**CERN** Openlab project

# Implementing the network debugging infrastructure for the new detector readout board

Christina Quast

October 12, 2015

CERN - LHCb

Supervisor: Niko Neufeld, Rainer Schwemmer

# Contents

1	<b>Intr</b> 1.1 1.2	oduction         Detector board to userspace interface	<b>2</b> 4 5
2	Imp	lementation	7
	2.1	Wireshark basics	7
	2.2	Implementation of a wireshark dissector in C	7
		2.2.1 Basic dissector functions and structures	7
		2.2.1.1 Example code for example protocol	7
		2.2.2 Important wireshark dissector functions	9
3	Con	clusion	12
	3.1	Accomplished tasks	12
	3.2	Known issues	12
	3.3	To be done in the future	12

# List of Figures

1.1	AMC40 board (left), PCIe40 board (right)	2
1.2	Readout board connection to detectors and network	3
1.3	Wireshark with filtering for http traffic and IP source address	3
1.4	MEP protocol	6
2.1	Example protocol structure	7
2.2	Wireshark without MEP dissector plugin	10
2.3	Wireshark with MEP dissector plugin	11

### **1** Introduction

For the next run of the LHCb experiment, new detectors are built, which use a different protocol from the one used for the old detectors in order to send the data collected from a collision. The data throughput will increase from the currently used 400 Gbit per second to astonishing 40 Tbit. The LHCb upgrade team built two FPGA boards: AMC40 and PCIe40 (see image 1.1). The AMC40 board has an Ethernet interface, produces UDP packets and sends them over the network. The PCIe40 board sends data over PCIe to a server, which in turn creates the network packets based on the PCIe data received, and sends them over the network. Each board is connected to the detectors and contains an FPGA, which does the preprocessing of the data received from the detector. The readout system is connected to the network and sends network packets to the backbone (see image 1.2). The readout system will operate at an overall speed of around 40 Tbit/s. In order to have a means of debugging all this network traffic, tools are necessary. The implementation of such a tool is the task for this Openlab Summer Student project.

A very commonly used tool when it comes to debugging network traffic is called Wireshark. It can display the network packet flow and dissect protocols, creating a neat view of the fields in the header of each layer of protocols. Furthermore, it allows for filtering of different protocol fields (e.g. the protocol http and a certain IP address of interest, see image 1.3). If there is an error in dissecting the packet, e.g. because the packet is corrupted and therefore does not conform to the standard form anymore, Wireshark will display an error message. Wireshark supports a large amount of protocols, but also provides a means for adding custom protocol dissectors by compiling them into the Wireshark code or adding them separately as plugins.

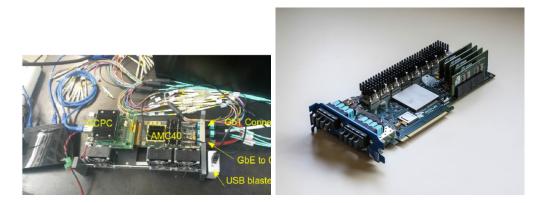
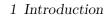


Figure 1.1: AMC40 board (left), PCIe40 board (right)



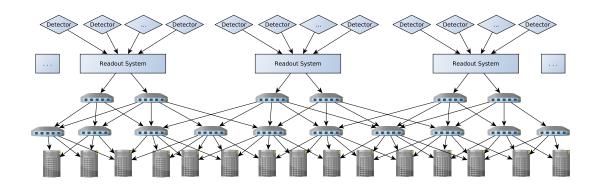


Figure 1.2: Readout board connection to detectors and network

0030       00       82       39       16       00       00       01       01       08       0a       05       c9       62       7a       03       db      9bz         0040       62       dc       48       54       50       2f       31       20       32       30       30       20       4f       b.HTTP/1       .1       200       0         0050       4b       od       a       35       72       76       65       72       3a       20       6e       67       69       6e       78       K.Serve r: nginx		atistics Telephony Tools	incontato incip				
No.       Time       Source       Destination       Protocol Length       Info         82 6.700012 62.146.26.38       128.141.187.149       TCP       74 80-36010       [SYN, ACK] Seq:         85 6.800015 62.146.26.38       128.141.187.149       TCP       66 80-36010       [ACK] Seq:         86 6.800055 62.146.26.38       128.141.187.149       HTTP       450 HTTP/1.1 304 Not Modifit         119 9.600001 62.146.26.38       128.141.187.149       HTTP/XML       649 HTTP/1.1 200 OK         125 9.900054 (62.146.26.38       128.141.187.149       HTP       2962       [TCP segment of a reass         • Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18)       • Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149       (128.141.187.149         • Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010)       [Stream index: 2]       [TCP Segment Len: 583]         Sequence number: 365 (relative sequence number)       [Next sequence number: 968 (relative ack number)]       •bz         Modd S2 de 48 54 54 50 27 31 22 81 20 32 20 30 30 20 4f       •bz       •bz         0030 00 82 39 16 00 00 01 10 80 80 59 62 73 80 38      Pbz       •bz         0030 00 62 de 48 54 54 50 27 31 22 81 20 32 30 30 20 4f       •bz       •b	• • 🖌 📕 🔬 🖿 🛅	X G Q &	~ * ~ ± EE	•			
82 6.700012t 62.146.26.38       128.141.187.149       TCP       74 80-36010 [SYN, ACK] Seq:         85 6.800015t 62.146.26.38       128.141.187.149       TCP       66 80-36010 [ACK] Seq=1 Ack         86 6.80005t 62.146.26.38       128.141.187.149       HTTP       450 HTTP/1.1 304 Not Modifix         119 9.600001t 62.146.26.38       128.141.187.149       HTTP/XML       649 HTTP/1.1 200 0K         125 9.900054t 62.146.26.38       128.141.187.149       TCP       2662 [TCP segment of a reass         • Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18)       Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144         • Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010)       [Stream index: 2]         [TCP Segment Len: 583]       Sequence number: 365 (relative sequence number)         [Next sequence number: 268 (relative sequence number)       .9bz         Header Length: 32 bytes       .9.23 0 30 0 20 41       .9bz         0030 00 85 09 50 80 ca aa b f0 84 5e 23 de d5 97 80 18      P*#       .9bz         0030 00 85 299 16 00 00 01 00 80 a0 5c 96 27 a0 30 80       .9bz       .9bz         0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 40       .9bz       .9bz         0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 40	Filter: ip.src == 62.146.26.38 && http	▼ Exp	pression Clear Apply Save				
85 6.800015( 62.146.26.38       128.141.187.149       TCP       66 80-36010 [ACK] Seq=1 Acl         86 6.800055( 62.146.26.38       128.141.187.149       HTTP       450 HTTP/1.1 304 Not Modifit         119 9.600001( 62.146.26.38       128.141.187.149       HTTP/XML       649 HTTP/1.1 200 0K         125 9.900054( 62.146.26.38       128.141.187.149       TCP       2962 [TCP segment of a reass         • Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18)       Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.145         • Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len       Source Port: 80 (80)         Destination Port: 36010 (36010)       [Stream index: 2]       [TCP Segment Len: 583]         Sequence number: 365 (relative sequence number)       [Next sequence number: 968 (relative ack number)       [.9bz.]         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18      P	No. Time Source	Destination	Protocol Length Info				
86 6.800055( 62.146.26.38 128.141.187.149 HTTP 450 HTTP/1.1 304 Not Modifit 119 9.600001( 62.146.26.38 128.141.187.149 HTTP/XML 649 HTTP/1.1 200 0K 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 125 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing 126 9.900054( 62.146.26.38 128.141.187.149 TCP 2962 [TCP segment of a reassing) 127 9.900 Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144 Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len Source Port: 80 (80) Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number)] Acknowledgment number: 1060 (relative sequence number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18Ppz 0040 62 de 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f D.HTTP/1 .1 200 0 0050 4b 04 0a 33 65 72 76 65 72 3a 20 6e 76 96 e7 8 KServe r: nginx 0060 04 04 46 174 65 3a 20 4d 6f 6e 22 00 32 34 0Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 04 0a 43 6f 6e 74 65 6e 74 2d 54 1 GMT C ontent.T Frame (649 bytes) De-chunked entity body (211 bytes)	82 6.700012( 62.146.26.38	128.141.187.149					
119 9.600001 62.146.26.38       128.141.187.149       HTTP/XML       649       649       HTTP/1.1 200 0K         125 9.900054 62.146.26.38       128.141.187.149       TCP       2962 [TCP segment of a reass         * Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18)         * Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144         * Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len         Source Port: 80 (80)         Destination Port: 36010 (36010)         [Stream index: 2]         [TCP Segment Len: 583]         Sequence number: 385 (relative sequence number)         [Next sequence number: 1060 (relative ack number)         Header Length: 32 bytes         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18        ,P,*#         0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db        ,P,*#         0040 62 de 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f         b.HTTP/1.1 200 0         0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 67 8 KServe r: nginx         0060 01 01 46 6e 2c 20 32 34 20      Det: Mon, 24         0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30       Aug 2015 15:50:0							
125 9.900054(62.146.26.38       128.141.187.149       TCP       2962 [TCP segment of a reass         • Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18)         • Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144         • Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len         Source Port: 80 (80)         Destination Port: 36010 (36010)         [Stream index: 2]         [TCP Segment Len: 583]         Sequence number: 385 (relative sequence number)         [Next sequence number: 1060 (relative ack number)         Header Length: 32 bytes         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18         0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db         0040 62 de 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f         b. HTTP/1 .1 200 0         0050 4b od 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78         K.Serve r: nginx         0060 01 23 30 13 20 31 35 20 31 35 3a 35 30 3a 30         Aug 2015 15:50:0         0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54         De-chunked entity body (211 bytes)				odifi			
Ethernet II, Src: Hewlettd7:81:00 (00:18:71:d7:81:00), Dst: IntelCor_06:e3:18 (00:1e:64:06:e3:18) Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.145 Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len Source Port: 80 (80) Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes          0020       bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18Pbz 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 dbPbz 0040 62 dd 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f b. HTTP/1.1 200 0 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 d5 5e 74 d5 4 l GMT C content.T Frame (649 bytes)							
<pre>&gt; Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144 Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len Source Port: 80 (80) Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P^# 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 30 33 00 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 1 GMT_C ontent.T Frame (649 bytes) De-chunked entity body (211 bytes)</pre>		128.141.187.149	ICP 2962 LICP segment of a r	eass(			
<pre>&gt; Internet Protocol Version 4, Src: 62.146.26.38 (62.146.26.38), Dst: 128.141.187.149 (128.141.187.144 Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len Source Port: 80 (80) Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P^# 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 30 33 00 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 1 GMT_C ontent.T Frame (649 bytes) De-chunked entity body (211 bytes)</pre>	Ethernet II Src: Hewlett, d7:81:00	(00:18:71:d7:81:00) D	0st: IntelCor 06:03:18 (00:10:64:06:03	2.18)			
<pre>Transmission Control Protocol, Src Port: 80 (80), Dst Port: 36010 (36010), Seq: 385, Ack: 1060, Len Source Port: 80 (80) Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P ^# 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 0 Heater Length: 32 bytes 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:S0:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 LOME Content-T Frame (649 bytes) De-chunked entity body (211 bytes) </pre>							
Destination Port: 36010 (36010) [Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P ^# 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 dbP ^# 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f b.HTTP/1 .1 200 0 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 1 GMT C ontent.T Frame (649 bytes) Pe-chunked entity body (211 bytes)							
[Stream index: 2] [TCP Segment Len: 583] Sequence number: 385 (relative sequence number) [Next sequence number: 968 (relative sequence number)] Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P ^# 0030 00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 dbPbz. 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f b.HTTP/1 .1 200 0 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 1 GMT C ontent.T Frame (649 bytes) De-chunked entity body (211 bytes)							
ITCP Segment Len: 583]         Sequence number: 385 (relative sequence number)         [Next sequence number: 968 (relative sequence number)]         Acknowledgment number: 1060 (relative ack number)         Header Length: 32 bytes         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P ^#         0030 008 23 916 00 00 01 01 08 0a 05 c9 62 7a 03 db         0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f         D.HTTP/1 .1 200 0         0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78         KServe r: nginx         0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20         0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30         Aug 2015 15:50:0         0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 1 CML C ontent-T         Frame (649 bytes)	Destination Port: 36010 (36010)						
Sequence number: 385 (relative sequence number)         [Next sequence number: 968 (relative sequence number)]         Acknowledgment number: 1060 (relative ack number)         Header Length: 32 bytes         0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18 (P			[Stream index: 2]				
[Next sequence number: 968 Acknowledgment number: 1060 Header Length: 32 bytes       (relative sequence number)] (relative ack number)         0020       bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18 0030      P/#         0020       bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18 0030      P/#         0030       00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db 0040      P/#         0040       62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 0070 41 75 67 20 32 30 31 35 20 31 35 30 33 30 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 2d 54 1 CML C ontent-T         Frame (649 bytes)							
Acknowledgment number: 1060 (relative ack number) Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18P ^# 0030 00 82 39 16 00 00 10 10 88 0a 05 c9 62 7a 03 db 0040 62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f b HTTP/1 .1 200 0 0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 KServe r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54 1 GMT C ontent-T Frame (649 bytes) De-chunked entity body (211 bytes)	5						
Header Length: 32 bytes         0020       bb 95 00 50 8c aa ab f0 84 5e 23 de d5 97 80 18      P^#         0030       00 82 39 16 00 00 01 01 08 0a 05 c9 62 7a 03 db      Pbz         0040       62 dc 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f       b.HTTP/1 .1 200 0         0050       4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78       KServe r: nginx         0060       00 a4 46 17 46 5 3a 20 4d 6f 6e 2c 20 32 34 20      Date: Mon, 24         0070       41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30       Aug 2015 15:50:0         0080       31 20 47 4d 54 0d 0a 43 6f 6e 74 2d 54       1 GMT_C ontent-T         Frame (649 bytes)	Sequence number: 385 (relative	•					
0020       bb 95       00       50       8c aa ab f0       84       5e 23       de d5       97       80       18      P/#         0030       00       82       39       16       00       00       10       08       0a 05       c9       62       7a       03       db      Pbz         0040       62       dc       48       54       54       50       2f       31       2e       31       20       30       20       4f       b.HTTP/1       1       200       0         0050       4b       od       0a       53       65       72       3a       20       6e       67       69       67       8       KServe r: nginx      bez.         0060       0d       0a       44       6f       6e       22       23       30       24       20      bez.      bez.         0070       41       75       67       20       32       30       31       35       30       33       30       Aug 2015       15:50:0      bez.      bez.      bez.      bez.      bez.      bez.      bez.      bez.      bez. <td>Sequence number: 385 (relative [Next sequence number: 968 (re</td> <td>lative sequence number)</td> <td>)]</td> <td></td>	Sequence number: 385 (relative [Next sequence number: 968 (re	lative sequence number)	)]				
0030       00       82       39       16       00       00       10       08       0a       05       c9       62       7a       03       db      9bz         0040       62       dc       48       54       50       2f       31       2e       31       20       32       30       20       4f       b.HTTP/1       1       200       0         0050       4b       0d       0a       53       65       72       3a       20       6e       67       69       67       78       K.Serve r: nginx         0060       0d       a44       61       74       65       3a       20       4f       6e       2c       20       32       34       20         0070       41       75       67       20       32       3a       36       6e       74       65       6e       74       2d       54       1       GMT       Content-T         0080       31       20       47       4d       54       0d       0a       43       6f       6e       74       2d       54       1       GMT       Content-T       De-chunked entity body (211 bytes)	Sequence number: 385 (relative [Next sequence number: 968 (re Acknowledgment number: 1060 (re	lative sequence number)	)]				
0040       62       dc       48       54       50       2f       31       2e       31       20       32       30       30       20       4f       b.HTTP/1       1       200       0         0050       4b       0d       0a       53       65       72       76       65       72       3a       20       6e       67       69       67       87       K. Serve r: nginx      Date:       Mon, 24         0070       41       75       67       20       32       30       33       30       Aug 2015       15:50:0       0         0080       31       20       47       45       66       67       65       67       2d       50       1       CMT       Content-T         Frame (649 bytes)       De-chunked entity body (211 bytes)       >	Sequence number: 385 (relative [Next sequence number: 968 (re Acknowledgment number: 1060 (re	lative sequence number)	)]				
0050 4b 0d 0a 53 65 72 76 65 72 3a 20 6e 67 69 6e 78 K.Serve r: nginx 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20 .Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 1 GMT C ontent-T Frame (649 bytes) De-chunked entity body (211 bytes)	Sequence number: 385 (relative [Next sequence number: 968 (re) Acknowledgment number: 1060 (re Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2	lative sequence number) slative ack number) 3 de d5 97 80 18	P				
0060 0d 0a 44 61 74 65 3a 20 4d 6f 6e 2c 20 32 34 20Date: Mon, 24 0070 41 75 67 20 32 30 31 35 20 31 35 3a 35 30 3a 30 Aug 2015 15:50:0 0080 31 20 47 4d 54 0d 0a 43 6f 6e 74 65 6e 74 2d 54 1 GMT C ontent.T Frame (649 bytes) De-chunked entity body (211 bytes)	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0	Lative sequence number) alative ack number) 3 de d5 97 80 18	P^#				
O080         31         20         47         4d         54         0d         0a         43         6f         6e         74         2d         54         1         GMT         C ontent-T           K         Frame (649 bytes)         De-chunked entity body (211 bytes)         >	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0 0040 62 dc 48 54 54 50 2f 31 2e 31 2	Lative sequence number) alative ack number) 3 de d5 97 80 18	P^# bz TTP/1 .1 200 0				
Frame (649 bytes)     De-chunked entity body (211 bytes)	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0 0040 62 dc 48 54 54 50 2f 31 2e 31 2 0050 4b 0d 0a 53 65 72 76 65 72 3a 2 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6	Lative sequence number) elative ack number) 3 de d5 97 80 18 5 c9 62 7a 03 db9 0 32 30 30 20 4f bH 0 6e 67 69 6e 78 K5 e 2c 20 32 34 20Da	Pbz TTP/1 .1 200 O Serve r: nginx ate: Mon, 24				
	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0 0040 62 dc 48 54 54 50 2f 31 2e 31 2 0050 4b 0d 0a 53 65 72 76 65 72 3a 2 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6 0070 41 75 67 20 32 30 31 35 20 31	Lative sequence number) alative ack number) 3 de d5 97 80 18 5 c9 62 7a 03 db 0 32 30 30 20 4f b.HT 0 6e 67 69 6e 78 KS e 2c 20 32 34 20 5 3a 35 30 3a 30 Aug	Pbz TTP/1 .1 200 0 Serve r: nginx ate: Mon, 24 2015 15:50:0				
⊖ 💆 Sourc Packets: 128 · Displayed: 6 (4.7%) · Dropped: 1 (0.8%) Profile: Default	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0 0040 62 dc 48 54 54 50 2f 31 2e 31 2 0050 4b 0d 0a 53 65 72 76 65 72 3a 2 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6 0070 41 75 67 20 32 30 31 35 20 31 3 0080 31 20 47 4d 54 0d 0a 43 6f 6e 7	Lative sequence number) alative ack number) 3 de d5 97 80 18 5 c9 62 7a 03 db 0 32 30 30 20 4f b.HT 0 6e 67 69 6e 78 KS e 2c 20 32 34 20 5 3a 35 30 3a 30 Aug 4 65 6e 74 2d 54	P^# TTP/1 .1 200 0 Serve r: nginx ate: Mon, 24 2015 15:50:0 MT_C_ontent-T				
	Sequence number: 385 (relative [Next sequence number: 968 (rel Acknowledgment number: 1060 (rel Header Length: 32 bytes 0020 bb 95 00 50 8c aa ab f0 84 5e 2 0030 00 82 39 16 00 00 01 01 08 0a 0 0040 62 dc 48 54 54 50 2f 31 2e 31 2 0050 4b 0d 0a 53 65 72 76 65 72 3a 2 0060 0d 0a 44 61 74 65 3a 20 4d 6f 6 0070 41 75 67 20 32 30 31 35 20 31 3 0080 31 20 47 4d 54 0d 0a 43 6f 6e 7	Lative sequence number) alative ack number) 3 de d5 97 80 18 5 c9 62 7a 03 db 0 32 30 30 20 4f b.HT 0 6e 67 69 6e 78 KS e 2c 20 32 34 20 5 3a 35 30 3a 30 Aug 4 65 6e 74 2d 54	P^# TTP/1 .1 200 0 Serve r: nginx ate: Mon, 24 2015 15:50:0 MT_C_ontent-T				

Figure 1.3: Wireshark with filtering for http traffic and IP source address

In case of the AMC40 board, data is sent as UDP packets, and can therefore directly be piped into Wireshark. In case of the PCIe40 board, the data is read out from the PCIe bus and delivered to userspace through a kernel driver, which is specifically created for this purpose. The kernel driver functions can be used through system calls. For sending the data to wireshark, a separate userspace program has to be written, which reads data from the kernel driver and wraps it into UDP packets before forwarding them to Wireshark.

#### **1.1** Detector board to userspace interface

The interface functions to access kernel driver functions are listed in 1.1 and can be found in pcie40\_driver/daq/daq.h in the lhcb-daq40-software git repository.

```
int p40_ctl_open(int dev);
void p40_ctl_close(int fd);
int p40_ctl_set_control(int fd, uint32_t ctl);
int p40_ctl_start(int fd);
int p40_ctl_stop(int fd);
uint32_t p40_ctl_get_status(int fd);
int p40_daq_open(int dev, void **buffer);
void p40_daq_close(int fd, void *buffer);
int p40_daq_set_read_off(int fd, uint32_t off);
uint32_t p40_daq_get_read_off(int fd);
uint32_t p40_daq_get_write_off(int fd);
uint32_t p40_daq_get_buf_size(int fd);
uint32_t p40_daq_get_msi_size(int fd);
uint32_t p40_daq_get_msi_size(int fd);
uint32_t p40_daq_get_msi_size(int fd);
```

Listing 1: Kernel driver interface

The function p40\_meta\_open returns a file descriptor and saves the pointer to the metadata in memory in one of the function arguments passed to it. The function call p40\_daq\_open works similar, but the pointer will point into the data area. The function p40\_daq\_set\_read\_off returns the offset, from which to read the current data element. p40\_ctl\_start starts data generation in the FPGA.

Only the metadata part of the kernel driver is implemented for now. The data part will follow in the next months.

#### 1.2 The new readout protocol

The new detector protocol will consist of two packet sources. The metadata source contains all the header of the data and describes which type of data of which size to find at which offset from the beginning of the data. The data consists of consecutive areas of payload; length of each payload fragment is defined in the metadata.

The header used for the data generating test program amc40\_capchecker can be found in amc40.hpp (see also listing 2).

```
struct __attribute__((__packed__)) amc40_hdr {
    uint32_t seqnum;
    uint16_t bytes;
    uint16_t frags;
    uint64_t evid;
};
```

Listing 2: MEP meta data structure

In each packet, what follows after the header are the fragments; each one of them with its own sub header, containing the sequence number (BXID) and payload length in nibbles or bytes, followed by data for each detector link (see image 1.4).

It is worth mentioning at this point, that the header structure explained is the header structure used for the readout systems currently under research. The structure will change in the future, in conjunction with the evolution of the readout system.

#### 1 Introduction

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help			
🖲 💿 🚄 🗏 📘 🛅 🗙 G I 🔍 🌚 🗞 🖄 🗏 🔲 📑 👻			
Filter: Expression Clear Apply Save			
NO. 11me Source Destination Protocol Length into			
1 0.000000 11.22.33.44 137.138.117.16 MEP2 met: 8250			
<pre>MEP2: Sequence number: 2 MEP2: Size: 8184 MEP2: Number of fragments: 488 MEP2: Event id: 953 Frag #0,FRG.bxid: 963, FRG.gdl: 68</pre>			
0 = MEP2 OPT: Data exists flag: 0x00 .000 0000 = MEP2 OPT: data length: 0 0011 1100 0011 = MEP2 OPT: bxid: 0x03c3 0 = MEP2 OPT: Data exists flag: 0x00 000 0000 = MEP2 OPT: data length: 0 ▶ Frag #1,FRG.bxid: 964, FRG.gdl: 80			
0038       00000011       10111001       00111000       000000000       01000100       001100000      <0<0<0			

Figure 1.4: MEP protocol

## 2 Implementation

### 2.1 Wireshark basics

The wireshark manpage tells us: On Unix-compatible systems, the plugins are looked for in the following directories: the lib/wireshark/plugins/\$VERSION directory under the main installation directory (for example, /usr/local/lib/wireshark/plugins/\$VERSION), and then \$HOME/.wireshark/plugins. It is possible to check whether a plugin was loaded by opening the About dialog box in Wireshark and looking at the Plugins tab.

### 2.2 Implementation of a wireshark dissector in C

Because of performance reasons, also there are Python for implementing a Wireshark dissector, the code will written in C. All the basic knowledge, how to implement the dissector, will be explained in the following subsection.

### 2.2.1 Basic dissector functions and structures

#### 2.2.1.1 Example code for example protocol

In order to understand better, how wireshark dissectors are written, we take the protocol in image 2.1 as an example.

First, we need to register the protocol we want to use. The abbreviation can later be used as a filter string in wireshark.

```
void proto_register_mep(void)
```

```
{
```

```
proto_mep = proto_register_protocol (
```

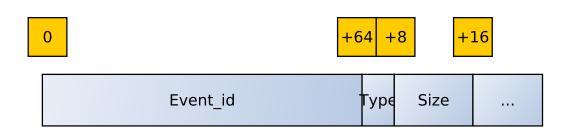


Figure 2.1: Example protocol structure

```
"MEP Protocol", /* name */
"MEP", /* short name */
"mep" /* abbrev */
);
```

}

The protocol dissector handle is registered, in our case in conjunction with the associated UDP port. In practice, probably any wireshark filter can be used here.

```
#define MEP_PORT 1234
void proto_reg_handoff_mep(void)
{
    static dissector_handle_t mep_handle;
    mep_handle = create_dissector_handle(dissect_mep, proto_mep);
    dissector_add_uint("udp.port", MEP_PORT, mep_handle);
}
```

For parsing packets, mainly the function proto\_item\* proto\_tree\_add\_item(tree, id, tvb, start, len, is used. The parameter id describes how the field should be presented in wireshark. Some example values can be seen in the following source code.

```
void proto_register_mep(void) {
    static hf_register_info hf_data[] = {
        { &hf_data_evid,
            { "MEP2 data: Event ID", "mep.data.evid",
            FT_UINT64, BASE_DEC,
            NULL, OxO,
            "evid", HFILL }
        },
        { &hf_data_type,
            { "MEP data: Type", "mep.data.type",
            FT_UINT8, BASE_DEC,
            VALS(data_type_names), 0x0,
            "data type", HFILL }
        },
        { &hf_data_size,
            { "MEP data: Size", "mep.data.size",
            FT_UINT16, BASE_DEC,
            NULL, OxO,
            "data size", HFILL }
        },
    }
}
```

In order to parse the first three fields of the protocol, we would use the following source code.

```
static void dissect_mep(tvbuff_t *tvb, packet_info *pinfo,
    proto_tree *tree) {
    gint offset = 0;
    . . .
    proto_tree *data_tree = proto_item_add_subtree(data_root,
            ett_data);
    proto_tree_add_item(data_tree, hf_data_evid, tvb, offset,
            8, ENC_BIG_ENDIAN); // 64bit = 8 byte
    offset += 8;
    proto_tree_add_item(data_tree, hf_data_type, tvb, offset,
            1, ENC_BIG_ENDIAN);
    offset += 1:
    proto_tree_add_item(data_tree, hf_data_size, tvb, offset,
            2, ENC_BIG_ENDIAN);
    offset += 2;
    . . .
    }
}
```

Without the wireshark plugin, the data can not be parsed by wireshark and the UDP payload looks like a binary blob (see 2.2). After copying the plugin into the respective wireshark plugin directory, the end result looks as seen in image 2.3.

#### 2.2.2 Important wireshark dissector functions

The argument to the function tvb\_get\_bits\* defined as tvbuff\_t \*tvb is the pointer to the buffer at that location, where the current protocol payload is stored. e.g., if UDP is the currently selected protocol, tvb will point to the first UDP payload element, just behind the UDP header (see code listing 2.2.2). From here, parsing of the data should start.

```
guint8 tvb_get_bits8(tvbuff_t *tvb, gint bit_offset, const gint no_of_bits);
guint16 tvb_get_bits16(tvbuff_t *tvb, guint bit_offset, const gint no_of_bits, const
guint32 tvb_get_bits32(tvbuff_t *tvb, guint bit_offset, const gint no_of_bits, const
guint64 tvb_get_bits64(tvbuff_t *tvb, guint bit_offset, const gint no_of_bits, const
```

Listing 3: Important dissector functions

#### Implementation

File Edit View Go Capture Analyze	Statistics Telephony Tools Interr	nals Help
	<b>x</b> G <b>Q</b> « »	* * * <b>     </b> •
Filter: !(icmp)	Expression	n Clear Apply Save
No. Time Source	Destination Pro	tocol Length Info
1 0.000000( 127.0.0.1	127.0.0.1 UDP	61 Source port: 42142 Desti
3 0.000119(127.0.0.1	127.0.0.1 UDP	71 Source port: 34129 Desti
Frame 3: 71 bytes on wire (568 bi		
<pre>Internet II, Src: 00:00:00_00:00: Internet Protocol Version 4, Src:</pre>		0:00:00_00:00:00 (00:00:00:00:00:00)
<ul> <li>User Datagram Protocol, Src Port:</li> </ul>		
<ul> <li>Data (29 bytes)</li> </ul>	04120 (04120), 000 10101 1200	(1200)
Data: 0900000000000000000000000000000000000	00242420005424242424200	
[Length: 29]		
0000 00 00 00 00 00 00 00 00 00 00 00 0010 00 39 53 ba 40 00 40 11 e8 f7		
	7 7f 00 00 01 7f 00 .9S.@.@. 3 09 00 00 00 00 00Q%	
0030 00 00 02 00 09 00 00 02 42 42	2 00 05 42 42 42 42	BBBBBB
0040 42 00 01 42 00 01 42	ВВВ	
○ M Data ( Packets: 4 · Displayed:	2 (50.0%) · Drop Profile: Defa	Jlt

Figure 2.2: Wireshark without MEP dissector plugin

#### Implementation

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help					
	🗎 🗙 G   <b>Q</b> 🖑				
Filter: !(icmp)  Expression Clear Apply Save					
No. Time Source	Destination	Protocol Length Info			
1 0.000000( 127.0.0.1	127.0.0.1	MEP2 meta 61 Num. events: 0			
3 0.000118( 127.0.0.1	127.0.0.1	MEP Data 65 Event size: 9			
<pre>&gt; Frame 3: 65 bytes on wire (520 bits), 65 bytes captured (520 bits) on interface 0 &gt; Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00), Dst: 00:00:00_00:00 (00:00:00:00) &gt; Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1) &gt; User Datagram Protocol, Src Port: 50102 (50102), Dst Port: 1235 (1235) &gt; MEP2 Protocol &gt; MEP2 Data MEP2 data: Event ID: 360287970189639680 MEP data: Type: Type 2 (2) MEP data: Size: 9</pre>					
▶ MEP2 Data Payload					
0000       <					
	: 2 (50.0%) · Drop Profile	e: Default			

Figure 2.3: Wireshark with MEP dissector plugin

### **3** Conclusion

#### 3.1 Accomplished tasks

A wireshark dissector was created, which can parse the MEP data provided by lhcb-daq40-software (see https://git.cern.ch/web/lhcb-daq40-software.git). The code for the dissector can be found on https://github.com/chrysh/lhcb-daq40-dissector. The plugin can be compiled using make. The plugin, which can then be found under .libs has then to be copied into the wireshark plugin directory before wireshark is started (see also section 2.1). For more and detailed information, refer to the wikipage https://lbdokuwiki.cern.ch/doku.php?id=upgrade: logbook\_openlab\_2015.

#### 3.2 Known issues

There are still some tasks to be done. The config file, which is provided for each detector protocol used and defines the length of each field, is not read yet. Nevertheless, a header file is used to define the length of the fields for each protocol for now. The length is set in the initialization function of the dissector plugin, and can therefore be adjusted easily.

The code is not based on the data checker to be found under https://git.cern.ch/ web/lhcb-daq40-software.git), because I needed to understand how wireshark dissectors work in the first place, and then how the lhcb-daq40-software code works. It would have been much better, if the code was actually based on the parser in lhcb-daq40-software in a modular way. In the current state, the dissector is written from scratch, with some inspiration taken from the amc40\_capchecker code, which can be found in lhcb-daq40-software.

Furthermore, the special case of having a data payload field, which is larger than four bytes can not be displayed properly, while parsing the packet does not impose a problem. Calling the function proto\_tree\_add\_bits\_item with a larger value runs into an assertion error in proto.c (line 7604). To ensure correct parsing, a temporary solution is found, which is merely displaying the first 64 bit of the field and continuing parsing with the original data length value.

#### 3.3 To be done in the future

After the functionality of the two boards, AMC40 and PCIe40, are ensured, a new network protocol has to be created, which will unite all the protocols for the different detector types. For that, another wireshark dissector has to be written. The dissector

created for this project can be a good starting point for the next dissector implementation.

Reading the configuration file from a predefined directory has to be implemented in the future. Furthermore, a solution has to be found for the special case of having data which is larger than 64 bit.