API for Internet Protocols (1): IPC characteristics

- synchronous and asynchronous communication
  - blocking send: waits until the corresponding receive is issued
  - non-blocking send: sends and moves on
  - blocking receive: waits until the msg is received
  - non-blocking receive: if the msg is not here, moves on
- Message Destination
  - IP address + port: one receiver, many senders
  - Location transparency
    - DNS (e.g. Mach): provides location-independent identifier mapping to lower lever addresses
    - send directly to processes (e.g. V System)
    - multicast to a group of processes (e.g. Chorus)
- Reliability
- Ordering

API for Internet Protocols (2): Sockets and ports

- programming abstraction for UDP/TCP
- originated from BSD UNIX

API for Internet Protocols (3): UDP Datagram

- message size: up to 2^{16}, usually restrict to 8K
- blocking: non-blocking send, blocking receive
- timeouts: timeout on blocking receive
- receive from any: doesn’t specify sender origin (possible to specify a particular host for send and receive)
- failure model:
  - omission failures: can be dropped
  - ordering: can be out of order
- use of UDP
  - DNS
  - less overhead: no state information, extra messages, latency due to start up

API for Internet Protocols (4): C and UDP datagrams

Sending a message

```
s = socket(AF_INET, SOCK_DGRAM, 0);
bind(s, ClientAddress);
Send(s, "message", ServerAddress);
```

Receiving a message

```
s = socket(AF_INET, SOCK_DGRAM, 0);
bind(s, ServerAddress);
amount = recvfrom(s, buffer, from);
```
API for Internet Protocols (5): Java and UDP

```java
DatagramPacket request = new DatagramPacket(buf, length, aHost, serverPort);
DatagramPacket reply = new DatagramPacket(buf, length);
...  
DatagramSocket reply = new DatagramSocket();
DatagramSocket request = new DatagramSocket();
...  
InetAddress aHost = InetAddress.getByName(…);
...  
DatagramPacket buffer = new DatagramPacket();
...  
DatagramPacket request = new DatagramPacket(buf, length, aHost, serverPort);
DatagramPacket reply = new DatagramPacket(buf, length);
...  
ServerAddress and ClientAddress are socket addresses
```

API for Internet Protocols (6): TCP stream

- message size: unlimited
- lost messages: sequence #, ack, retransmit after timeout of no ack
- flow control: sender can be slowed down or blocked by the receiver
- message duplication and ordering: sequence #
- message destination: establish a connection, one sender-one receiver, high overhead for short communication
- matching of data items: two processes need to agree on format and order (protocol)
- blocking: non-blocking send, blocking receive (send might be blocked due to flow control)
- concurrency: one receiver, multiple senders, one thread for each connection
- failure model
  - checksum to detect and reject corrupt packets
  - sequence # to deal with lost and out-of-order packets
  - connection broken if ack not received when timeout
  - multiple flows, could be split, could be data-parallel
- type tag followed by the selected member
- union
- signed long (the values are specified by the order declared)
External Data Representation (3):

- CORBA IDL compiler generates marshalling and unmarshalling routines
- Struct with string, string, unsigned long

<table>
<thead>
<tr>
<th>Index in sequence of bytes</th>
<th>Notes on representation</th>
<th>String length</th>
<th>String value</th>
<th>Unsigned long</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-3</td>
<td>4 bytes</td>
<td>5</td>
<td>&quot;Smith&quot;</td>
<td></td>
</tr>
<tr>
<td>4-7</td>
<td></td>
<td></td>
<td>&quot;London&quot;</td>
<td></td>
</tr>
<tr>
<td>8-11</td>
<td></td>
<td></td>
<td>1934</td>
<td></td>
</tr>
<tr>
<td>12-15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16-19</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20-23</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24-27</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The flattened form represents a Person struct with value: {'Smith', 'London', 1934}

External Data Representation (5): Java serialization

- serialization and de-serialization are automatic in arguments and return values of Remote Method Interface (RMI)
- flattened to be transmitted or stored on the disk
- write class information, types and names of instance variables
- new classes, recursively write class information, types, names...
- each class has a handle, for subsequent references
- values are in Universal Transfer Format (UTF)

External Data Representation (6): Java serialization

```java
public class Person implements Serializable {
    private String name;
    private String place;
    private int year;
    public Person(String aName, String aPlace, int aYear) {
        name = aName;
        place = aPlace;
        year = aYear;
    }
}
```

The true serialized form contains additional type markers:
- h0 and h1 are handles/references to other objects within the serialized form

External Data Representation (7)

- references to other objects
  - other objects are serialized
  - handles are references to objects in serialized form
  - each object is written only once
  - second or subsequent occurrence of the object is written as a handle
- reflection
  - ask the properties (name, types, methods) of a class
  - help serialization and de-serialization

External Data Representation (8): XML

- User-defined tags (vs. HTML has a fixed set of tags)
- different applications agree on a different set of tags
- E.g. SOAP for web services, tags are published
- Tags are in plain text (not binary format)—not space efficient

External Data Representation (9)

```xml
<person id="123456789">
  <name>Smith</name>
  <place>London</place>
  <year>1934</year>
</person>
```

The true serialized form contains additional type markers:
- h0 and h1 are handles/references to other objects within the serialized form
External Data Representation (10): XML namespace

- Name clashes within an application
- Namespaces: a set of names for a collection of element types and attributes
- xmlns: xml namespace
- pers: name of the namespace (used as a prefix)
- http://www.cdk4.net/person: location of schema

```xml
<person pers:id="123456789" xmlns:pers="http://www.cdk4.net/person">
  <pers:name> Smith </pers:name>
  <pers:place> London </pers:place>
  <pers:year> 1934 </pers:year>
</person>
```

External Data Representation (11): XML schema

- Defines elements and attributes
- Similar to type definition
- xsd: namespace for xml schema definition

```xml
<xsd:schema xmlns:xsd="URL of XML schema definitions">
  <xsd:element name="person" type="personType"/>
  <xsd:complexType name="personType">
    <xsd:sequence>
      <xsd:element name="name" type="xs:string"/>
      <xsd:element name="place" type="xs:string"/>
      <xsd:element name="year" type="xs:positiveInteger"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xs:positiveInteger"/>
  </xsd:complexType>
</xsd:schema>
```

External Data Representation (12): Remote object reference

- Call methods on a remote object (CORBA, Java)
  - Unique reference in the distributed system
  - Reference = IP address + port + process creation time + local object # in a process + interface
  - Port + process creation time -> unique process
  - Address can be derived from the reference
  - Objects usually don’t move; is there a problem if the remote object moves?
  - Name of interface: what interface is available

<table>
<thead>
<tr>
<th>Internet address</th>
<th>port number</th>
<th>time</th>
<th>object number</th>
<th>Interface of remote object</th>
</tr>
</thead>
</table>

Client-server communication (1)

- Synchronous: client waits for a reply
- Asynchronous: client doesn’t wait for a reply

Client-server communication (2): Request-reply message structure

<table>
<thead>
<tr>
<th>messageType</th>
<th>int (0=Request, 1=Reply)</th>
</tr>
</thead>
<tbody>
<tr>
<td>requestId</td>
<td>int</td>
</tr>
<tr>
<td>objectReference</td>
<td>RemoteObjectRef</td>
</tr>
<tr>
<td>methodId</td>
<td>int or Method</td>
</tr>
<tr>
<td>arguments</td>
<td>array of bytes</td>
</tr>
</tbody>
</table>

Why requestID?

Client-server communication (3)

- Failure model
  - UDP: could be out of order, lost...
  - process can fail...
  - not getting a reply
  - timeout and retry
  - duplicate request messages on the server
  - How does the server find out?
  - Idempotent operation: can be performed repeatedly with the same effect as performing once.
  - Idempotent examples?
  - Non-idempotent examples?
  - History of replies.
  - Retransmission without re-execution
  - How far back if we assume the client only makes one request at a time?
Client-server communication (4): RPC exchange protocols

<table>
<thead>
<tr>
<th>Name</th>
<th>Messages sent by</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>Client</td>
</tr>
<tr>
<td>Reply</td>
<td>Server</td>
</tr>
<tr>
<td>Acknowledge reply</td>
<td>Client</td>
</tr>
</tbody>
</table>

Client-server communication (5)

- using TCP increase reliability and also cost
- HTTP uses TCP
  - one connection per request-reply
  - HTTP 1.1 uses "persistent connection"
    - multiple request-reply
    - closed by the server or client at any time
    - closed by the server after timeout on idle time
- Marshal messages into ASCII text strings
- resources are tagged with MIME (Multipurpose Internet Mail Extensions) types: test/plain, image/gif...
- content-encoding specifies compression alg

Client-server communication (6): HTTP methods

- GET: return the file, results of a cgi program, ...
- HEAD: same as GET, but no data returned, modification time, size are returned
- POST: transmit data from client to the program at url
- PUT: store (replace) data at url
- DELETE: delete resource at url
- OPTIONS: server provides a list of valid methods
- TRACE: server sends back the request

Client-server communication (6): HTTP request/reply format

```
GET //www.dcs.qmw.ac.uk/index.html HTTP/1.1
```

HTTP version status code reason headers message body

HTTP/1.1 200 OK resource data

Group communication (1)

- multicast
- useful for:
  - fault tolerance based on replicated services
  - discovering services
  - multicast to find out who has the services
  - better performance through replicated data
  - multicast updates
  - event notification
  - new items arrived, advertising services

Group communication (2): IP multicast

- class D addresses, first four bits are 1110 in IPv4
- UDP
- Join a group via socket binding to the multicast address
- messages arriving on a host deliver them to all local sockets in the group
- multicast routers: route messages to out-going links that have members
- multicast address allocation
  - permanent
  - temporary:
    - no central registry by IP (one addr might have different groups)
    - use (time to live) TTL to limit the # of hops, hence distance
    - tools like sd (session directory) can help manage multicast addresses and find new ones
Group communication (3): Reliability and ordering

- UDP-level reliability: missing, out-of-order...

- Effects on
  - fault tolerance based on replicated services
    - ordering of the requests might be important, servers can be inconsistent with one another
  - discovering services
    - not too problematic
  - better performance through replicated data
    - loss and out-of-order updates could yield inconsistent data, sometimes this may be tolerable
  - event notification
    - not too problematic