DSRC Proxy

**Open issues to be address in the next revision**

* **Convert to the common COC template**
* **DSRC Proxy must import shared types from TCICommonTypes**
* **Update Time64 to TCICommonType**
* **Update DataRate to use from IEEE-1609-3-WEE?**
* **Alert that time stamp is in milliseconds, not microseconds**
* **Incorporate requirements from Danlaw for dynamic change of the channel**
* **Incorporate IP forwarding strategy…**

# General

DSRC Proxy is a piece of software running on the DSRC radio. It handles conversion of messages received between two interfaces: UDP messages exchanged on the Ethernet interface and wireless 802.11 frames exchanged on the radio interface.

Test System running on PC

SUT

Ethernet

DSRC

radio



DSRC-based

wireless connection

## User Cases

A sample of use cases for the DSRC Proxy is listed below. Specific values for certain parameters are provided for illustration purposes.

* **Message generator**: receive messages from the Test System on a UDP port 12333 and send them as 802.11 broadcast frames with transmit power 21, Data rate 6, User priority 3, on channel 172 during slot 0. This supports transmission of WSM, WSA and IP messages. Message signing for security is performed by the test system and included into the UDP message payload.
* **Message receiver**: receive 802.11 frames containing WSM/WSA/IP messages on specific radio channel and forward them to the Test System for further analysis. The WSA message signature verification is executed by the test system.

## Required Functionality

DSRC Proxy handles two message flows:

1. TS🡪SUT: Inbound from the Test System, outbound from the DSRC radio
2. SUT🡪TS: Outbound to the Test System, inbound from the DSRC radio

Case 1: TS🡪SUT

1. Test System sends UDP/IPv4 messages to the IP address and static UDP port on the DSRC Radio.
2. DSRC Proxy running on the DSRC radio:
   1. receives the UDP message
   2. extracts the payload
   3. constructs the 802.11
   4. configures radio parameters using information from the UDP message
   5. transmits 802.11 through DSRC radio

There’s no validation message transmitted back from the DSRC radio to the TS.

Case 2: SUT🡪TS

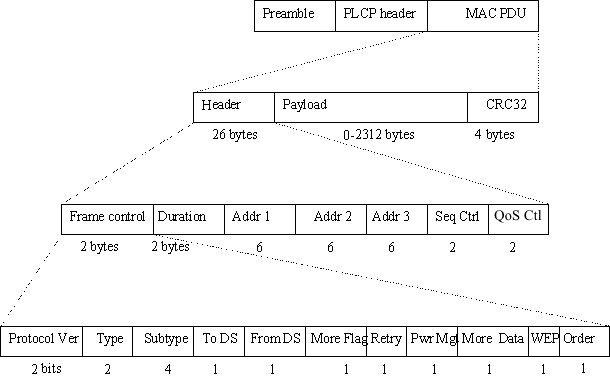
1. SUT transmits 802.11 frames over DSRC
2. DSRC Proxy running on the DSRC radio:
   1. receives the 802.11 frames
   2. constructs the UDP message
   3. include certain radio parameters
   4. transmits UDP message to the remote IP address and static UDP port of the TS.

There’s no validation message transmitted back from the DSRC radio to the SUT.

## Supported Message Formats

The DSRC Proxy is specified so that it can support WSM, WSA and IP messages.

The UDP payload will contain a set of parameters for radio configuration and encapsulate the ‘raw’ MAC frame to be transmitted over the air.



For the outbound DSRC messages, the MAC frame extracted from the UDP message will include:

* 802.11 header (QoS data frames)
* LLC header (including DSAP, SSAP, Control Field, Organization Code, Ethertype)
* (WSM header + body) OR (IP header + body)

Before transmission, the message will be appended with the MAC CRC field which will create a complete 802.11 frame. Preamble and PLCP header are generated by the radio device itself.

An example of MAC frame included into Opaque payload is shown below. This hexdump represents a simple packet with payload of

This is the payload

It consists of:

[000] 88 00 00 00 FF FF FF FF FF FF 04 E5 48 00 10 00

[010] FF FF FF FF FF FF FE FF 24 00 AA AA 03 00 00 00

[020] 88 B5 54 68 69 73 20 69 73 20 74 68 65 20 70 61

[030] 79 6C 6F 61 64 0A

with reference to the 802.11 frame fields:

88 00 IEEE80211QoSHeader

00 00 IEEE80211QoSHeader.DurationId

FF FF FF FF FF FF IEEE80211QoSHeader.Address1 (dest)

04 E5 48 00 10 00 IEEE80211QoSHeader.Address2 (src)

FF FF FF FF FF FF IEEE80211QoSHeader.Address3 (BSSID)

FE FF IEEE80211QoSHeader.tDot4SeqCtrl

(set by upper MAC)

24 00 IEEE80211QoSHeader.tDot4QoSCtrl

AA SNAPHeader.DSAP

AA SNAPHeader.SSAP

03 SNAPHeader.Control

00 00 00 SNAPHeader.OUI

88 B5 SNAPHeader.Type = 0x88b5

(IEEE Std 802 - Local Experimental Ethertype 2)

54 68 69 73 20 69 73 20 74 68 65 20 70 61 79 6C 6F 61 64 0A

"This is the payload\n"

For inbound wireless messages, the entire 802.11 frame will be encapsulated into UDP payload including:

* 802.11 header
* LLC header
* (WSM header + body) OR (IP header + body)

The DSRC Proxy can support transmission and reception of both secured and unsecure WSM/WSA messages. According to the IEEE 1609.2, message security is incorporated into the WSM payload. As long as the Test System can perform necessary security operation off-board and creates the corresponding message payload, the DSRC Proxy will convert security parameters into the DSRC messages frame. This also allows for transmission of WSM/WSA messages with “invalid” security.

The DSRC Proxy will receive and forward to the test system all 802.11 packets including those containing IPv6 packets. The DSRC Proxy will encapsulate them into UDP packets.

Once intercepted by the DSRC Proxy, these packets will not be processed by the radio stack, e.g. IPv6 frames will not be routed.

## Requirements for UDP to Wireless conversion

Upon receipt of the UDP message, the DSRC Proxy will parse the UDP payload, extract parameters and invoke LLC API to create and transmit 802.11 frame. The extracted parameters shall be used to configure radio parameters (i.e. transmit power, data rate, user priority, etc).

|  |  |  |
| --- | --- | --- |
| IP Header | UDP header | UDP payload |

The UDP payload message structure is described in **MsgUDPtoWI** message using ASN.1 in the Appendix A. The UDP payload is encoded using OER encoding [1].

The LLC API will not verify contents of the opaque data copied into 802.11 frame from the UDP message.

For transmission of outbound 802.11 frames, the DSRC Proxy will use radio configuration (radio interface, channel and time slot) from the configuration file. The DSRC Proxy will set Transmit Power, Data Rate, and User Priority using parameters specified in the UDP.

## Requirements for Wireless to UDP conversion

Upon receipt of a 802.11 frame as indicated by LLC API, the DSRC Proxy will parse & extract certain radio parameters. Then, the DSRC Proxy will create a UDP message containing the extract parameters and include the 802.11 frame into the UDP payload. The UDP message will be sent to one or more remote addresses/port.

|  |  |  |
| --- | --- | --- |
| IP Header | UDP header | UDP payload |

UDP payload message structure is defined in **MsgDEVtoUDP** messageusing ASN.1 in the Appendix A.

The UDP payload is encoded using OER encoding.

DSRC Proxy uses the configuration file to setup radio channel access (radio interface, channel, mode). The configuration file also binds the channel access with PSID and remote IP/port. This allows the DSRC Proxy to tune the radio and listen for WSM with specific PSID and forward them to the remote IP/port assigned to the Test System.

## Setting Channel access

It is assumed that DSRC radio may have one or more physical wireless interfaces identified as radio0, radio1, etc. Each radio can be configured in continuous or alternating mode (channel switching). Being in alternating mode, radio can transmit or receive messages during time slot0 or slot1.

Channel Access is defined by configuration parameters including:

* Radio: radio0, radio1, etc
* Channel: 172, 178, etc
* Mode: continuous or alternating
* Time slot (alternating mode only): slot0, slot1, both/either.
  + It is assumed 50%/50% time split between slot0 and slot1.

The Channel Access is set statically in the DSRC Proxy configuration file. Example of the Channel Access configuration is listed below. The corresponding configuration file is listed in the Appendix B.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Wireless interface | Channel | Mode | Time slot | Listen(Port)  Remote Send(IP/Port) |
| radio0 | 172 | Continuous |  | 12172 |
| radio1 | 178 | Alternating | Slot0 | 12178 |
| radio1 | 174 | Alternating | Slot1 | 12174 |
| Destinations address/port for all inbound wireless messages and DSRC proxy exceptions | | | | ::1:20000  127.0.0.1:20001 |

This configuration allows the DSRC Proxy to send outbound wireless messages WSM/WSA/IP on one of 3 channels: channel 172 continously, channel 178 during slot 0, channel 174 during slot 1. Separate UDP ports assigned to access each channel. All inbound wireless messages will be sent to the same two remote address. All exceptions generated by the DSRC Proxy will be sent to the same remote addresses.

## Requirements for sending Exceptions via UDP

This message is designed to send exception information from the DSRC Proxy to the test system. General types of exceptions may include Errors, Warnings, and Information. The DSRC Proxy will send a single UDP message for every exception occurrence. The UDP message will be sent to the remote address and the remote port for exception handling.

|  |  |  |
| --- | --- | --- |
| IP Header | UDP header | UDP payload |

UDP payload message structure is defined in **MsgDEVtoUDP** messageusing ASN.1 in the Appendix A.

The UDP payload is encoded using OER encoding.

The following exceptions must be reported:

|  |  |  |
| --- | --- | --- |
| Exception Type | Exception Id | Exception Description |
| Warning | 1  2 | DSRC Proxy startup  DSRC Proxy shutdown |
| Error | 1  2  3 | WSMP Tx rejected max length exceeded  WSMP Tx rejected – invalid radio configuration  Invalid configuration file |
| Info |  |  |

Additional exceptions can be suggested by the implementor.

## DSRC Proxy user interface

DSRC Proxy can be implemented as a user-space utility. It can be started from a command line or via shell script.

> dsrc\_proxy -c <configuration file>

A sample configuration file is shown in the Appendix B. The sample uses structured ASN.1 representation. Alternative structured ASCII-based formats may be proposed too (e.g. XML, JSON, yaml, and etc).

Only one instance of dsrc\_proxy is expected to run at one time. Each UDP stream and/or PSID may run as a separate threads managed by the single instance of the program.

## Exception handling

All exceptions generated in DSRC Proxy shall be reported to the test system via UDP messages according to the requirements specified previously. Also, those conditions can reported into standard system error output or put into a log file.

Depending on the exception severity, the program may exit or continue its operation with default parameters.

# Appendix A ASN.1 specification for UDP messages and configuration files

DSRCProxy

--<OSS.PDU DSRCProxy.MsgUDPtoWI>--

--<OSS.PDU DSRCProxy.MsgDEVtoUDP>--

--<OSS.PDU DSRCProxy.ProxyAppConfiguration>--

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

IMPORTS

;

MsgUDPtoWI ::= SEQUENCE{

dataRate DataRate80211,

transmitPowerLevel Power80211,

doNotModify DoNotModifyMask OPTIONAL,

data Opaque -- MAC PDU: MAC header + LLC + WSM/IP

-- UserPriority included in 802.11 header QOS TID

}

MsgDEVtoUDP ::= SEQUENCE{

time Time64, -- 64-bit UTC timestamp with microsecond, GPS syncronized

frame FrameType

}

FrameType ::= CHOICE{

wiFrame [0] WiFrame,

expFrame[1] ExpFrame

}

WiFrame ::= SEQUENCE{

channelNumber ChannelNumber80211,

timeSlot TimeSlot,

dataRate DataRate80211,

recvPwrAnt1 Power80211,

recvPwrAnt2 Power80211,

data Opaque

}

ExpFrame ::= SEQUENCE{

exceptionType ExceptionType,

exceptionID ExceptionId OPTIONAL,

module Module OPTIONAL,

exceptionText ExceptionText OPTIONAL

}

/\*

configuration file

\*/

ProxyAppConfiguration ::= SEQUENCE{

version UTF8String (SIZE(0..255)), -- configuration file date/version

description UTF8String (SIZE(0..65535)), -- describes configuration, human readable

transmitWi SEQUENCE OF WiInstance,

transmitUDP SEQUENCE OF TrInstance

}

WiInstance ::= SEQUENCE{

radio RadioInterface,

mode ChannelMode,

antenna Antenna OPTIONAL,

listenUDP ListenUDP

}

TrInstance ::= SEQUENCE{

address IPAddress,

port UDPPort

}

ListenUDP ::= SEQUENCE{

port UDPPort

}

RadioInterface ::= INTEGER{

radio0 (0),

radio1 (1),

radio2 (2),

radio3 (3)

} (0..16)

ChannelMode ::= CHOICE{

alternating [0] AlternatingMode,

continuous [1] ContinuousMode

}

AlternatingMode ::= SEQUENCE{

channel ChannelNumber80211,

timeSlot TimeSlot1

}

ContinuousMode ::= SEQUENCE{

channel ChannelNumber80211

}

Antenna ::= INTEGER{

antenna1 (1),

antenna2 (2)

}

/\*

Common Datatypes

\*/

UDPPort ::= INTEGER(0..65535)

IPAddress ::= UTF8String(SIZE(2..255)) -- IPv4, IPv6 or hostname

TimeSlot ::= INTEGER {

slot0 (0),

slot1 (1),

slotEither (2)

} (0..255)

TimeSlot1 ::= INTEGER {

slot0 (0),

slot1 (1),

both (2)

} (0..255)

UserPriority ::= INTEGER(0..7)

DataRate80211 ::= ENUMERATED {

-- see Table 9-5—Non-HT reference rate 802.11 - 2012 pp864

r6Mbps-12BPSK (1), -- (0xB)

r9Mbps-34BPSK, -- (0xF)

r12Mbps-12QPSK, -- (0xA)

r18Mbps-34QPSK, -- (0xE)

r24Mbps-1216QAM, -- (0x9)

r36Mbps-3416QAM, -- (0xD)

r48Mbps-1264QAM, --

r48Mbps-2364QAM, -- (0x8)

r54Mbps-3464QAM, -- (0xC)

r54Mbps-5664QAM --

}

Power80211 ::= INTEGER(-128..127)

-- dBm units

ChannelNumber80211 ::= INTEGER (0..255)

Opaque ::= OCTET STRING

ExceptionType ::= ENUMERATED{

info, -- all info messages

warning, -- all warnings

error -- all errors

}

Module ::= UTF8String (SIZE(0..255))

-- Functiona/Module reporting an exception

ExceptionId ::= INTEGER (0..65535)

-- exception numerical identifier

ExceptionText ::= UTF8String (SIZE(0..1200))

-- short description of exception

Time64 ::= INTEGER (0..18446744073709551615)

-- 64-bit integer providing system time expressed as the number of microseconds

DoNotModifyMask ::= INTEGER{

seqCtrl (1), --

durationId (2), -- IEEE80211QoSHeader.DurationId

ackPolicy (4), -- IEEE80211QoSHeader.QoSControl.AckPolicy

retry (8) -- etc....

}

END

Sample MsgUDPtoWI

06 15 07 36 88 00 00 00 FF FF FF FF FF FF 04 E5

48 00 10 00 FF FF FF FF FF FF FE FF 24 00 AA AA

03 00 00 00 88 B5 54 68 69 73 20 69 73 20 74 68

65 20 70 61 79 6C 6F 61 64 0A

value1 MsgUDPtoWI ::= {

dataRate 6,

transmitPowerLevel 21,

userPriority 7,

data '88000000FFFFFFFFFFFF04E548001000FFFFFFFFFFFFFEFF2400AAAA0300000088B55468697320697320746865207061796C6F61640A'H}

Sample MsgDEVtoUDP (option – WiFrame)

00 00 01 50 D3 F8 36 D8 80 AC 00 06 BF BE 36 88

00 00 00 FF FF FF FF FF FF 04 E5 48 00 10 00 FF

FF FF FF FF FF FE FF 24 00 AA AA 03 00 00 00 88

B5 54 68 69 73 20 69 73 20 74 68 65 20 70 61 79

6C 6F 61 64 0A

value1 MsgDEVtoUDP ::= {

time 1446665271000,

frame wiFrame : {

channelNumber 172,

timeSlot slot0,

dataRate 6,

recvPwrAnt1 -65,

recvPwrAnt2 -66,

data '88000000FFFFFFFFFFFF04E548001000FFFFFFFFFFFFFEFF2400AAAA0300000088B55468697320697320746865207061796C6F61640A'H }

}

Sample MsgDEVtoUDP (option – EXPFrame)

00 00 01 50 D3 F8 3A C0 81 E0 01 00 01 09 44 53

52 43 50 72 6F 78 79 12 44 53 52 43 20 50 72 6F

78 79 20 73 74 61 72 74 75 70

value1 MsgDEVtoUDP ::= {

time 1446665272000,

frame expFrame : {

exceptionType warning,

exceptionID 1,

module "DSRCProxy",

exceptionText "DSRC Proxy startup"

}

}

# Appendix B DSRC Proxy configuration files

value1 ProxyAppConfiguration ::= {

version "2015-11-04",

description "Sample configuration file",

transmitWi {

{

radio radio0,

mode continuous : {

channel 172

},

listenUDP {

port 12172

}

},

{

radio radio1,

mode alternating : {

channel 178,

timeSlot slot0

},

listenUDP {

port 12178

}

},

{

radio radio1,

mode alternating : {

channel 174,

timeSlot slot1

},

listenUDP {

port 12174

}

}

},

transmitUDP {

{

address "::1",

port 20000

},

{

address "127.0.0.1",

port 20001

}

}

}

## References

[1] “Overview of the Octet Encoding Rules.” Alessandro Triglia, OSS Nokalva, January 2015 <http://www.oss.com/asn1/resources/bookswhitepapers-pubs/Overview%20of%20OER.pdf>

## Revision History

|  |  |  |
| --- | --- | --- |
| V0.1.0 | Oct 22, 2015 | Initial Draft |
| V0.2.0 | Oct 23, 2015 | Updated ASN.1   * Updated PSID encoding for OER * Added MsgEXPtoUDP |
| V0.3.0 | Oct 28,2015 | Added considerations for Channel Access parameters  Added considerations for a configuration file |
| V0.4.0 | Nov 4, 2015 | Interface reworked to LLC API |
| V0.5.0 | Nov 6, 2015 | Clarifications to the Opaque data. Small changes to ASN.1 |
| V0.6.0 | Mar 17, 2016 | Added updated ASN1 as of current implementation (ASN.1 2015-11-19-1.zip) Added Open issues at the beginning of the doc |

◙ End of Document ◙