urn:uuid:79c19bb1-af01-48c1-9981-43dc34a055fa
          </RelatesTo>
        </soap:Header>
    v <soap:Body>
      v <authorizeResponse
          xmlns="urn://Ocpp/Cs/2012/06/">
        v <idTagInfo>
          v <status>
              Accepted
              </status>
          v <expiryDate>
              2019-08-13T16:53:05.178Z
              </expiryDate>
            </idTagInfo>
          </authorizeResponse>
        </soap:Body>
</soap:Envelope>
```

DOCUMENTATION PP10 : Extrait du fichier WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
                  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
                  xmlns:s="http://www.w3.org/2001/XMLSchema"
                  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
                  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
                  xmlns:tns="urn://Ocpp/Cs/2012/06/"
                  targetNamespace="urn://Ocpp/Cs/2012/06/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
                  xmlns:wsa="http://www.w3.org/2005/08/addressing"
                  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
                  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">

    <wsdl:types>
    <s:schema targetNamespace="urn://Ocpp/Cs/2012/06/" elementFormDefault="qualified">

      <!-- Begin of types shared with ChargePointService -->
      <s:simpleType name="IdToken">
        <s:annotation>
          <s:documentation>Type of string defining identification token, e.g. RFID or credit card number. To be treated as case insensitive.</s:documentation>
        </s:annotation>
        <s:restriction base="s:string">
          <s:maxLength value="20"/>
        </s:restriction>
      </s:simpleType>
        <s:simpleType name="AuthorizationStatus">
        <s:annotation>
          <s:documentation>Defines the authorization-status-value</s:documentation>
        </s:annotation>
        <s:restriction base="s:string">
          <s:enumeration value="Accepted"/>
          <s:enumeration value="Blocked"/>
          <s:enumeration value="Expired"/>
          <s:enumeration value="Invalid"/>
          <s:enumeration value="ConcurrentTx"/>
        </s:restriction>
      </s:simpleType>

      <s:complexType name="IdTagInfo">
        <s:sequence>
          <s:element name="status" type="tns:AuthorizationStatus" minOccurs="1" maxOccurs="1"/>
          <s:element name="expiryDate" type="s:dateTime" minOccurs="0" maxOccurs="1"/>
          <s:element name="parentIdTag" type="tns:IdToken" minOccurs="0" maxOccurs="1"/>
        </s:sequence>
      </s:complexType>
        <!-- End of types shared with ChargePointService -->

      <s:simpleType name="ChargeBoxSerialNumber">
        <s:annotation>
          <s:documentation>String type of max 25 chars that is to be treated as case insensitive.</s:documentation>
        </s:annotation>
        <s:restriction base="s:string">
          <s:maxLength value="25"/>
        </s:restriction>
      </s:simpleType>

      <s:simpleType name="ChargePointModel">
        <s:annotation>
          <s:documentation>String type of max 20 chars that is to be treated as case insensitive.</s:documentation>
        </s:annotation>
    <s:restriction base="s:string">
      <s:maxLength value="20"/>
    </s:restriction>
  </s:simpleType>


<s:simpleType name="ChargePointSerialNumber">
  <s:annotation>
    <s:documentation>String type of max 25 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="25"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="ChargePointVendor">
  <s:annotation>
    <s:documentation>String type of max 20 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="20"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="FirmwareVersion">
  <s:annotation>
    <s:documentation>String type of max 50 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="50"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="IccidString">
  <s:annotation>
    <s:documentation>String type of max 20 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="20"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="ImsiString">
  <s:annotation>
    <s:documentation>String type of max 20 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="20"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="MeterSerialNumber">
  <s:annotation>
    <s:documentation>String type of max 25 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="25"/>
  </s:restriction>
</s:simpleType>

<s:simpleType name="MeterType">
  <s:annotation>
    <s:documentation>String type of max 25 chars that is to be treated as case insensitive.</s:documentation>
  </s:annotation>
  <s:restriction base="s:string">
    <s:maxLength value="25"/>
  </s:restriction>
</s:simpleType>
```

DOCUMENTATION PP11 : Infrastructure réseau



DOCUMENTATION PP12 : iptables

NOM

iptables - outil d'administration pour le filtrage de paquets IPv4 et le NAT

DESCRIPTION

iptables est utilisé pour mettre en place, maintenir et inspecter les tables des règles de filtrage des paquets IP du noyau Linux. Différentes tables peuvent être définies. Chaque table contient plusieurs chaînes prédéfinies et peut aussi contenir des chaînes définies par l'utilisateur.

Chaque chaîne est une liste de règles que peuvent vérifier un ensemble de paquets ; dans ce cas, on dit qu'on cherche à établir une correspondance avec la règle. Chaque règle détermine ce qui doit être fait avec un paquet qui correspond. Cette action est appelée une «cible», qui peut être un saut vers une chaîne définie par l'utilisateur dans la même table.

filter :

C'est la table par défaut (si l'option -t est omise). Elle contient les chaînes prédéfinies **INPUT** (pour les paquets entrants dans la machine), **FORWARD** (pour les paquets routés à travers la machine) et **OUTPUT** (pour les paquets générés localement).

COMMANDES

Ces options précisent une action particulière à accomplir. Une seule option peut être indiquée sur la ligne de commande, sauf indication contraire. Pour tous les noms en version longue des commandes et des options, vous avez le droit d'utiliser un nombre restreint de lettres du moment qu' **iptables** peut identifier chaque commande sans ambiguïté.

-A, --append *chaîne règle*

Ajoute une ou plusieurs règles à la fin de la chaîne sélectionnée. Lorsque les noms source et/ou destination désignent plus d'une adresse, une règle sera ajoutée pour chaque combinaison d'adresses possible.

-D, --delete *chaîne règle*

Les chaînes standards :

FORWARD : chaine désignant les paquets désirant traverser le pare-feu
INPUT : chaine désignant les paquets s'adressant au pare-feu lui-même
OUTPUT : chaine désignant les paquets expédiés par le pare-feu lui-même
PREROUTING : chaine désignant les paquets attendant d'être routés
POSTROUTING : chaine désignant les paquets venant d'être routés

PARAMÈTRES

Les paramètres suivants composent une spécification de règle (quand ils sont utilisés dans les commandes **add**, **delete**, **insert**, **replace** et **append**).

-p, --protocol [!] *protocole*

Protocole de la règle ou du paquet à vérifier. Le protocole spécifié est l'un des suivants : *tcp*, *udp*, *icmp, ftp, ssh* ou *all*, ou bien sous forme d'une valeur numérique, représentant un de ces protocoles ou un protocole différent. Un nom de protocole issu du fichier /etc/protocols est aussi autorisé. Un «!» avant le protocole inverse le test. La valeur zéro est équivalente à *all*. Le protocole *all* correspond à tous les protocoles ; c'est aussi la valeur par défaut lorsque cette option est omise.

-s, --source [!] *adresse*[/*masque*]

Spécification de la source. L'*adresse* peut être un nom de réseau, un nom d'hôte (attention : spécifier un nom à résoudre avec une requête distante de type DNS est vraiment une mauvaise idée), une adresse de réseau IP (avec /masque) ou une simple adresse IP. Le *masque* peut être un masque de réseau ou un nombre entier spécifiant le nombre de bits égaux à 1 dans la partie gauche du masque de réseau (bits de poids fort). Par conséquent, un masque de *24* est équivalent à *255.255.255.0*. Un «!» avant la spécification d'adresse inverse la sélection d'adresse. L'option **--src** est un synonyme de --source.

-d, --destination [!] *adresse*[/*masque*]

Spécification de la destination. Voir la description du paramètre **-s** (source) pour une description détaillée de la syntaxe. L'option **--dst** est un synonyme de --destination.

-j, --jump *cible*

Ceci détermine la cible de la règle ; c'est-à-dire ce qu'il faut faire si le paquet correspond à la règle. La cible peut être une chaîne définie par l'utilisateur (autre que celle dans laquelle se situe cette règle), une des cibles prédéfinies qui décide immédiatement du sort du paquet, ou une extension (voir **EXTENSIONS** ci-dessous). Si cette option est omise dans une règle, la correspondance d'un paquet avec la règle n'aura aucun effet sur le sort du paquet, mais les compteurs seront incrémentés.

-i, --in-interface [!] [*nom*]

Nom de l'interface qui reçoit les paquets (seulement pour les paquets passant par les chaînes **INPUT**, **FORWARD** et **PREROUTING**). Lorsqu'un «!» est utilisé avant le nom d'interface, la sélection est inversée. Si le nom de l'interface se termine par un «+», il désigne toutes les interfaces commençant par ce nom. Si cette option est omise, toutes les interfaces réseau sont désignées.

-o, --out-interface [!] [*nom*]

Nom de l'interface qui envoie les paquets (seulement pour les paquets passant par les chaînes **FORWARD**, **OUTPUT** et **POSTROUTING**). Lorsqu'un «!» est utilisé avant le nom d'interface, la sélection est inversée. Si le nom de l'interface se termine par un «+», il désigne toutes les interfaces commençant par ce nom. Si cette option est omise, toutes les interfaces réseau sont désignées.

CIBLES

Une règle de pare-feu spécifie des critères de correspondance pour un paquet et une cible. Si le paquet correspond, la règle suivante est déterminée par la valeur de la cible, qui peut être une des valeurs spéciales suivantes : *ACCEPT, DROP,…*

ACCEPT signifie que le paquet est autorisé à passer

DROP signifie que le paquet est rejeté ou détruit.

Exemples :

iptables -A FORWARD -i eth0 -o eth1 -p ftp -j ACCEPT

ajoute une règle qui autorise les paquets ftp à traverser la machine s’il rentre par l’interface réseau eth0 et sort par l’interface réseau eth1.

iptables -A INPUT -s 192.168.0.0/24 -i eth0 -j DROP

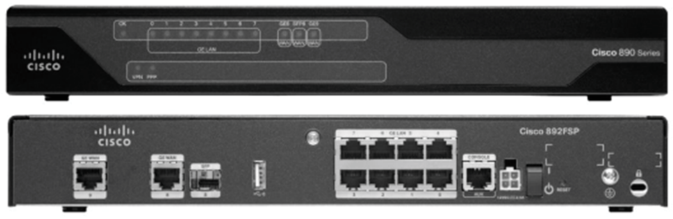
ajoute une règle qui rejette tous les paquets provenant des machines du réseau 192.168.0.0/24 entrants par l’interface réseau eth0 et destinés à cette machine.

iptables -A INPUT -i eth0 -p tcp -j ACCEPT

ajoute une règle qui autorise tous les paquets tcp entrants par l’interface eth0 et destinés à cette machine.

DOCUMENTATION PP13 : CISCO 890 series

Cisco 890 Series Integrated Services Routers



Cisco 890 Series ISRs come with an 8-port managed switch, providing LAN ports to connect multiple devices. An optional Power-over-Ethernet (PoE) capability can also supply power to IP phones and other devices. Eleven Cisco 890 Series models are available: Figure 1 shows the front and back of one, the Cisco 892FSP.

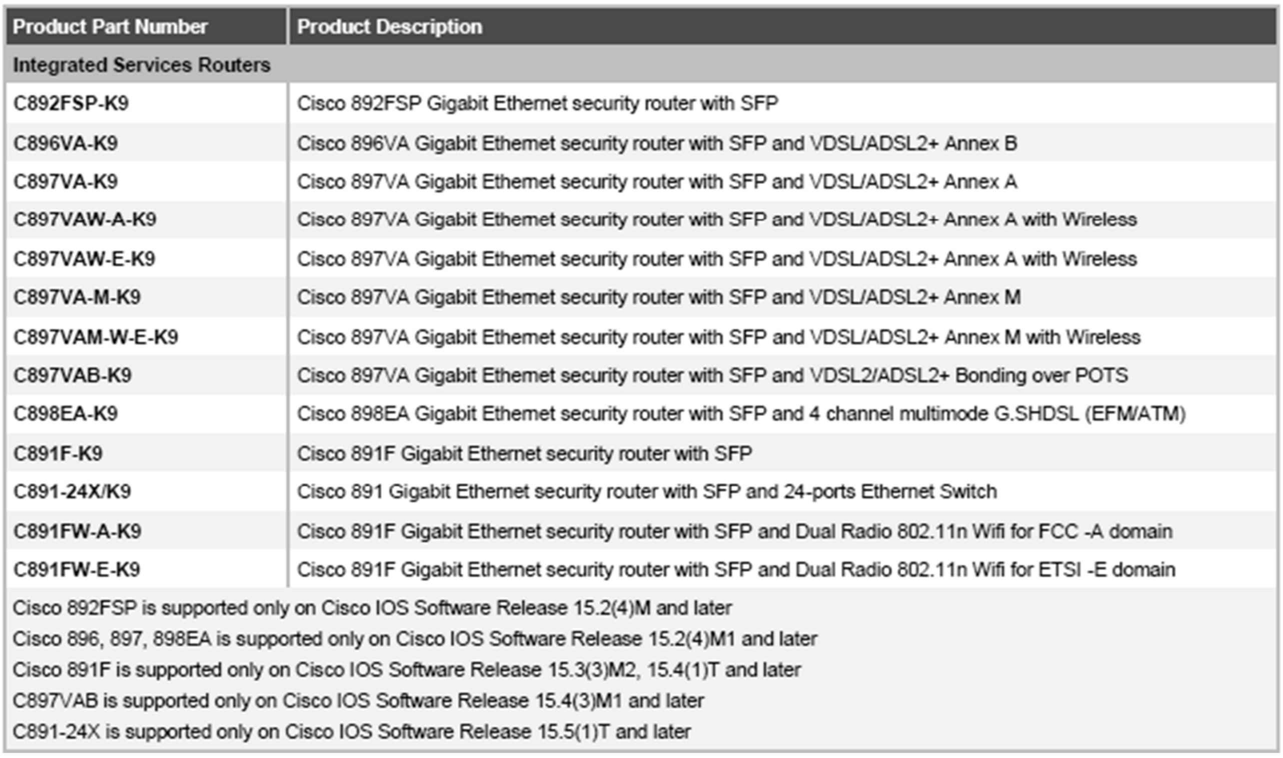
Product Specifications

Table 3 shows Cisco IOS Software features, WLAN features, and general system specifications for the 890 Series ISRs.

Table 3. 890 Series IOS Software Features, WLAN Features, and System Specifications

Feature	Specification
Cisco IOS Software: Advanced IP Features Set (Default)	
IP and IP services	• Routing Information Protocol Versions 1 and 2 (RIPv1 and RIPv2) • Generic Routing Encapsulation (GRE) and Multipoint GRE (MGRE) • Cisco Express Forwarding • Standard 802.1d Spanning Tree Protocol • Layer 2 Tunneling Protocol (L2TP) • Layer 2 Tunneling Protocol Version 3 (L2TPv3) • Network Address Translation (NAT) • Dynamic Host Configuration Protocol (DHCP) server, relay, and client • Dynamic Domain Name System (DNS) • DNS Proxy • DNS Spoofing • Access control Lists (ACLs) • IPv4 and IPv6 Multicast • Open Shortest Path First (OSPF) • Border Gateway Protocol (BGP) • Performance Routing (PfR) • Enhanced Interior Gateway Routing Protocol (EIGRP) • Virtual Route Forwarding (VRF) Lite • Next Hop Resolution Protocol (NHRP) • Bidirectional Forwarding Detection (BFD)
xDSL	• True Multimode VDSL2 and ADSL2+ over Annex A, B, J, and M including traditional G.DMT and T1.413 • World-class interoperability with industry-standard DSL access multiplexer (DSLAM) chipsets • Highest field reliability with Impulse Noise Protection over REIN/SHINE, Extended INP-Delay, G.INP, Physical Layer Retransmission, SRA, and Bitswap • VDSL2 Persistent Storage Device (PSD) profiles up to 17a/b with support for Spectral Shaping • VDSL2 Vectoring to offer blazing fiber speeds over copper • Support for 4-pair multimode G.SHDSL; that is, ATM and EFM • Remote management with TR069 and CWMP • Investment protection with GE and SFP for future fiber that could replace xDSL deployment
Switch features	• Auto Media Device In/Media Device Cross Over (MDI-MDX) • 25 802.1QVLANs • MAC filtering • Four-port 802.3af and Cisco compliant PoE • Switched Port Analyzer (SPAN) • Storm Control • Smart ports • Secure MAC address • Internet Group Management Protocol Version 3 (IGMPv3) snooping • 802.1x

Table 4. Product Part Numbers and Software Images



Product Part Number	Product Description
Integrated Services Routers	
C892FSP-K9	Cisco 892FSP Gigabit Ethernet security router with SFP
C896VA-K9	Cisco 896VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex B
C897VA-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex A
C897VAW-A-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex A with Wireless
C897VAW-E-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex A with Wireless
C897VA-M-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex M
C897VAM-W-E-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL/ADSL2+ Annex M with Wireless
C897VAB-K9	Cisco 897VA Gigabit Ethernet security router with SFP and VDSL2/ADSL2+ Bonding over POTS
C898EA-K9	Cisco 898EA Gigabit Ethernet security router with SFP and 4 channel multimode G.SHDSL (EFM/ATM)
C891F-K9	Cisco 891F Gigabit Ethernet security router with SFP
C891-24X/K9	Cisco 891 Gigabit Ethernet security router with SFP and 24-ports Ethernet Switch
C891FW-A-K9	Cisco 891F Gigabit Ethernet security router with SFP and Dual Radio 802.11n Wifi for FCC -A domain
C891FW-E-K9	Cisco 891F Gigabit Ethernet security router with SFP and Dual Radio 802.11n Wifi for ETSI -E domain
Cisco 892FSP is supported only on Cisco IOS Software Release 15.2(4)M and later Cisco 896, 897, 898EA is supported only on Cisco IOS Software Release 15.2(4)M1 and later Cisco 891F is supported only on Cisco IOS Software Release 15.3(3)M2, 15.4(1)T and later C897VAB is supported only on Cisco IOS Software Release 15.4(3)M1 and later C891-24X is supported only on Cisco IOS Software Release 15.5(1)T and later	

DOCUMENTATION PP14 : std::vector

std::vector

template < class T, class Alloc = allocator<T> > class vector; // generic template

Vectors are sequence containers representing arrays that can change in size.

Just like arrays, vectors use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.

Internally, vectors use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

operator [ ]

const_reference operator[ ] (size_type **n**) const;

Access element **n** in the vector container.

A similar member function, vector::**at**, has the same behavior as this operator function, except that vector::**at** is bound-checked and signals if the requested position is out of range by throwing an out_of_range exception.

Portable programs should never call this function with an argument n that is out of range, since this causes undefined behavior.

Parameters

n

Position of an element in the container.
Notice that the first element has a position of 0 (not 1).
Member type size_type is an unsigned integral type.

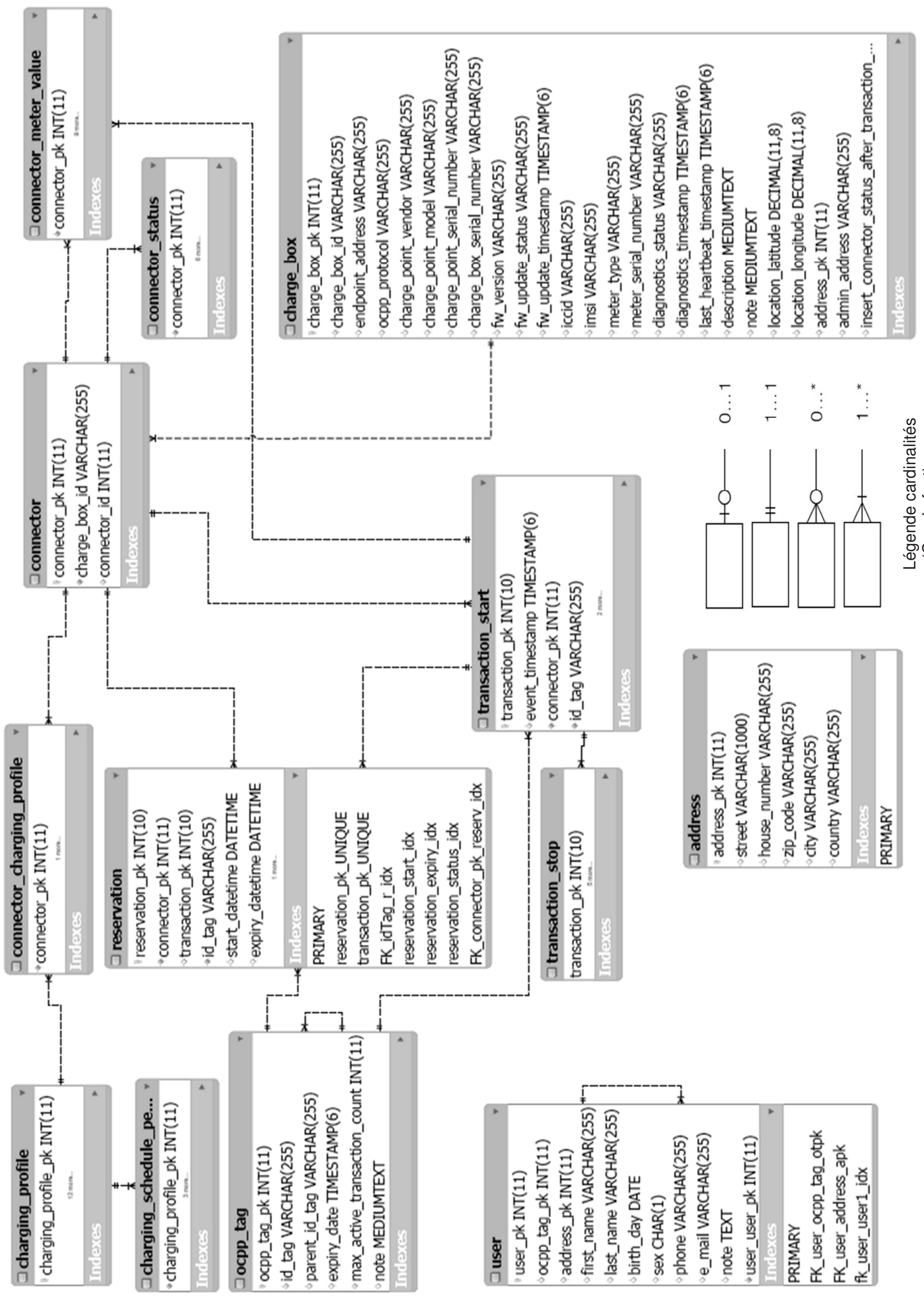
return value

The element at the specified position in the vector.

Example

```
#include <iostream>
#include <vector>
int main ( )
{
std::vector<int> myVector ( 10 );   // 10 non-initialized integers
int tailleVector = myVector.size();
    for ( unsigned indice = 0; indice < tailleVector; indice++ )
            std::cout << '  ' << myVector[ indice ];
    std::cout << '\n';
    return 0;
}
```

DOCUMENTATION PP15 : Schéma entités-relations incomplet de la BDD de la supervision


connector_meter_value
connector_pk INT(11)
Indexes
connector_status
connector_pk INT(11)
Indexes
charge_box
charge_box_pk INT(11)
charge_box_id VARCHAR(255)
endpoint_address VARCHAR(255)
ocpp_protocol VARCHAR(255)
charge_point_vendor VARCHAR(255)
charge_point_model VARCHAR(255)
charge_point_serial_number VARCHAR(255)
charge_box_serial_number VARCHAR(255)
fw_version VARCHAR(255)
fw_update_status VARCHAR(255)
fw_update_timestamp TIMESTAMP(6)
iccid VARCHAR(255)
imsi VARCHAR(255)
meter_type VARCHAR(255)
meter_serial_number VARCHAR(255)
diagnostics_status VARCHAR(255)
diagnostics_timestamp TIMESTAMP(6)
last_heartbeat_timestamp TIMESTAMP(6)
description MEDIUMTEXT
note MEDIUMTEXT
location_latitude DECIMAL(11,8)
location_longitude DECIMAL(11,8)
address_pk INT(11)
admin_address VARCHAR(255)
insert_connector_status_after_transaction_...
Indexes
connector
connector_pk INT(11)
charge_box_id VARCHAR(255)
connector_id INT(11)
Indexes
transaction_start
transaction_pk INT(10)
event_timestamp TIMESTAMP(6)
connector_pk INT(11)
id_tag VARCHAR(255)
Indexes
0...1
1...1
0...*
1...*
Légende cardinalités (Crow's foot) :
connector_charging_profile
connector_pk INT(11)
Indexes
reservation
reservation_pk INT(10)
connector_pk INT(11)
transaction_pk INT(10)
id_tag VARCHAR(255)
start_datetime DATETIME
expiry_datetime DATETIME
Indexes
PRIMARY
reservation_pk_UNIQUE
transaction_pk_UNIQUE
FK_idTag_r_idx
reservation_start_idx
reservation_expiry_idx
reservation_status_idx
FK_connector_pk_reserv_idx
transaction_stop
transaction_pk INT(10)
Indexes
address
address_pk INT(11)
street VARCHAR(1000)
house_number VARCHAR(255)
zip_code VARCHAR(255)
city VARCHAR(255)
country VARCHAR(255)
Indexes
PRIMARY
charging_profile
charging_profile_pk INT(11)
Indexes
charging_schedule_pe...
charging_profile_pk INT(11)
Indexes
ocpp_tag
ocpp_tag_pk INT(11)
id_tag VARCHAR(255)
parent_id_tag VARCHAR(255)
expiry_date TIMESTAMP(6)
max_active_transaction_count INT(11)
note MEDIUMTEXT
Indexes
user
user_pk INT(11)
ocpp_tag_pk INT(11)
address_pk INT(11)
first_name VARCHAR(255)
last_name VARCHAR(255)
birth_day DATE
sex CHAR(1)
phone VARCHAR(255)
e_mail VARCHAR(255)
note TEXT
user_user_pk INT(11)
Indexes
PRIMARY
FK_user_ocpp_tag_otpk
FK_user_address_apk
fk_user_user1_idx