Cloud computing and distributed systems

Chapter #4
Interprocess communication

Zeynep Yücel

Ca’ Foscari University of Venice
zeynep.yucel@unive.it
yucelzeynep.github.io

The API for the Internet protocols

The characteristics of interprocess communication

- Message Passing: Send/receive operations.
- Synchronous vs. Asynchronous Communication:
  - Synchronous: Processes block messaging.
  - Asynchronous: Non-blocking sending.

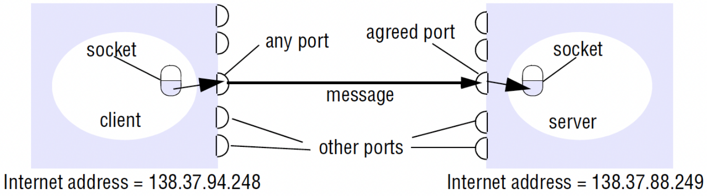
The API for the Internet protocols

The characteristics of interprocess communication

- Message Destinations: Pairs of internet address - port number.
  - Each port: One receiver, possibly multiple senders.
  - Processes: may receive at multiple ports.
  - Any process: can send messages to a process, if it knows the port number
  - Service by IP: Dependency on location.
  - Service by name: Flexibility.
- Reliability: Message delivery with no corruption or duplication.
- Ordering: Delivery sequence requirement.

The API for the Internet protocols

Sockets


socket
client
any port
agreed port
socket
server
message
other ports
Internet address = 138.37.94.248
Internet address = 138.37.88.249

- Interprocess communication: Messages via sockets.
- UDP and TCP use sockets as end points of communication.
  - Socket binding: Local port and address.
  - Single socket: Both send/receive.
  - Many port numbers in one computer.
  - No port sharing between processes.
  - Multiple processes can send to the same port.

The API for the Internet protocols

UDP Datagram communication

- UDP: No acknowledgements, retries.
- To send/receive, first create a socket bound to an internet address and port no.
    - Server: Known port. Client: Any port.
- UDP issues:
    - Message size
    - Blocking
    - Timeout
    - Receive from any

The API for the Internet protocols

UDP Datagram communication

- Message size: Limit on the array size.
  - Truncated or fragmented.
- Blocking: Non-blocking sends.
  - "Send" operation returns immediately.
  - Messages queued at receiver.
  - Collect message with "receive" operation.
  - Messages discarded at a port, if no socket is bound to it.
  - Receive operation blocks until a timeout.

The API for the Internet protocols

UDP Datagram communication

- Timeout:
  - Timeouts avoid indefinite waiting.
- Receive from any: "Receive" operation accepts messages universally.
  - It returns sender details (internet address, port no).
  - Send/receive restricted to specific address and port.

The API for the Internet protocols

UDP Datagram communication

- UDP datagrams suffer from omission failures.
  - Application implement their own reliability checks.
- UDP is suitable for applications with occasional message loss tolerance.
  - E.g. DNS, VoIP.
  - UDP has low overhead.
  - Overhead arises from storage of state information, extra messages, latency.

The API for the Internet protocols

Key points of TCP stream communication

- Byte Stream Abstraction: Stream of bytes.
- Message Sizes: Varying data amounts.
- Lost Messages: Acknowledgments ensure receipt.
- Flow Control: Regulates transfer speed.
- Message Duplication and Ordering: Detects duplicates, reorders.
- Message destinations: First connect and accept, then read from/write to a stream.

The API for the Internet protocols

Key points of TCP stream communication

- Pair of sockets, pair of streams one in each direction.
- Closing Sockets: Indicates stream end.
- Matching of data items: Agree on formats to avoid interpretation errors.
- Blocking: Processes may block, if no data is available or due to flow control limits.
- Thread Usage: Separate threads for each client for efficiency.
- Failure Handling: Checksums (against corruption), sequence numbers (against duplicates), timeouts (for retransmission) ensure reliability.
- Common TCP Services: HTTP, FTP, Telnet, SMTP.

External data representation and marshalling

External data representation

- Running programs with data structures, messages with byte sequences.
- Data structures need flattening for transmission.
- Different data representations on different computers.
- Binary data exchange methods:
  - Agreed format.
  - Sender's format.
- Agreed standard: External data representation.

External data representation and marshalling

Marshalling

- Marshalling/unmarshalling: Data assembly/disassembly
- Three alternative approaches to external data representation: CORBA, Java, XML
- In CORBA and Java, middleware handles marshalling
- XML larger than binary
- Some methods include type information.
- Lightweight methods: Protocol Buffers, JSON

External data representation and marshalling

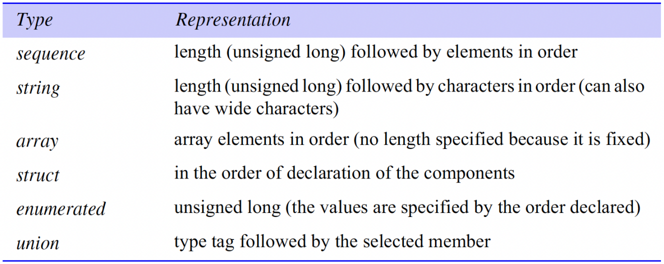
CORBA's Common Data Representation (CDR)

- CDR standard for RMIs in CORBA
- 15 primitive types and other composite types

External data representation and marshalling

CORBA's Common Data Representation (CDR)

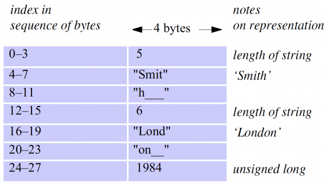
- Data as byte sequence
  - Primitive types: Big-endian/little-endian order
    - Byte storage methods
  - Constructed types: primitive types added in specific order



Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

External data representation and marshalling

CORBA CDR message



index in sequence of bytes	← 4 bytes →	notes on representation
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20–23	"on__"	
24–27	1984	*unsigned long*

```
struct Person{
string name;
string place;
unsigned long year;}
```

- Person struct for CDR messages with 3 fields
  - Length and data calculations
- Automatic marshalling via IDL

External data representation and marshalling

Java Object Serialization

- Serialization: flattening objects
- Java RMI: objects and primitives can be passed as arguments or results.
- Implementing serializable interface

External data representation and marshalling

Java Object Serialization

- Objects must implement Serializable
- Serialization flattens the object for storage/transmission, deserialization restores it
- No class information is assumed for the deserializing interface
- So some class info must be included in serialized form
  - Class name/version

External data representation and marshalling

Java Object Serialization

- Objects reference other objects
  - Referenced objects serialized together.
- References are serialized as handles (i.e. reference to an object within the serialized form)
  - One-to-one relation between objects and their handles.
- Object are written only once (second on handles are used).
- To serialize an object, its class info is written
  - if instance variables contain other objects, also their class info is written
  - Recursive for all instance variables.
- No class duplication in serialization

External data representation and marshalling

Java Object Serialization

- Primitive types (chars etc) are serialized in a portable binary format
- Strings serialized in UTF-8 format
- Automatic handling of de/serialization by middleware
  - No programmer involvement.
  - Custom methods possible.
- Transient variables: object that should not be serialized.

External data representation and marshalling

Java Object Serialization - Reflection

- Reflection: Inquiry of class properties (e.g. names and types of instance variables)
- Allows for generic de/serialization
- Dynamic class creation
- Specific argument constructors
  - No special functions needed.
  - Identifying class details.
  - Recreate class from name.
- Instantiate new object with the constructor

External data representation and marshalling

Extensible Markup Language (XML)

- Markup languages represent text and its structure
  - HTML define web pages, XML structured documents for web.
- Data's logical structure, attribute value pairs
- XML definition of Person:

```
<person id="123456789">
<name>Smith</name>
<place>London</place>
<year>1984</year>
<!-- a comment -->
</person >
```

External data representation and marshalling

Extensible Markup Language (XML)

- Web services communication
- Archiving, interfaces and configuration files
- Extensible
- Self-describing
- Human-readable

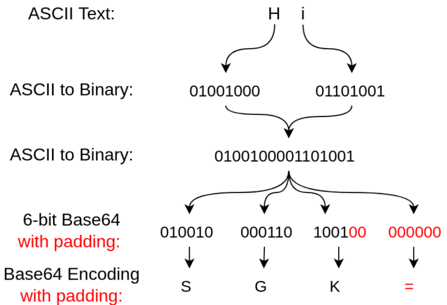
External data representation and marshalling

XML elements and attributes

- Tags and data structure
- Actual data (e.g. "Smith")
- Elements: data with tags
  - Hierarchical representation.
- Tags can have attributes (e.g. id = 123456789)
- Data container vs labeling
- All information in XML is represented as character data
  - Base64 notation is necessary to represent for encrypted elements

How Base64 works

- Fundamentally, Base64 is used to encode binary data as printable text.
- Consider the sentence `Hi`.
- Obtain binary representation of each character (see ASCII-to-binary conversion table).
- ASCII uses 8 bits to represent individual characters, but Base64 uses 6 bits (in 24-bit sequences). So the binary needs to be broken up into 6-bit chunks.
- These 6-bit values can be converted into the appropriate printable character by using a Base64 table.
- Since Base64 uses 24-bit sequences, padding is needed when the original binary cannot be divided into a 24-bit sequence.



External data representation and marshalling

Parsing and well-formed documents in XML

- Must be well-formed
- Matching tags
- Single root
- Fatal error, if not well-formed
- Special characters
  - Use CDATA

```
<place> King&apos Cross </place >
<place> <![CDATA [King’s Cross]]></place >
```

- Prolog required in first line
  - Version, encoding

    ```
    <?XML version = "1.0" encoding = "UTF-8" standalone = "yes"?>
    ```

External data representation and marshalling

XML Namespaces

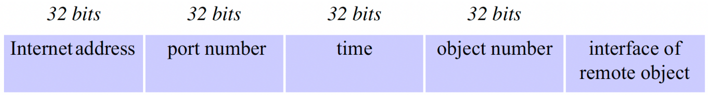
- Possible name clashes
- Namespaces for distinguishing between names
- URL scoping

```
<person pers:id="123456789" xmlns:pers = "http://www.cdk5.net/person"
<pers:name> Smith </pers:name>
<pers:place> London </pers:place >
<pers:year> 1984 </pers:year>
</person>
```

  - Pers prefix is associated with the namespace defined by the URL
  - Shorthand reference
- Using multiple namespaces is possible

Remote object references

- For languages supporting distributed object model
- Unique identifiers for remote objects
- Sent inside invocation messages
- Unique over time and space
- One way to ensure uniqueness:



32 bits	*32 bits*	*32 bits*	*32 bits*	
Internet address	port number	time	object number	interface of remote object

Remote object references

- Align with process lifespan
- Invocation messages sent to the internet address and port no
- Location-independent references
  - Distributed routing

Multicast communication

- Pairwise message exchange is inefficient
- Multicast: one message to a group of processes
- No guarantees on delivery and ordering
- Key benefits
  - Fault tolerance
  - Service discovery
  - Performance improvement
  - Event notifications

Multicast communication

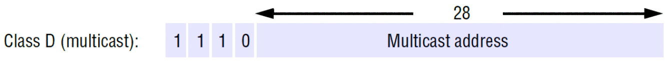
IP multicast – An implementation of multicast communication

- Single packet to a multiple computers.
- Identities or number of recipients are unknown.
- Multicast groups identified by Class D addresses
- Dynamic membership
- Join the group to receive
- No need to join the group to send

Multicast communication

Multicast Routers, Multicast Address Allocation

- Local/Internet multicasts
- Network capabilities at local scale, multicast routers at Internet scale
- TTL setting to limit journey of a packet
- Address management
  - Local Network Block
  - Internet Control Block
  - Ad Hoc Block
  - Administratively Scoped Block



Multicast communication

Multicast Address Allocation

- Permanent or temporary addresses
  - Permanent addresses for reserved protocols
  - Temporary groups need to be created before being used (with a free address).
- IP multicast protocol does not handle this issue.

Network virtualization: Overlay networks

- Diversity of applications: Network virtualization
- Reasons for interest in network virtualization
  - Different applications varying needs
  - Changing protocols impractical
- Multiple virtual networks tailored for specific applications
- Optimization without changing the underlying network

Network virtualization: Overlay networks

Overlay networks

- Virtual network layer
- Tailored services, better efficiency, additional services
- Advantages
  - New service definition
  - Experimentation support
  - Multiple overlays can coexist
- Disadvantages
  - Performance penalties
  - Indirection layer

Network virtualization: Overlay networks

Overlay networks

- Overlays: layers existing outside standard architecture.
- Freely redefine elements.
- Unique approaches for specific applications.
    - E.g. distributed hash tables.
    - E.g. Skype.

Network virtualization: Overlay networks

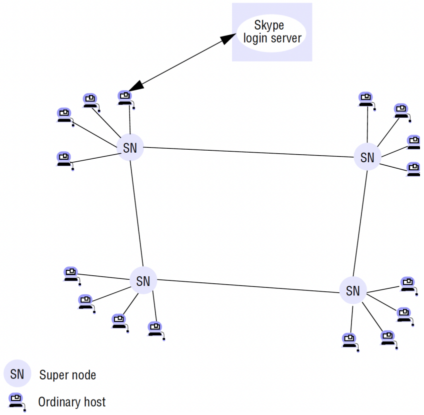
Skype: An example of an overlay network

- Skype: peer-to-peer VoIP.
- In Mar 2020, 100M users/month, 40M users/day
- Virtual network connecting active users.
- Architecture studied through traffic analysis.

Network virtualization: Overlay networks

Skype architecture and user connection

- Peer-to-peer with super nodes.
  - Super nodes: better capabilities.
- Login through login server, then connect to a super node.
- Clients cache super node identities.



Network virtualization: Overlay networks

Search for users and voice connection

- Super nodes search global index of users.
- Client's chosen super node contacts other super nodes.
- If found, establish voice connection.
- UDP preferred for streaming audio.

Discussion topic

Is it conceivably useful for a port to have several receivers?

Discussion topic

Why can't binary data be represented directly in XML, for example, by representing it as Unicode byte values? XML elements can carry strings represented as base64. Discuss the disadvantages of using this method to represent binary data.

Discussion topic

What are the main arguments for adopting a super node approach in Skype?