# P2021\_017\_ILinkRT\_Python\_ESP32

# Lua in Wireshark

Produced by **Olaf Mennerich**

Exported from **HEAD**

Mär 16 2022 14:04

http://localhost:8080/cb/images/newskin/header/cblogo-xl.png

## Table of contents

[P2021\_017\_ILinkRT\_Python\_ESP32 1](#_Toc98331893)

[Lua in Wireshark 1](#_Toc98331894)

[Table of contents 2](#_Toc98331895)

[Lua in Wireshark 3](#_Toc98331896)

[History 3](#_Toc98331897)

[Lua in Wireshark 3](#_Toc98331898)

[Setup 3](#_Toc98331899)

[Setting up the boilerplate code 3](#_Toc98331900)

[Adding fields 5](#_Toc98331901)

[Debugging 6](#_Toc98331902)

[Extending the iLinkRT protocol dissector 8](#_Toc98331903)

### Lua in Wireshark

# History

|  |  |  |
| --- | --- | --- |
| **Version** | **Item** | **date** |
| V1.01 | spelling mistakes corrected | 13.12.2021 |
| V1.0 | initial version | 01.09.2021 |

# Lua in Wireshark

## Setup

If you go to Help –> About Wireshark –> Folders, you’ll find all the folders Wireshark reads Lua scripts from. Choose either the Personal Lua Plugins, Global Lua Plugins or Personal configuration folder. E.g. C:\Program Files\Wireshark\plugins\<version> on Windows. The script will be active when Wireshark is started. You have to restart Wireshark after you do changes to the script, or reload all the Lua scripts with **Ctrl+Shift+L**.

![](data:application/xhtml+xml;base64,)

## Setting up the boilerplate code

Let’s start by setting up some of the boilerplate code that’s needed in all dissectors::

-- iLinkRT\_Proto.lua  
-- Uses V3.0 of the ILinkRT specification  
--  
iLinkRT\_protocol = Proto("iLinkRT", "iLinkRT Protocol")  
  
iLinkRT\_protocol.fields = {}  
  
function iLinkRT\_protocol.dissector(buffer, pinfo, tree)  
 length = buffer:len()  
 if length == 0 then return end  
  
 pinfo.cols.protocol = iLinkRT\_protocol.name  
  
 local subtree = tree:add(iLinkRT\_protocol, buffer(), "iLinkRT Protocol Data")  
end  
  
local udp\_port = DissectorTable.get("udp.port")  
udp\_port:add(8090, iLinkRT\_protocol)

We start by creating a Proto (protocol) object and call it iLinkRT\_protocol. The table constructor takes two arguments: name and description. The protocol requires a fields table and a dissector function. We haven’t added any fields yet, so the fields table is empty. The dissector function is called once for every packet of our type.  
  
The dissector function has three parameters: buffer, pinfo and tree. buffer contains the packet’s buffer and is a Tvb object. It contains the data we want to dissect. pinfo contains the columns of the packet list and is a Pinfo object. Finally, tree is the tree root and is a TreeItem object.

![](data:application/xhtml+xml;base64,)  
  
Inside the dissector function, we start by checking the length of the buffer and then returning if it’s empty.  
  
As mentioned before, the pinfo object contains the columns in the packet list. We can use it to set the protocol name when we receive a packet of iLinkRT type. On the script’s first line we set the name of the protocol to be “iLinkRT” (by passing the name to the constructor). We set the protocol column name here.

pinfo.cols.protocol = mongodb\_protocol.name

and the protocol column name changes from TCP to ILINKRT:  
  
We must then create a sub tree in the tree structure found in the Packet Details pane. It done by adding an additional tree item to the tree object that was passed as an argument to the dissector function.

local subtree = tree:add(iLinkRT\_protocol, buffer(), "iLinkRT Protocol Data")

Finally, we must assign the protocol to a port. In my case, I’ll use port 8090, because that’s the port I use to connect to the Mongo database.

local udp\_port = DissectorTable.get("udp.port")  
 udp\_port:add(8090, iLinkRT\_protocol)

## Adding fields

The script already runs at this stage, but it isn’t doing anything useful. For the script to do something useful we have to add the fields that we want to parse. Fields are made by creating ProtoField objects. We can start off simple by adding only the first field. The first field in the iLinkRT protocol specification is the message length, which is an int16.

-- iLinkRT\_Proto.lua  
-- Uses V3.0 of the ILinkRT specification  
--  
iLinkRT\_protocol = Proto("iLinkRT", "iLinkRT Protocol")  
  
message\_length = ProtoField.int16("ilinkrt.message\_length", "messageLength", base.DEC)  
  
iLinkRT\_protocol.fields = {message\_length}  
  
function iLinkRT\_protocol.dissector(buffer, pinfo, tree)  
 length = buffer:len()  
 if length == 0 then return end  
  
 pinfo.cols.protocol = iLinkRT\_protocol.name  
  
 local subtree = tree:add(iLinkRT\_protocol, buffer(), "iLinkRT Protocol Data")  
  
 subtree:add\_le(message\_length, buffer(0,2))  
end  
  
local udp\_port = DissectorTable.get("udp.port")  
udp\_port:add(8090, iLinkRT\_protocol)

We add the following above the dissector function:

message\_length = ProtoField.int16("ilinkrt.message\_length", "messageLength", base.DEC)

The first argument is used as a label in the filter settings, second is used as a label in the sub tree and the last is used to decide how the variable’s value should be displayed. In this case I want to show the value in decimal, but I could also use base.HEX to show it in hexadecimal format. Hexadecimal format doesn’t work for int32 though.  
  
ProtoField has several types of functions we can use: uint8(), uint16(), string() and so on. We have to use the one that matches the specification. A list of all the functions can be found [here](https://www.wireshark.org/docs/wsdg_html_chunked/lua_module_Proto.html#lua_fn_ProtoField_char_abbr___name____base____valuestring____mask____desc__) .  
  
We then add the field to the fields table of the protocol:

iLinkRT\_protocol.fields = {message\_length}

and finally add the field to the sub tree:

subtree:add\_le(message\_length, buffer(0,2))

I use add\_le rather than add, because we are working with a little-endian protocol. If the protocol was big endian we would have to use add. The function takes two arguments: the field we made further up, and a buffer range. We can get a range of the buffer by using the range function that is a part of the buffer object. buffer(offset,length) is the short form for the range function. buffer(0,2) means we want to start at the first byte, and then take 2 bytes. The reason we want to start at 0 is because we’re dealing with the first field in the header. We take 2 bytes because that is the size of an int16.

## Debugging

When I’m talking about debugging I am not really thinking of debugging the normal way, where you use a symbolic debugger to step through code. You won’t do that here. I am rather thinking about the process of finding and fixing errors in the code. There are generally three ways to debug dissectors written in Lua.  
  
The first is to check if you get any error messages during startup of the script. This happens either when you start Wireshark or when you reload the script with **Ctrl+Shift+L**. Syntax errors in the script will be caught this way. Here is what an error message looks like when an end statement is missing:

![](data:application/xhtml+xml;base64,)

Runtime errors are often shown in the sub tree for the dissector. For example, if a function is called with the wrong name the error message will look like this.

![](data:application/xhtml+xml;base64,)  
  
Finally, Wireshark has a Lua console built in that you can print error messages to. It’s found in the Tools → Lua menu. Wireshark has a function called print() that can be used for logging. So the following code:

print ("Number of messages: " .. num\_of\_msg)

will end up looking like this when printed to the console:   
  
  
![](data:application/xhtml+xml;base64,)  
  
where .. is used for string concatenation.

## Extending the iLinkRT protocol dissector

In the previous post we made a dissector that ended up looking like this in the packet details pane:  
  
  
  
![](data:application/xhtml+xml;base64,)  
  
The Command code is only a number. It would be nicer if we showed the opcode name too. According to the iLinkRT wire protocol, the Command code have the following names:

|  |  |  |
| --- | --- | --- |
| **Command HEX** | **Command DEC** | **Caommand Name** |
| FFFF | 65535 | RT\_GET\_ALL\_SERVER |
| 0100 | 256 | RT\_GET\_SERVER\_STATE |
| 0101 | 257 | RT\_GET\_SERVER\_TIME |
| 0102 | 258 | RT\_SERVER\_CONNECT |
| 0103 | 259 | RT\_SERVER\_DISCONNECT |
| 0200 | 512 | RT\_GET\_CALPAGE\_INFO |
| 0201 | 513 | RT\_GET\_CHARACTERISTIC\_ID\_LIST |
| 0202 | 514 | RT\_GET\_CHARACTERISTIC\_INFO |
| 0203 | 515 | RT\_GET\_DAQ\_RESOLUTION\_INFO |
| 0204 | 516 | RT\_GET\_DEVICE\_INFO |
| 0205 | 517 | RT\_GET\_DEVICE\_STATE |
| 0206 | 518 | RT\_GET\_MEASUREMENT\_ID\_LIST |
| 0207 | 519 | RT\_GET\_MEASUREMENT\_INFO |
| 0208 | 520 | RT\_GET\_RASTER\_OVERVIEW |
| 0209 | 521 | RT\_GET\_SELECTED\_DEVICES |
| 0300 | 768 | RT\_CHANGE\_DESCRIPTION\_FILE |
| 0301 | 769 | RT\_CHANGE\_HEX\_FILE |
| 0302 | 770 | RT\_CONFIGURE\_SERVER |
| 0303 | 771 | RT\_COPY\_DATA\_EXCHANGE\_FILE\_TO\_DEVICE |
| 0304 | 772 | RT\_DEVICE\_CONNECT |
| 0305 | 773 | RT\_DISTRIBUTE\_EVENT |
| 0306 | 774 | RT\_SAVE\_HEX\_FILE |
| 0307 | 775 | RT\_SELECT\_CHARACTERISTIC\_ID |
| 0308 | 776 | RT\_SELECT\_DEVICE |
| 0309 | 777 | RT\_SELECT\_DEVICE\_SET |
| 030A | 778 | RT\_SELECT\_MEASUREMENT\_ID |
| 0400 | 1024 | RT\_CLEAR\_MEASURING\_LIST |
| 0401 | 1025 | RT\_CONFIGURE\_MEASURING |
| 0402 | 1026 | RT\_GET\_DAQ\_EVENT\_INFO |
| 0403 | 1027 | RT\_GET\_DAQ\_MEASUREMENT\_LIST |
| 0404 | 1028 | RT\_GET\_DEVICE\_DAQ\_LIST |
| 0405 | 1029 | RT\_START\_STOP\_MEASURING |
| 0500 | 1030 | RT\_GET\_CALPAGE |
| 0501 | 1031 | RT\_READ\_CELL\_VALUES |
| 0502 | 1032 | RT\_READ\_CHARACTERISTIC |
| 0503 | 1033 | RT\_SET\_CALPAGE |
| 0504 | 1034 | RT\_WRITE\_CELL\_VALUES |
| 0505 | 1035 | RT\_WRITE\_CHARACTERISTIC |
| 0600 | 1536 | RT\_ADD\_KEY\_VALUE\_PAIR\_TO\_RECORDER\_FILE |
| 0601 | 1537 | RT\_CONFIGURE\_RECORDER |
| 0602 | 1538 | RT\_CONTROL\_RECORDER |
| 0603 | 1539 | RT\_GET\_RETRIGGERING |
| 0604 | 1540 | RT\_GET\_TRIGGER |
| 0605 | 1541 | RT\_SET\_CLIENT\_BOOKMARK |
| 0606 | 1542 | RT\_SET\_RETRIGGERING |
| 0607 | 1543 | RT\_SET\_TRIGGER |
| 0700 | 1792 | RT\_EXECUTE\_SERVICE |
| 0701 | 1793 | RT\_ GET\_AVAILABLE\_CHARACTERISTICS |
| 0702 | 1794 | RT\_ GET\_AVAILABLE\_DEVICE\_SETS |
| 0703 | 1795 | RT\_ GET\_AVAILABLE\_DEVICES |
| 0704 | 1796 | RT\_ GET\_AVAILABLE\_MEASUREMENTS |

To grab the command as an integer we can use

local cmd = buffer(8,2):le\_uint()

le\_int() gets a little endian int from the buffer range. The variable cmd now contains an int representing the command in decimal. We can then make a function that returns the command name given the command number:

function get\_cmd\_name(cmd)  
 local cmd\_name = "Unknown"  
  
 if cmd == 65535 then cmd\_name = "RT\_GET\_ALL\_SERVER"  
 elseif cmd == 256 then cmd\_name = "RT\_GET\_SERVER\_STATE"  
 elseif cmd == 257 then cmd\_name = "RT\_GET\_SERVER\_TIME"  
 elseif cmd == 258 then cmd\_name = "RT\_SERVER\_CONNECT"  
 elseif cmd == 259 then cmd\_name = "RT\_SERVER\_DISCONNECT"  
 elseif cmd == 512 then cmd\_name = "RT\_GET\_CALPAGE\_INFO"  
 elseif cmd == 513 then cmd\_name = "RT\_GET\_CHARACTERISTIC\_ID\_LIST"  
 elseif cmd == 514 then cmd\_name = "RT\_GET\_CHARACTERISTIC\_INFO"  
 elseif cmd == 515 then cmd\_name = "RT\_GET\_DAQ\_RESOLUTION\_INFO"  
 elseif cmd == 516 then cmd\_name = "RT\_GET\_DEVICE\_INFO"  
 elseif cmd == 517 then cmd\_name = "RT\_GET\_DEVICE\_STATE"  
 elseif cmd == 518 then cmd\_name = "RT\_GET\_MEASUREMENT\_ID\_LIST"  
 elseif cmd == 519 then cmd\_name = "RT\_GET\_MEASUREMENT\_INFO"  
 elseif cmd == 520 then cmd\_name = "RT\_GET\_RASTER\_OVERVIEW"  
 elseif cmd == 521 then cmd\_name = "RT\_GET\_SELECTED\_DEVICES"  
 elseif cmd == 768 then cmd\_name = "RT\_CHANGE\_DESCRIPTION\_FILE"  
 elseif cmd == 769 then cmd\_name = "RT\_CHANGE\_HEX\_FILE"  
 elseif cmd == 770 then cmd\_name = "RT\_CONFIGURE\_SERVER"  
 elseif cmd == 771 then cmd\_name = "RT\_COPY\_DATA\_EXCHANGE\_FILE\_TO\_DEVICE"  
 elseif cmd == 772 then cmd\_name = "RT\_DEVICE\_CONNECT"  
 elseif cmd == 773 then cmd\_name = "RT\_DISTRIBUTE\_EVENT"  
 elseif cmd == 774 then cmd\_name = "RT\_SAVE\_HEX\_FILE"  
 elseif cmd == 775 then cmd\_name = "RT\_SELECT\_CHARACTERISTIC\_ID"  
 elseif cmd == 776 then cmd\_name = "RT\_SELECT\_DEVICE"  
 elseif cmd == 777 then cmd\_name = "RT\_SELECT\_DEVICE\_SET"  
 elseif cmd == 778 then cmd\_name = "RT\_SELECT\_MEASUREMENT\_ID"  
 elseif cmd == 1024 then cmd\_name = "RT\_CLEAR\_MEASURING\_LIST"  
 elseif cmd == 1025 then cmd\_name = "RT\_CONFIGURE\_MEASURING"  
 elseif cmd == 1026 then cmd\_name = "RT\_GET\_DAQ\_EVENT\_INFO"  
 elseif cmd == 1027 then cmd\_name = "RT\_GET\_DAQ\_MEASUREMENT\_LIST"  
 elseif cmd == 1028 then cmd\_name = "RT\_GET\_DEVICE\_DAQ\_LIST"  
 elseif cmd == 1029 then cmd\_name = "RT\_START\_STOP\_MEASURING"  
 elseif cmd == 1030 then cmd\_name = "RT\_GET\_CALPAGE"  
 elseif cmd == 1031 then cmd\_name = "RT\_READ\_CELL\_VALUES"  
 elseif cmd == 1032 then cmd\_name = "RT\_READ\_CHARACTERISTIC"  
 elseif cmd == 1033 then cmd\_name = "RT\_SET\_CALPAGE"  
 elseif cmd == 1034 then cmd\_name = "RT\_WRITE\_CELL\_VALUES"  
 elseif cmd == 1035 then cmd\_name = "RT\_WRITE\_CHARACTERISTIC"  
 elseif cmd == 1536 then cmd\_name = "RT\_ADD\_KEY\_VALUE\_PAIR\_TO\_RECORDER\_FILE"  
 elseif cmd == 1537 then cmd\_name = "RT\_CONFIGURE\_RECORDER"  
 elseif cmd == 1538 then cmd\_name = "RT\_CONTROL\_RECORDER"  
 elseif cmd == 1539 then cmd\_name = "RT\_GET\_RETRIGGERING"  
 elseif cmd == 1540 then cmd\_name = "RT\_GET\_TRIGGER"  
 elseif cmd == 1541 then cmd\_name = "RT\_SET\_CLIENT\_BOOKMARK"  
 elseif cmd == 1542 then cmd\_name = "RT\_SET\_RETRIGGERING"  
 elseif cmd == 1543 then cmd\_name = "RT\_SET\_TRIGGER"  
 elseif cmd == 1792 then cmd\_name = "RT\_EXECUTE\_SERVICE"  
 elseif cmd == 1793 then cmd\_name = "RT\_ GET\_AVAILABLE\_CHARACTERISTICS"  
 elseif cmd == 1794 then cmd\_name = "RT\_ GET\_AVAILABLE\_DEVICE\_SETS"  
 elseif cmd == 1795 then cmd\_name = "RT\_ GET\_AVAILABLE\_DEVICES"  
 elseif cmd == 1796 then cmd\_name = "RT\_ GET\_AVAILABLE\_MEASUREMENTS"  
 end  
  
 return cmd\_name  
end

Finally, we have to replace the old addition to the sub tree with the following code:

local cmd\_name = get\_cmd\_name(cmd)  
 subtree:add\_le(command, buffer(8,2)):append\_text(" (" .. cmd\_name .. ")")

We append the name of the name of the command in parentheses to the original statement that only showed command number. The packet details pane in Wireshark will then end up looking like this:

![](data:application/xhtml+xml;base64,)  
  
With the final code being:

-- iLinkRT\_Proto.lua  
-- Uses V3.0 of the ILinkRT specification  
--  
iLinkRT\_protocol = Proto("iLinkRT", "iLinkRT Protocol")  
  
message\_length = ProtoField.int16("ilinkrt.message\_length", "messageLength", base.DEC)  
message\_ctr = ProtoField.int16("ilinkrt.message\_ctr", "messageCounter", base.DEC)  
message\_cf1 = ProtoField.int16("ilinkrt.message\_cf1", "messageFirstControl", base.DEC)  
message\_fill1 = ProtoField.int16("ilinkrt.message\_fill1", "messagefillBytes", base.DEC)  
message\_cf2 = ProtoField.int16("ilinkrt.message\_cf2", "messageSecondControl", base.DEC)  
command = ProtoField.int16("ilinkrt.command", "command", base.DEC)  
  
num\_of\_msg = 0  
  
iLinkRT\_protocol.fields = {message\_length, message\_ctr, message\_cf1, message\_fill1, message\_cf2, command }  
  
function iLinkRT\_protocol.dissector(buffer, pinfo, tree)  
 length = buffer:len()  
 if length == 0 then return end  
  
  
 --num\_of\_msg = num\_of\_msg +1  
 --print ("Number of messages: " .. num\_of\_msg)  
  
  
  
 pinfo.cols.protocol = iLinkRT\_protocol.name  
  
 local subtree = tree:add(iLinkRT\_protocol, buffer(), "iLinkRT Protocol Data")  
  
 subtree:add\_le(message\_length, buffer(0,2))  
 subtree:add\_le(message\_ctr, buffer(2,2))  
 subtree:add\_le(message\_cf1, buffer(4,2))  
 subtree:add\_le(message\_fill1, buffer(6,2))  
 subtree:add\_le(message\_cf2, buffer(length-2,2))  
  
 local cmd = buffer(8,2):le\_uint()  
 local cmd\_name = get\_cmd\_name(cmd)  
 subtree:add\_le(command, buffer(8,2)):append\_text(" (" .. cmd\_name .. ")")  
  
end  
  
  
function get\_cmd\_name(cmd)  
 local cmd\_name = "Unknown"  
  
 if cmd == 65535 then cmd\_name = "RT\_GET\_ALL\_SERVER"  
 elseif cmd == 256 then cmd\_name = "RT\_GET\_SERVER\_STATE"  
 elseif cmd == 257 then cmd\_name = "RT\_GET\_SERVER\_TIME"  
 elseif cmd == 258 then cmd\_name = "RT\_SERVER\_CONNECT"  
 elseif cmd == 259 then cmd\_name = "RT\_SERVER\_DISCONNECT"  
 elseif cmd == 512 then cmd\_name = "RT\_GET\_CALPAGE\_INFO"  
 elseif cmd == 513 then cmd\_name = "RT\_GET\_CHARACTERISTIC\_ID\_LIST"  
 elseif cmd == 514 then cmd\_name = "RT\_GET\_CHARACTERISTIC\_INFO"  
 elseif cmd == 515 then cmd\_name = "RT\_GET\_DAQ\_RESOLUTION\_INFO"  
 elseif cmd == 516 then cmd\_name = "RT\_GET\_DEVICE\_INFO"  
 elseif cmd == 517 then cmd\_name = "RT\_GET\_DEVICE\_STATE"  
 elseif cmd == 518 then cmd\_name = "RT\_GET\_MEASUREMENT\_ID\_LIST"  
 elseif cmd == 519 then cmd\_name = "RT\_GET\_MEASUREMENT\_INFO"  
 elseif cmd == 520 then cmd\_name = "RT\_GET\_RASTER\_OVERVIEW"  
 elseif cmd == 521 then cmd\_name = "RT\_GET\_SELECTED\_DEVICES"  
 elseif cmd == 768 then cmd\_name = "RT\_CHANGE\_DESCRIPTION\_FILE"  
 elseif cmd == 769 then cmd\_name = "RT\_CHANGE\_HEX\_FILE"  
 elseif cmd == 770 then cmd\_name = "RT\_CONFIGURE\_SERVER"  
 elseif cmd == 771 then cmd\_name = "RT\_COPY\_DATA\_EXCHANGE\_FILE\_TO\_DEVICE"  
 elseif cmd == 772 then cmd\_name = "RT\_DEVICE\_CONNECT"  
 elseif cmd == 773 then cmd\_name = "RT\_DISTRIBUTE\_EVENT"  
 elseif cmd == 774 then cmd\_name = "RT\_SAVE\_HEX\_FILE"  
 elseif cmd == 775 then cmd\_name = "RT\_SELECT\_CHARACTERISTIC\_ID"  
 elseif cmd == 776 then cmd\_name = "RT\_SELECT\_DEVICE"  
 elseif cmd == 777 then cmd\_name = "RT\_SELECT\_DEVICE\_SET"  
 elseif cmd == 778 then cmd\_name = "RT\_SELECT\_MEASUREMENT\_ID"  
 elseif cmd == 1024 then cmd\_name = "RT\_CLEAR\_MEASURING\_LIST"  
 elseif cmd == 1025 then cmd\_name = "RT\_CONFIGURE\_MEASURING"  
 elseif cmd == 1026 then cmd\_name = "RT\_GET\_DAQ\_EVENT\_INFO"  
 elseif cmd == 1027 then cmd\_name = "RT\_GET\_DAQ\_MEASUREMENT\_LIST"  
 elseif cmd == 1028 then cmd\_name = "RT\_GET\_DEVICE\_DAQ\_LIST"  
 elseif cmd == 1029 then cmd\_name = "RT\_START\_STOP\_MEASURING"  
 elseif cmd == 1030 then cmd\_name = "RT\_GET\_CALPAGE"  
 elseif cmd == 1031 then cmd\_name = "RT\_READ\_CELL\_VALUES"  
 elseif cmd == 1032 then cmd\_name = "RT\_READ\_CHARACTERISTIC"  
 elseif cmd == 1033 then cmd\_name = "RT\_SET\_CALPAGE"  
 elseif cmd == 1034 then cmd\_name = "RT\_WRITE\_CELL\_VALUES"  
 elseif cmd == 1035 then cmd\_name = "RT\_WRITE\_CHARACTERISTIC"  
 elseif cmd == 1536 then cmd\_name = "RT\_ADD\_KEY\_VALUE\_PAIR\_TO\_RECORDER\_FILE"  
 elseif cmd == 1537 then cmd\_name = "RT\_CONFIGURE\_RECORDER"  
 elseif cmd == 1538 then cmd\_name = "RT\_CONTROL\_RECORDER"  
 elseif cmd == 1539 then cmd\_name = "RT\_GET\_RETRIGGERING"  
 elseif cmd == 1540 then cmd\_name = "RT\_GET\_TRIGGER"  
 elseif cmd == 1541 then cmd\_name = "RT\_SET\_CLIENT\_BOOKMARK"  
 elseif cmd == 1542 then cmd\_name = "RT\_SET\_RETRIGGERING"  
 elseif cmd == 1543 then cmd\_name = "RT\_SET\_TRIGGER"  
 elseif cmd == 1792 then cmd\_name = "RT\_EXECUTE\_SERVICE"  
 elseif cmd == 1793 then cmd\_name = "RT\_ GET\_AVAILABLE\_CHARACTERISTICS"  
 elseif cmd == 1794 then cmd\_name = "RT\_ GET\_AVAILABLE\_DEVICE\_SETS"  
 elseif cmd == 1795 then cmd\_name = "RT\_ GET\_AVAILABLE\_DEVICES"  
 elseif cmd == 1796 then cmd\_name = "RT\_ GET\_AVAILABLE\_MEASUREMENTS"  
 end  
  
 return cmd\_name  
end  
  
  
local udp\_port = DissectorTable.get("udp.port")  
udp\_port:add(8090, iLinkRT\_protocol)