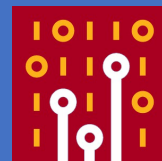




SharkFest '18 US

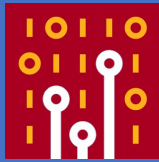


Generating Wireshark Dissectors: A Status Report

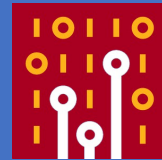
Richard Sharpe
Hammerspace



About me?



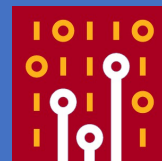
- Contributed to Wireshark since ~1999
 - Was called Ethereal then
- Do a lot of work with Wireless protocols now
- Wrote a bunch of early dissectors like SMB, FTP, etc.



- Why Generate Dissectors?
- A Data Structure Approach
- How it is done, Anltr4 & Java
- A deeper look at the description language
- Handling exceptions
- What I currently have working
- Problems
- What else could it do?



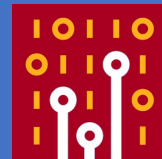
Why Generate Dissectors



- Wireshark is complex; not enough developers
- Protocols change; easier to generate new dissectors
- Have done it before and it works
 - Initial SMB dissector code was generated
 - More recently, generated code for XDR-based protocols



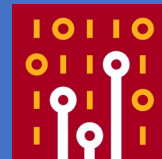
Are there any already



- ASN1
- IDL
- Private one for XDR
 - Last two take an RPC approach
- Have heard of other private ones
- Possibly gRPC



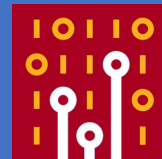
A Data Structure Approach



- Each packet is a data structure
- Various types
 - bit, uint8, byte, char, oui, ether, uint16 etc
- Specify various data structures
- Switch statements, arrays
- Specify which table to insert the dissector into



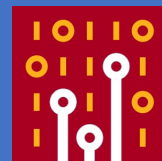
Data Structure Example



```
struct ieee1905_steering_btm_report {
    bssid "Report BSSID";
    ether_t "Reported STA MAC address";
    uint8 "BTM Status Code";
    switch (../tlv_header/tlvLength - 13) {
        case 6:
            bssid "Target BSSID";
        case 0:
            void;
        default:
            exception("Malformed Steering BTM Report, len should be
13 or 19");
    };
};
```

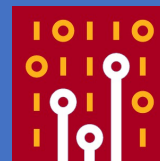


How it is done, Antlr4, Java

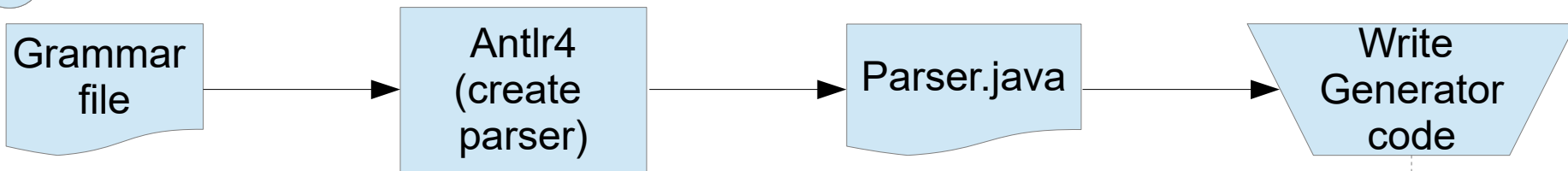


- Antlr4
 - A parser generator
 - Takes an extended BNF
 - Generates a recursive descent Java parser
- A bunch of Java
 - Walks the 'parse tree' and
 - Generates the dissector

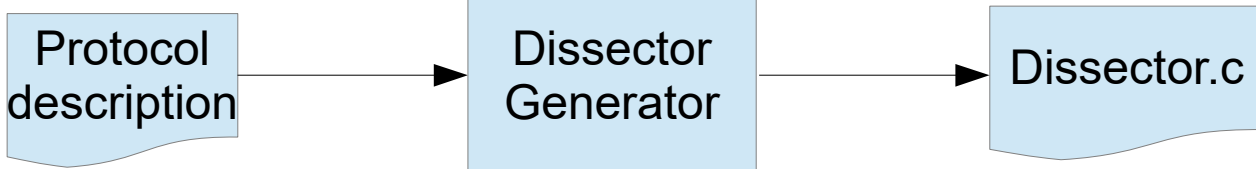
Antlr4 Workflow



1



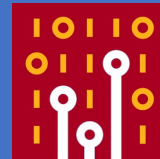
2





Working with Antlr

- Quite easy
- Create the grammar input file as BNF with regexes
- Build the parser
- Extend the parser in Java to
 - Walk the parse tree
 - Generate code



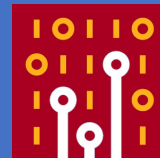
```
// ANTLR grammar for the wireshark generator
grammar WiresharkGenerator;
```

```
protocol : protoDecl+ ;
```

```
protoDecl : dissectorTableDecl ';'
           | endianDecl ';'
           | protoDetailsDecl ';'
           | dissectorEntryDecl ';'
           | enumDecl ';'
           | structDecl ';'
           | typeDef ';'
           ;
```



Antl4 Input, cont



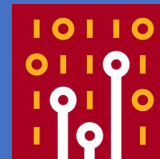
```
endianDecl : 'endian' '=' (E_BIG | E_LITTLE) ;
```

```
E_BIG      : 'big' ;
```

```
E_LITTLE   : 'little' ;
```



Antlr Input, cont



```
// We want to allow a field to carve out some contiguous bits from the
// containing type and apply a defined type to that. Eg:
// uint8:7-4:some-def some-field-name -- this is a 4-bit field or
// uint8:7:some-other-def some-field-name -- this is a 1-bit field
localEltDeclCont: ':' INT ( '-' INT)? ':' ID (ID | STRING) ;

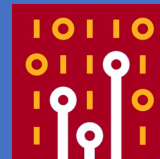
localEltDecl : ID (ID | STRING)
             | ID localEltDeclCont ( ',' localEltDeclCont)*
             ;

// Some switchStructEltCtrl items will have to be filtered out after
// parsing, because a string that matches no field would be illegal.
arrayEltDecl : ID (ID | STRING) '[' switchStructEltCtrl | INT ']' ;

structEltDecl : externEltDecl
              | localEltDecl
              | arrayEltDecl
              | switchDecl
              ;
```



Antlr4 Input, cont

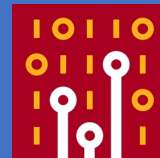


```
// A fieldpath is a series of IDs or STRINGS separated by '/'
// and perhaps
// preceded by '../' to go back up one level.
field : ID | STRING ;
fieldPath : startSym=('../' | '/')? field ( '/' field)* ;

switchStructEltCtrl : fieldPath // Can be a path to a field.
    | fieldPath op=( '!=' | '>=' | '<=' | '==' | '<<'
                    | '>>' | '+' | '-' | '&' ) (INT | ID)
    ;
```



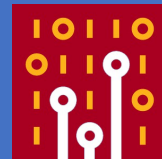
Antlr4 Input, cont



```
STRING: '''.*?''' ; //Embedded quotes?
COMMENT: '#' .*? [\n\r] -> skip ; // Discard comments for now
ID : [a-zA-Z][a-zA-Z0-9_]* ;
WS : [ \t\n\r]+ -> skip ;
INT : '0x' [0-9a-fA-F]+
      | [0-9]+ ; // Hex or decimal numbers
```



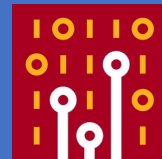
Protocol Description



- Endianness
- Top-level structures and tables to insert into
- Field types & names
- Typedefs
- Enums
- Switch statements and arrays
- Structures



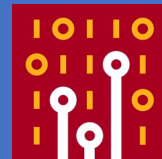
Endianness



```
# Set the endianness ...  
endian = little ; # Default = big
```



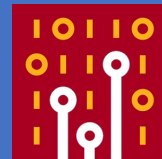
Top-level Structures



```
protoDetails = { "IEEE 1905.1a", "ieee1905", "ieee1905" };  
dissectorEntry ieee1905 = ieee1905_cmdu;  
dissectorTable["ethertype", 0x893A] = ieee1905;
```



Field Types



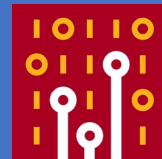
- Bit, byte, uint8, int8, uint16, int16, uint32, ... uint64
- ether_t, oui_t
- Subdivisions:

```
uint8:4-0:uint8 "Rsvd",  
      :5:extiv_vals "Ext IV",  
      :7-6:uint8 "Key ID";
```

- ▾ Key ID octet: 0x68, Ext IV
 ...0 1000 = Rsvd: 0x08
 ..1. = Ext IV: True
 01.. = Key ID: 0x01



Field Names



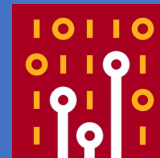
```
struct target_bssid_info {  
    bssid "Target BSSID";  
    uint8 "Target BSSID Operating Class";  
    uint8 "Target BSSID Channel Number";  
};
```

```
    uint8:4-0:uint8 "Rsvd",  
        5:extiv_vals "Ext IV",  
        7-6:uint8 "Key ID";
```

- ▾ Key ID octet: 0x68, Ext IV
 - ...0 1000 = Rsvd: 0x08
 - ..1. = Ext IV: True
 - 01.. = Key ID: 0x01



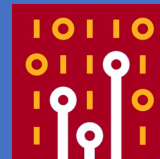
Bit fields



```
struct steering_request_flags {  
    uint8:7:steering_request_mode "Request Mode",  
        :6:boolean "BTM Disassociation Imminent",  
        :5:boolean "BTM Abridged",  
        :4-0:uint8 "Reserved";  
};
```



Typedefs & Enums

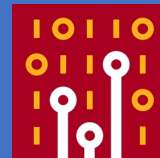


```
typedef byte bssid[6];
```

```
enum channel_preference_prefs:uint4 {  
    0x0 = NON_OPERABLE: "Non-operable",  
    0x1 = OPERABLE_PREF_1: "operable with preference score 1",  
    0x2 = OPERABLE_PREF_2: "operable with preference score 2",  
    default = "Reserved"  
};
```



Switch Statements

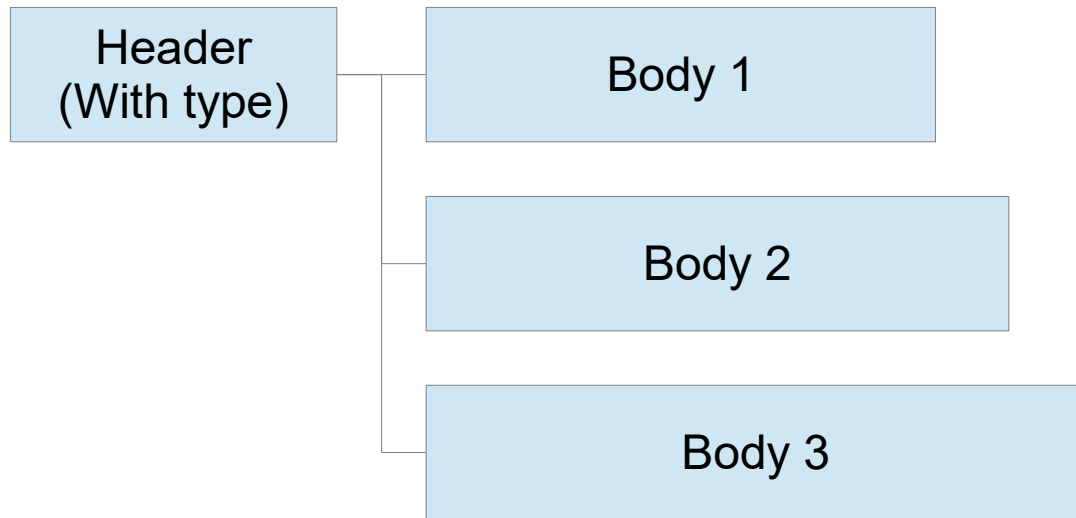
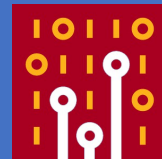


```
switch ("Steering request flags"/"Request Mode") {  
    case REQUEST_IS_STEERING_OPPORTUNITY:  
        void;  
    case REQUEST_IS_STEERING_MANDATE:  
        steering_op_window;  
};
```

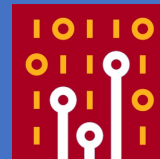
```
switch (../tlv_header/tlvLength - 13) {  
    case 6:  
        bssid "Target BSSID";  
    case 0:  
        void;  
    default:  
        exception("Malformed Steering BTM Report, len should be  
13 or 19");  
};
```



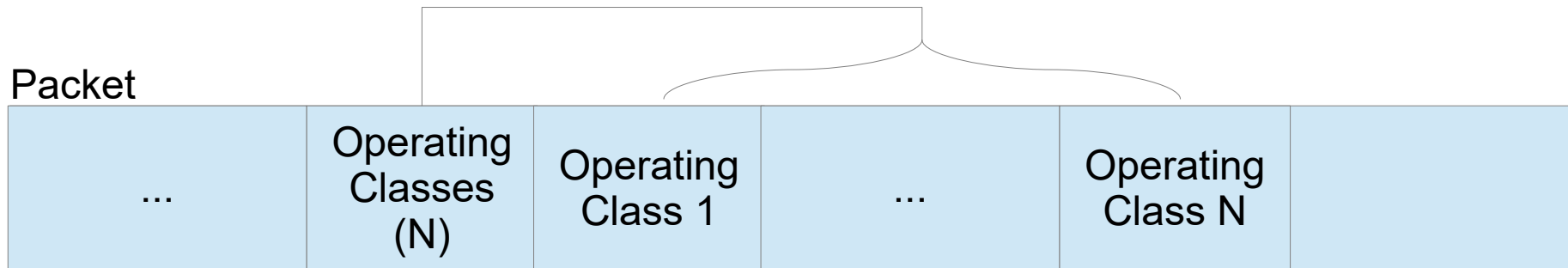
Switch Statements, cont



Arrays

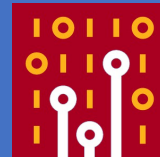


```
struct ieee1905_channel_preference {  
    radio_id "Radio unique identifier";  
    uint8 "Operating classes";  
    channel_pref_det "Operating Class List"["Operating classes"];  
};
```





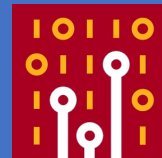
Arrays, cont



```
struct ieee1905_cmdu {
    message_version_enum messageVersion;
    uint8 reserved;
    message_type_enum messageType;
    uint16 messageId;
    uint8 fragmentId;
    uint8:7:last_fragment_enum lastFragmentIndicator,
        :6:relay_indicator_enum relayIndicator,
        :5-0:uint8 reserved;
    proto_tlv ProtocolTlvs[ProtocolTlvs/tlv_header/tlvType != 0];
    proto_tlv endOfMessageTlv;
};
```



References to fields



```
struct protocol_tlv {
    tlv_header_type tlv_header;
    switch (tlv_header/tlvType) {
        case IEEE1905_END_OF_MESSAGE:
            void;

        ...

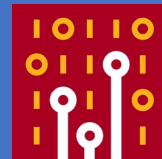
        case IEEE1905_STEERING_REQUEST_TLV:
            ieee1905_steering_request;

        case IEEE1905_STEERING_BTM_REPORT_TLV:
            ieee1905_steering_btm_report;

        default:
            exception("Unknown tlv type: %s", tlv_header/tlvType);
    };
};
```



Expert information



```
default:
```

```
    exception("Unknown tlv type: %s", tlv_header/tlvType);
```

```
switch (../tlv_header/tlvLength - 13) {
```

```
    case 6:
```

```
        bssid "Target BSSID";
```

```
    case 0:
```

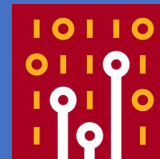
```
        debug("Invalid value for tlvLength in xxx");
```

```
    default:
```

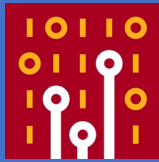
```
        exception("Malformed Steering BTM Report, len should be  
13 or 19");  
};
```



Problems



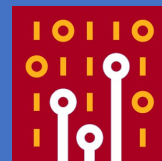
- Really only useful for new protocols today
 - Hard to work with updates to existing protocols
- Focused on Wireshark dissectors today



- About 85% done
- Written in Java
- Still have to generate:
 - The self-relative array indexes
 - hf declarations, ett declarations, boilerplate and some small things
- Another month or so

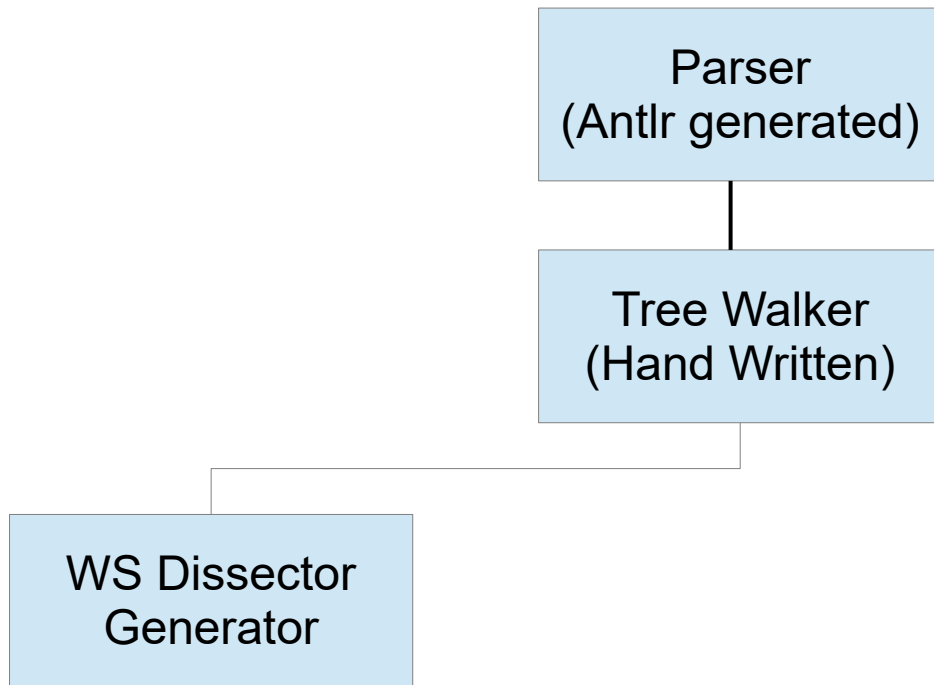
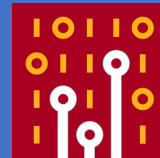


What else could it do?

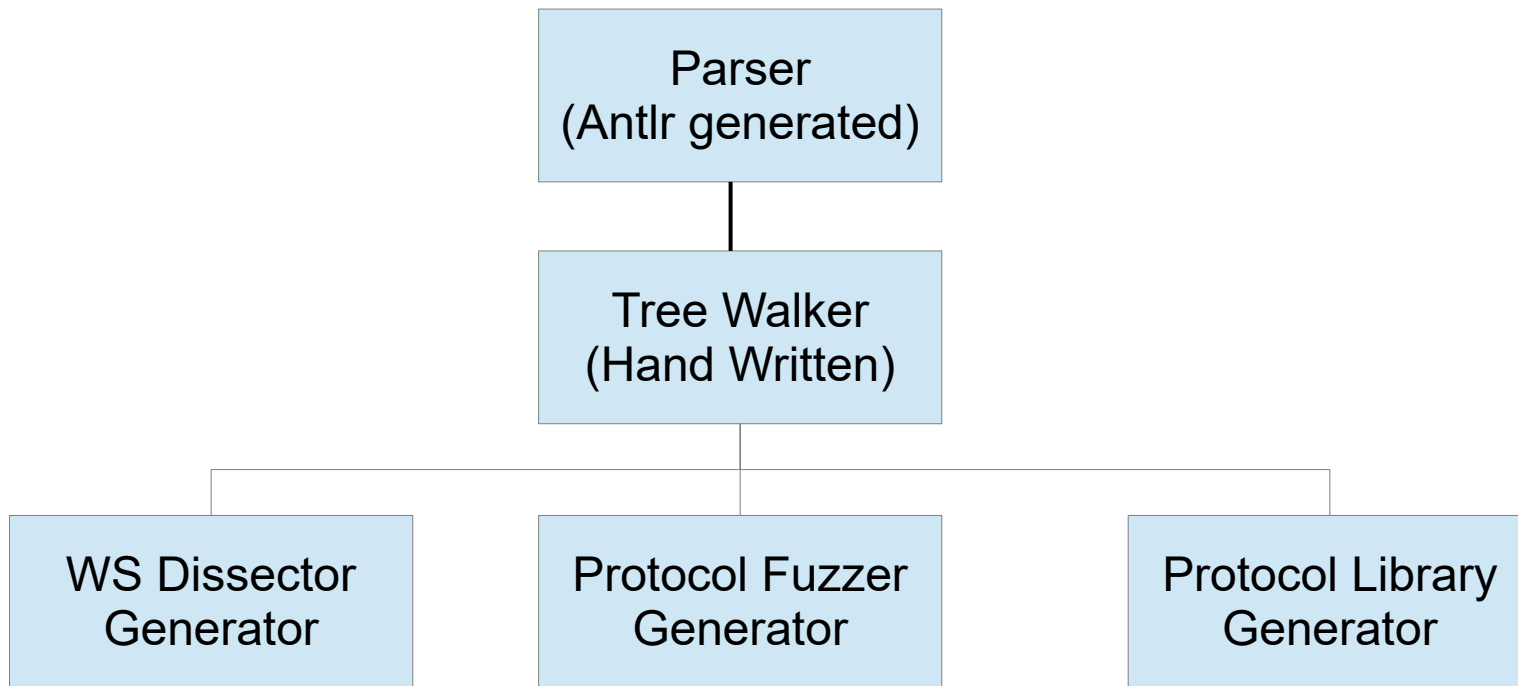
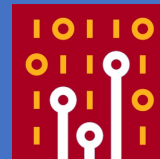


- Generate packet generators
 - For testing dissectors, implementations, fuzz testing
- For generating libraries to create/parse packets described by the protocol description
 - Different code generation back ends

What else?



What else?



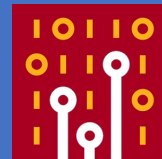


What else, cont?

- A protocol is just a list of Key:Value pairs
- A pcap file is also just a list of Key:Value pairs
 - Some values are lists or arrays of Key:Value pairs
- Could generate a program to convert to
 - JSON
 - Whatever.



The end



- Comments?
- Questions?