Introduction: How it works

Let's consider the simplest example with Hello message for OpenFlow 1.0 protocol.

First of all we describe OpenFlow 1.0 protocol in Avro protocol file:

```
{"namespace" :"of",
 "protocol":"ofp10",
 "types":[

 {"name":"uint_8", "type":"fixed", "size":1},
 {"name":"uint_16", "type":"fixed", "size":2},
 {"name":"uint_32", "type":"fixed", "size":4},

 {"name":"ofp_type",
  "type":"enum",
  "items":"uint_8",
  "list":[
   {"name":"OFPT_HELLO", "default":[0]}
  ]
 },

 {"name":"ofp_length",
  "type":"enum",
  "items":"uint_16",
  "list":[
  {"name":"OFPL_HELLO_LEN", "default":[0,8]}
  ]
 },

 {"name":"ofp_header",
   "type":"record",
   "fields":[
   {"name":"version", "type":"uint_8"},
   {"name":"type", "type":"ofp_type"},
   {"name":"length", "type":"ofp_length"},
   {"name":"xid", "type":"uint_32"}
   ]
 },

 {"name":"hello_header",
   "type":"record",
   "fields":[
   {"name":"version", "type":"uint_8", "default":[1]},
   {"name":"type", "type":"ofp_type", "default":"OFPT_HELLO"},
   {"name":"length", "type":"ofp_length", "default":"OFPL_HELLO_LEN"},
   {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
   ]
 },

 {"name":"ofp_hello",
   "type":"record",
   "fields":[
    {"name":"header", "type":"hello_header"}
```

```
    ]
  }

]
```

Now Java sample to encode the Hello:

```
import org.apache.avro.Protocol;
import org.apache.avro.generic.GenericRecordBuilder;
import org.apache.avro.generic.GenericRecord;
import org.apache.avro.generic.GenericData.Fixed;

/* Modified Avro code */
import org.apache.avro.Schema;
…….

Schema headerSchema = protocol.getType("of.hello_header");
Schema msgSchema = protocol.getType("of.ofp_hello");

GenericRecord msgRecord = new GenericData.Record(msgSchema);
GenericRecordBuilder builder = new GenericRecordBuilder(headerSchema);

// We build it to apply “default” values for hello_header fields
GenericRecord headerRecord = builder.build();
msgRecord.put("header", headerRecord);

ByteArrayOutputStream out = new ByteArrayOutputStream();

DatumWriter<GenericRecord> writer = new GenericDatumWriter<GenericRecord>(msgSchema);
Encoder encoder = EncoderFactory.get().binaryEncoder(out, null);
writer.write(msgRecord, encoder);
// Now the buffer out contains encoded message and can send it to a Switch!
```

Now Java sample to decode the Hello:

```
/* We have a byte array in containing a message. We decode a header to analyze
 * incoming message type.
 */
Schema headerSchema = protocol.getType("of.ofp_header");

GenericRecord headerRecord = new GenericData.Record(headerSchema);
GenericDatumReader<GenericRecord> reader = new
GenericDatumReader<GenericRecord>(headerSchema);
Decoder decoder = DecoderFactory.get().binaryDecoder(in, null);

reader.read(headerRecord, decoder);

// Now we can access fields. For instance, let’s extract type of the
// message
Byte type = getByte((GenericData.Fixed)headerRecord.get("type"));

// We’re able to obtain ofpt_hello_value from Avro protocol as well (to be continued)

if (type == ofpt_hello_value) {

   // Some code to handle ofp_hello message
```

```
}
```

Now Java sample to decode the Switch Config message:

What is important:

1. This can guarantee that message fields exact size according to Avro file
2. This can guarantee that message fields are encoded exact in the order according to Avro file
3. Using ‘default’ values gets rid of putting values in Java code.

What’s for OpenFlow 1.3?

```
  {"name":"hello_header",
    "type":"record",
    "fields":[
    {"name":"version", "type":"uint_8", "default":[4]},
    {"name":"type", "type":"ofp_type", "default":"OFPT_HELLO"},
    {"name":"length", "type":"ofp_length", "default":"OFPL_HELLO_LEN"},
    {"name":"xid", "type":"uint_32", "default":[0,0,0,0]}
    ]
  },

  {"name":"ofp_hello",
    "type":"record",
    "fields":[
     {"name":"header", "type":"hello_header"}
     {"name":"elements", "type":{"type":"array", "items":"ofp_hello_elems"}}
    ]
  }
```

But we still can use the same Java to operate with header. What we need is to add code to operate with Hello elements and describe Hello elements in Avro file.

In fact, we have OpenFlow 1.0 and 1.3 described in Avro. Our controller is able to select protocol is runtime, loading an appropriate Protocol object.

One boring thing that was solved easily is Padding. We just use ‘default values’ for this:

```
{"name":"ofp_action_output",
    "type":"record",
    "fields":[
```

```
     {"name":"type", "type":"ofp_action_type", "default":"OFPAT_OUTPUT"},
     {"name":"len", "type":"action_length", "default":"AL_OUTPUT"},
     {"name":"port", "type":"uint_32", "default":[0,0,0,0]},
     {"name":"max_len", "type":"ofp_controller_max_len",
"default":"OFPCML_NO_BUFFER"},
     {"name":"pad", "type":{"type":"fixed", "size":6}, "default":[0,0,0,0,0,0]}
    ] }
```

It guarantees, that pad will exactly 6 bytes and will be filled with zeros.

Avro for OpenFlow: Some techniques

Emulation of Base class (or Interface)

It's possible to use Union to emulate Base class paradigm:

```
   {"name":"ofp_instruction",
    "type":"record",
    "fields":[
      {"name":"instruction", "type":["ofp_instruction_goto_table",
                                     "ofp_instruction_write_metadata",
                                     "ofp_instruction_write_actions",
                                     "ofp_instruction_clear_actions",
                                     "ofp_instruction_apply_actions"]}
    ]
   },


   {"name":"instruction_set",
    "type":"record",
    "fields":[
      {"name":"set", "type":{"type":"array", "items":["ofp_instruction", "null"]}}
    ]
   },

   {"name":"ofp_flow_mod",
    "type":"record",
    "fields":[
      {"name":"header", "type":"flow_mod_header"},
      {"name":"base", "type":["flow_mod_body_add","flow_mod_body_delete"]},
      {"name":"match", "type":"ofp_match"},
      {"name":"instructions", "type":"instruction_set"}
    ]
   },
```

In this example `ofp_instruction` is a kind of base class for all OF instruction structures. We can refer to it from Java code. And it's possible to create an Avro array containing variable ofp_instructions (`instruction_set`)