Dissector Developer Notes

Beyond the APIs

#sf24us

Jaap Keuter

Wireshark core dev

Welcome

#sf24us

First some questions to get to know each other:

Who has cloned the Wireshark repository?

Who's writing dissectors? In C or Lua?

Who's developing on which OS?

Who has read the development documentation?

What will we discuss?

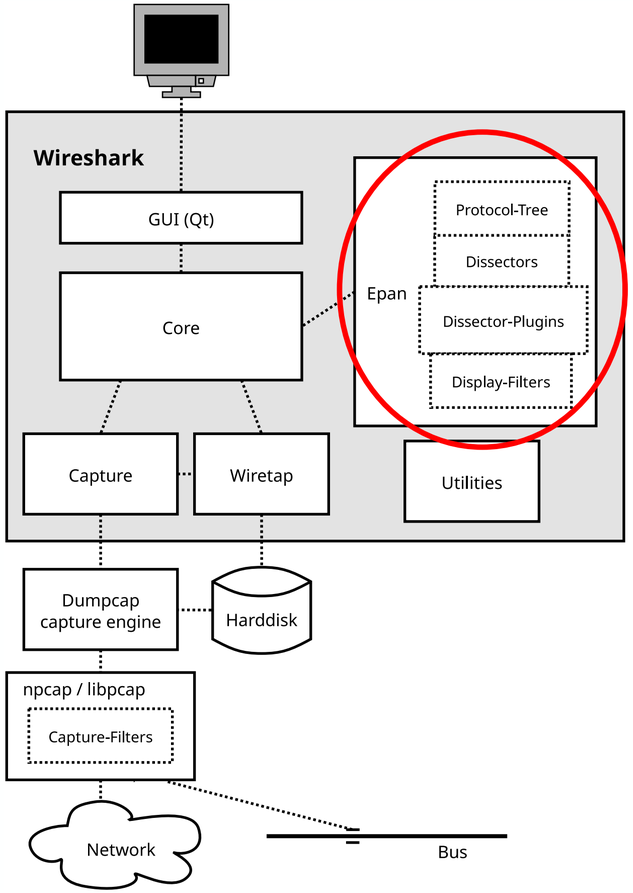
#sf24us

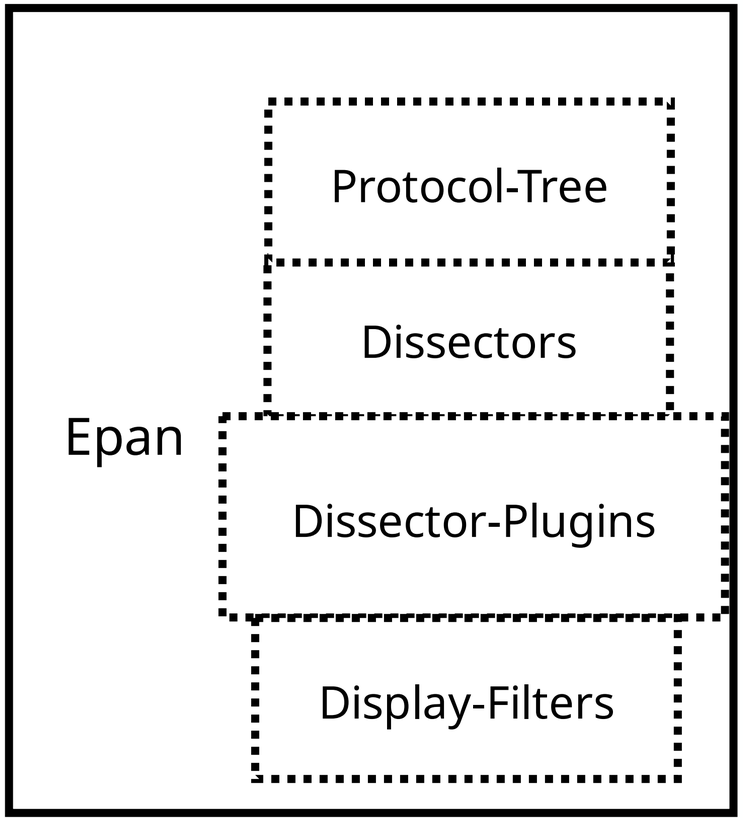
Dissectors.

Not just the EPAN API's, but what lies beyond them.

How do we think about packet dissection design?

Getting our bearings





Dissector entry point

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Who’s calling?

Who is calling `dissect_PROTOABBREV()`?
For this we need to register the dissector with EPAN.

Dissector registration

#sf24us

```
/* Register the protocol with EPAN. */
void
proto_register_PROTOABBREV(void)
{
    proto_PROTOABBREV = proto_register_protocol("PROTONAME",
        "PROTOSHORTNAME", "PROTOFILTERNAME");

    PROTOABBREV_handle = create_dissector_handle(dissect_PROTOABBREV,
        proto_PROTOABBREV);
}
```

Who’s calling? (2)

Now that EPAN knows about `dissect_PROTOABBREV()`
when does it call us?
For this we need to setup dissection handoff.

Dissection handoff

#sf24us

```
#define PROTOABBREV_UDP_PORT 10000

/* Register for handoff to the dissector. */
void
proto_reg_handoff_PROTOABBREV(void)
{
    dissector_add_uint("udp.port",
        PROTOABBREV_UDP_PORT,
        PROTOABBREV_handle);
}
```

Dissector entry point (2)

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Testy, Virtual Buffer

#sf24us

```
*
* Testy, Virtual(-izable) Buffer of guint8*'s
*
* "Testy" -- the buffer gets mad when an attempt is made to access data
*      beyond the bounds of the buffer. An exception is thrown.
*
* "Virtual" -- the buffer can have its own data, can use a subset of
*      the data of a backing tvbuff, or can be a composite of
*      other tvbuffs.
*
* Copyright (c) 2000 by Gilbert Ramirez <gram@alumni.rice.edu>
*
```

Access to packet data

#sf24us

epan/tvbuff.h

```
WS_DLL_PUBLIC guint8
tvb_get_guint8(tvbuff_t *tvb,
    const gint offset);
```

Besides this there are access functions for any imaginable data type in a TVB. Use them!

Dissector entry point (3)

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Packet info struct

#sf24us

epan/packet_info.h

```
typedef struct _packet_info {
    <insane amount of parameters>
} packet_info;
```

Frame data struct

#sf24us

epan/frame_data.h

```
typedef struct _frame_data {
    <less insane amount of parameters>
} frame_data;
```

Dissector entry point (4)

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

The protocol tree

#sf24us

epan/proto.h

```
WS_DLL_PUBLIC proto_item *
proto_tree_add_item(proto_tree *tree,
    int hfindex, tvbuff_t *tvb,
    const gint start, gint length,
    const guint encoding);
```

Header field

#sf24us

epan/proto.h

```
static hf_register_info hf[] = {
    { &hf_FIELDABBREV,
      { "FIELDNAME", "FIELDFILTERNAME",
        FT_FIELDTYPE, FIELDDISPLAY,
        FIELDCONVERT, BITMASK,
        "FIELDDESCR", HFILL }
    }
};
```

Converting values

Often when creating your dissections you want to convert a number into a representative string. But can you trust the number read from the TVB to be valid?
Setup a `value_string` array and make sure to terminate that with a `{0, NULL}` tuple. Then use the value_string conversion functions, or stick it in the header field.

Dissector entry point (5)

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
```

Out of band data

During dissection we want to pass out-of-band data between dissectors. If this is not part of packet_info, then the data parameter allows for this.
Since, in most cases, this is an unused parameter, use the "_U_" attribute to tell the compiler to ignore it.

Dissector entry point (6)

#sf24us

doc/packet-PROTOABBREV.c

```
/* Code to actually dissect the packets. */
static int
dissect_PROTOABBREV(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree,
        void *data _U_)
{
    /* create display subtree for the protocol */
    ti = proto_tree_add_item(tree, proto_PROTOABBREV, tvb, 0, -1, ENC_NA);
```

Dissector design considerations

Packet dissection order

#sf24us

In what order are packets dissected, i.e., in what order is my dissector being called?

Wireshark: First sequential, then in random order.

Tshark: Once sequential, twice sequential with “-2” option.

Ergo: you can’t use static variables!

State across packets

Packets are often not dissected in isolation. They can depend on data in earlier packets.

How to keep track of which packets belong together?

Conversations: An association defined by endpoint tuples, e.g., side A and B: IPv4 address + UDP port#

Streaming vs datagram

With datagram protocols (e.g., UDP) you know that you are getting a Protocol Data Unit worth of data.

How about streaming protocols (e.g. TCP) ?

You cannot expect the TCP dissector to give you your complete Protocol Data Units!

TCP PDU finder

#sf24us

epan/dissectors/packet_tcp.h

```
WS_DLL_PUBLIC void
tcp_dissect_pdus(tvbuff_t *tvb,
    packet_info *pinfo, proto_tree *tree,
    gboolean proto_desegment, guint fixed_len,
    guint (*get_pdu_len)(packet_info *,
      tvbuff_t *, int, void *),
    dissector_t dissect_pdu,
    void *dissector_data);
```

What to do with errors?

Remember that Wireshark gets put into action when things don't work. That may be when there's a protocol error.
To help the user, always try to show as much as possible.

What to do with errors?

Use the safety of the EPAN facilities to cover for errors, e.g., TVB, value_string, etc.
Always check the validity of values read from the TVB before using it for loop counts, shifts, etc. These EPAN can't protect you against.

What we didn’t cover

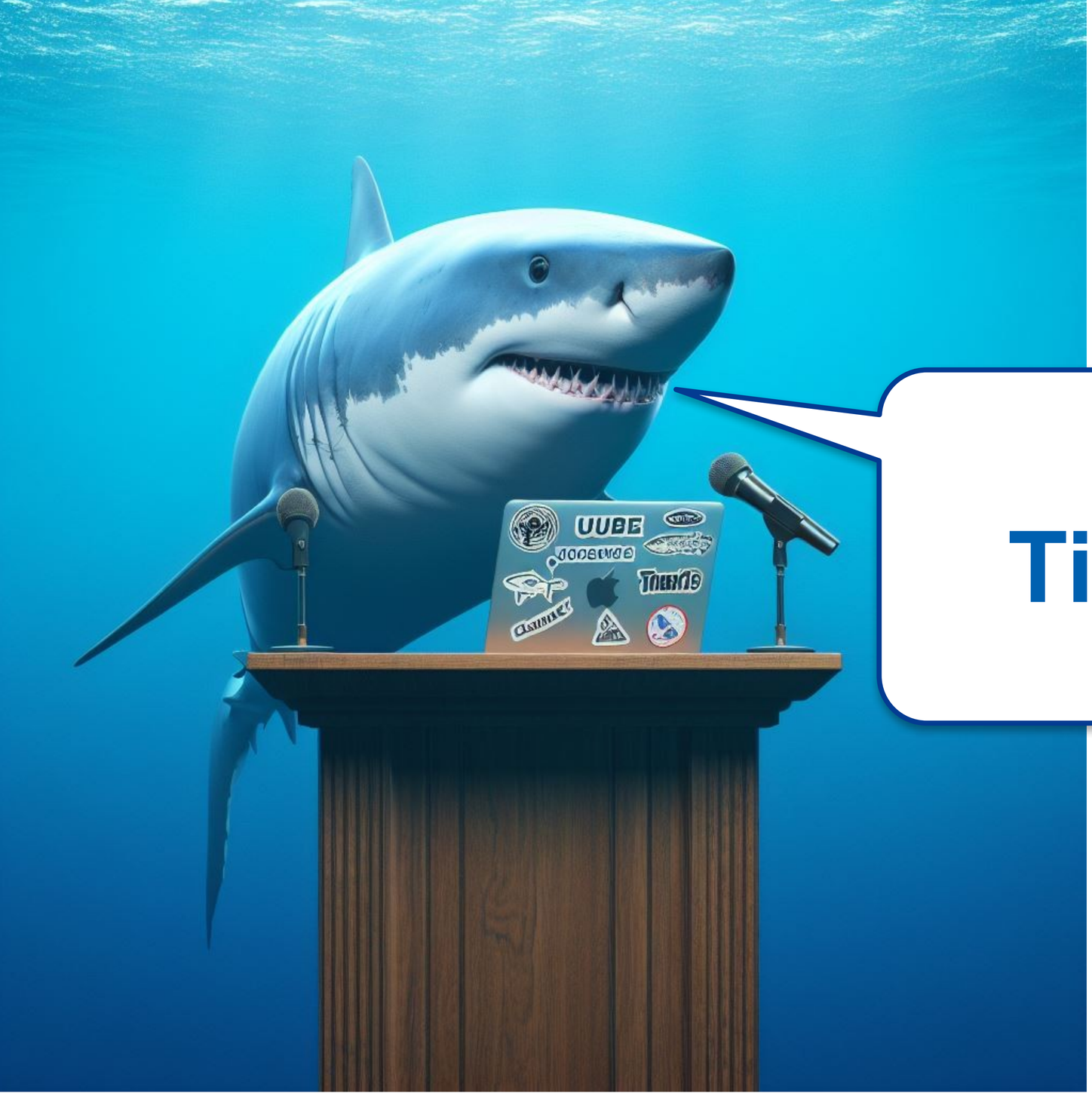
#sf24us

- Columns
- Generated and hidden fields
- Per packet data
- Heuristics
- Taps and statistics
- ...

More info to be found

#sf24us

- In the documentation
  - Developer Guide
  - doc/README.*
- In the source code



SharkFest'24 US
June 15-20 · Fairfax, VA

#sf24us

Time for Q & A