Chapter 7.1:  
Layer 6:  Presentation Formatting

CS/ECPE 5516:  Comm. Network  
Prof. Abrams  
Spring 2000
Layer 5’s Data Formatting Problem

Good morning → Bonjour

Little-endian: 00000001 → 01000000

Big-endian: 00000001 → 01000000
Layer 5’s Data Formatting Problem

- Format messages going from host A to B:
  - Computers represent primitive types differently
    - Integers: big vs. little endian
    - Floating point: IEEE 754, Vax, Cray, IBM
  - Compilers lay out C structures differently
    - May/may not add padding to align on word boundaries
    - Complex structures can be "flattened differently":
      - Arrays: row vs. column major
      - Trees: pre, in, post-order traversal
Design Choices

- To convert or not to convert
  - Convert to network standard byte ordering, or
  - Don’t convert (receiver-makes-right)

- To tag or not to tag
  - Just send the value
  - Send type (e.g., "int") and value
  - Send architecture (e.g., "big-endian"), type, value

- To align or not to align (to word boundaries)
Example 1: XDR (eXternal Data Representation)

- To convert or not to convert:
  - Convert!
- To tag or not to tag:
  - Don’t tag!
    - Compilers on both ends must be compatible (rpcgen) or libraries on both ends must be compatible (xdr)
- To align or not to align to words:
  - Characters are 1 byte, everything else 4 bytes
## Sample XDR Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>32-bits</td>
<td>32-bit signed int</td>
</tr>
<tr>
<td>Unsigned int</td>
<td>32-bits</td>
<td>32-bit unsigned int</td>
</tr>
<tr>
<td>String</td>
<td>Arbitrary</td>
<td>String of ASCII chars</td>
</tr>
<tr>
<td>Structure</td>
<td>Arbitrary</td>
<td>Data aggregate, like C’s <em>struct</em></td>
</tr>
</tbody>
</table>
XDR Library Calls

- XDR calls will **encode** or **decode**
- Start by creating **XDR stream**:
  - Memory buffer
  - Designate buffer for encoding or decoding
    - So same library calls are used by sender & receiver
- Example:

```c
#define BUFSIZE 100
XDR *xdrs;
char buf[BUFSIZE];
xdrmem_create(xdrs, buf, BUFSIZE, XDR_ENCODE);
```
## Sample XDR Procedures

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Arguments</th>
<th>Data Type Converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>xdr_int</td>
<td>xdrs, ptr_to_int</td>
<td>32-bit signed int</td>
</tr>
<tr>
<td>xdr_u_int</td>
<td>xdrs, ptr_to_uint</td>
<td>32-bit unsigned int</td>
</tr>
<tr>
<td>xdr_string</td>
<td>xdrs, prt_string, maxsize</td>
<td>String of ASCII chars</td>
</tr>
</tbody>
</table>
Example of XDR on Sender

- Following copies an int into network standard byte order in memory area for xdrs:

```c
int j;
...
j=260;
xdr_int(xdrs, &j);
```

<table>
<thead>
<tr>
<th>hdr</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
</table>
Example of XDR on Receiver

- Following copies an int from network standard byte order in memory area for xdrs to receiver:

```c
int j;
...
xdr_int(xdrs, &j);
```
XDR Variations

**For TCP:**
- Call `xdrstdio_create` to connect XDR stream to TCP socket.
- Doing `xdr_int(…)` automatically reads or writes socket!

**For UDP:**
- Call `xdrrec_create` to use XDR
- Then calls like `xdr_int(…)` check the memory buffer: When full, call `outproc` to send datagram
Alternative to XDR: ASN.1

- Abstract Syntax Notation One is ISO standard
- To convert or not to convert:
  - Convert!
- To tag or not to tag:
  - Tag with type and length
  
  int | 4 | byte0 | byte1 | byte2 | byte3

- To align or not to align to words:
  - No! So data rarely falls on word boundary!
Yet Another Alternative to XDR

- NDR: Network Data Representation
- Used in Distributed Computing Environment
- To convert or not to convert:
  - NO conversion!
- To tag or not to tag:
  - Tag: 4-byte architecture, type and length (1/stream)

<table>
<thead>
<tr>
<th>Int-Rep</th>
<th>Char-Rep</th>
<th>FloatRep</th>
<th>Extension1</th>
<th>Extension2</th>
</tr>
</thead>
</table>

13 CS/ECPE5516