Communication Abstractions

- Architectures
  - model application / system building blocks, e.g., Service-Oriented Architecture (SOA) services
- Middleware
  - abstract communication details
  - model high-level communication exchanges, e.g., group communication, publish-subscribe
- Remote Invocations
  - Interface Definition Languages (IDLS) formalize interfaces
  - model remote interfaces for data and objects
- High-level protocols
  - model data exchange between entities
- Low-level protocols
  - specifications formalize communication details

 Protocols

- A system of rules that allow communicating entities to transmit and exchange information
- Typically "standardized" in specifications that detail all necessary information to implement the protocol
  - data formats
  - address formats
  - address mapping (e.g., IP to Ethernet MAC)
  - routing (for indirect communication)
  - detection of transmission errors (e.g., checksum errors)
  - acknowledgements (connection-oriented protocols)
  - information loss (timeouts, retries)
  - media access control (half-duplex links)
  - sequence control (stream-oriented protocols)
  - flow control (e.g., packet transmission rate limitations)

 Protocol Taxonomies

- Classified by domain
  - connection-oriented (e.g., TCP)
  - connectionless (e.g., UDP)
- Classified by function
  - tunneling (e.g., SSH)
  - interaction (e.g., SMTP, RMI, DCOM)
  - application (e.g., HTTP, IMAP)
- Classified by abstraction level
  - application (e.g., HTTP, POP, IMAP)
  - transport (e.g., TCP, UDP)
  - network (e.g., IP)
  - network interface (e.g., Ethernet)
- "protocols are to communications as programming languages are to computations"

 Protocol Layering

TCP/IP - model

- Application
- Transport
- Internet
- Network interface
Marshalling
- The process of transforming the memory representation of an object to a storage or transmission data format
- Raises the abstraction level of communication
  - abstracts serialization of objects to primitive types
- Higher level protocols often (when a data semantic can be inferred) define marshaling methods as part of the protocol, or expose customization points (plug in interfaces) for marshaling
- Transport protocols typically agnostic of data semantics
- Many standards exist
  - CORBA’s Common Data Representation (CDR)
  - Java Object Serialization
  - Extensible Markup Language (XML)
  - Javascript Object Notation (JSON)
  - Google protocol buffers

Interface Definition Languages (IDLs)
- Specification languages used to describe software component interfaces
  - define data type sets and component method interfaces
- Often used in distributed systems to define network-accessible component interfaces, e.g., services
  - WSDL (SOAP web services)
- Typically used as templates for source code generation
  - stub / skeleton code for RPC, RMI, web services

IDL Example
```java
// interfaceDataService
// interface definitions
import java.util;

interface DataService {
    // method definitions
    void sendData();
    void storeData(Data[] data);
}
```

Protocol Example
```
Byte layout of the Creo protocol request message for the sendData() method of the IDL example. Data encoded in the order defined in service descriptions, arrays and strings prefixed with item counts. Byte block sizes and primitive types in black, protocol preamble (protocol and method ids) and aggregated (struct and array) types in red.
```

Marshalling
- Impossible to describe all states of a distributed algorithm due to failures in processes and message transmissions

Distributed Programs
- Single process programs
  - sequential instruction steps
  - behavior and state determined by sequence of operations
- Distributed process programs
  - parallel instruction steps
  - state is private and hidden
  - behavior and state determined by sequences of operations and transmission of messages (compute, send, receive)
  - execution rates and message transmissions asynchronous
  - flow depends on local state and messages received

Message Passing
- Protocols are essential but too low level to
  - expose interfaces over networks
  - access remote contexts and semantics
- Programming constructs to raise the abstraction level
  - systems programming rather than communication code
  - requires vertical stack (“under the hood”) knowledge
- Synchrony of the system
  - asynchronous vs synchronous
- Type of communication network
  - point-to-point vs broadcast tunnel
- Failure models
  - process failures
  - communication failures
Point-to-Point Network Models

- Processes can fail, be slow or produce incorrect output
- Channels suffer message delays or losses
  - packet retransmissions
- Messages typically buffered at sender and receiver
- Assumptions
  - sent messages are eventually received
  - processes execute indefinitely

Network Failure Models

- Node level packet drops (buffer overflows)
  - sender and receiver
- Network congestion (switch / router overflows)
  - delays and drops
- Latency (TTL packet and connection timeouts)
- Hardware failures
  - partial (e.g., overheating) or complete failures
- Software failures
  - bugs, configuration errors, human factor
- Recovery is hard
  - trace message delivery stacks
  - rollback distributed transactions
  - rollback deployments

Network Types

- Broadcast networks (e.g., Ethernet)
  - all messages sent to everyone
  - recipients filter out their own messages
- Circuit-switched networks (e.g., PSTN, ISDN)
  - emulates a physically switched system
  - capacity allocated statically
  - (route and bandwidth reserved)
  - typically optical mesh or telecom network
- Packet-Switched networks (e.g., IP networks)
  - breaks data streams into packets and send in routed hops
  - datagram packets routed best effort (may drop packets)
  - improves medium utilization
  - (especially with resource limitations)

Network Types

- Wireless networks (e.g., cell networks, WLAN)
  - shared medium
  - subject to medium contention
  - (noise and bandwidth limitations)
  - new topologies exploring medium characteristics
- Lossless networks (e.g., Infiniband)
  - rate limitations and buffer control to avoid packet drops
  - congestion = link failure
  - lossless at the price of high latencies when saturated

Synchronous Systems

- Known upper bound for process execution time steps
- Processes have clocks with known / approximatable
  bounds on clock drift (versus real time)
- Known bounds for message delays
  - time to send, transport, receive message over link
- Attractive from a programming perspective
  - simple programming models

Asynchronous Systems

- No timing assumptions
  - message delays, clock drifts, step execution times
  - all unknown
- Attractive from a modeling perspective
  - simplicity of semantics
  - simplicity of porting applications
  - variable or unexpected workloads result in asynchrony
  - processes share resources
  - communication channels share networks
Network Types

- Software-Defined Networks (SDN)
  - software overlays mapped on physical networks
    - network virtualization
  - separates networks in control and data planes
  - capacity (bandwidth) reservations
  - dynamic control of link qualities
- Virtual networks (e.g., VPN)
  - virtual secure channels
  - overlays onto (any type of) transport

TCP vs UDP

- Transmission Control Protocol (TCP)
  - dropped packets resent
  - packet delivery sequencing (ordering) guarantees
  - flow control (sliding buffer window)
  - congestion control
  - connection-oriented reliable data stream
- User Datagram Protocol (UDP)
  - best effort (single try) packet delivery (no retransmissions)
  - no sequencing, no flow control, no connection
  - developer responsible for avoiding congestion

Reliability vs Sequencing

- Reliability and sequencing (TCP)
  - all packets need to arrive in order
    - e.g., simpler programming model
- Reliability
  - all packets need to arrive, order doesn’t matter
    - e.g., striped (parallel) file transfers
- Sequencing
  - order matters for delivered packets, some may be missing
    - e.g., real-time systems (VoIP, streaming video)
- Neither reliability nor sequencing (UDP)
  - what arrives arrives, order doesn’t matter
    - e.g., unreliable channels, non-deadline constrained systems

Understanding Distributed Systems

- What are the communicating entities (representation)?
- What communication paradigms are used?
- What roles and responsibilities do entities have?
- How are entities mapped onto physical resources (placement)?

Communicating Entities

- Systems perspective
  - processes (threads)
  - interprocess communication mechanisms
- Programming perspective
  - need for problem-oriented abstractions
  - encapsulate behavior in modules
  - expose functionality through interfaces
Roles and Responsibilities

- Communicating entities take on roles to establish interaction semantics
- Client-Server
  - entities may take on both roles, but not in the same interaction
  - communication typically client-driven
  - the most widely used communication pattern (simplicity)
- Peer-to-Peer
  - entities (peers) collaborate with comparable roles
  - nodes organized in (structured or unstructured) overlays
  - all nodes offer the same interfaces
  - nodes typically provide (as well as consume) resources
  - designed for resilience
  - widely popular in file sharing (decentralization and read-only data facilitates scalability)

Communication Paradigms

- Interprocess communication
  - (relatively) low-level communication tools
  - basic message passing primitives and APIs (e.g., sockets)
  - fine-grained knowledge of other entities required
- Remote invocation
  - encapsulates invocation of a remote method
  - data marshalling, message serialization, invocation semantics
  - entities separated in space, but not in time
  - request-reply patterns often used
  - coarse-grained knowledge of other entities required
- Indirect communication
  - entities separated in space and/or time
  - higher communication abstraction levels
  - less (if any) knowledge of other entities required

Indirect Communication Techniques

- Group communication
  - messages delivered to sets of recipients
  - aka distributed event-based communication
- Publish-subscribe systems
  - subscription-based one-to-many message routing
- Message queues
  - point-to-point message delivery
  - provides a level of indirection between parties
- Tuple spaces
  - entities communicate indirectly through data items (tuples) stored on shared persistent storage (tuple space)
- Distributed Shared Memory (DSM)
  - distributed RAM abstracted transparently
  - infrastructure handles synchronization and consistency

Placement

- Requires careful consideration of a number of factors
  - communication patterns
  - load patterns
  - resource capability and reliability
  - application knowledge
- Placement strategies
  - multi-tenancy
  - server consolidation
  - caching
  - replication
  - (user and/or resource) mobility
  - affinity (and anti-affinity) constraints
- Currently the topic of much research
  - virtualization facilitates placement flexibility
Tiered Architectures

- Two-tier architecture
  - client / interface
  - server / data

- Three-tier architecture
  - presentation / interface (e.g., web server)
  - logic (e.g., application server)
  - data (e.g., database)

- Four tier / N-tier architectures
  - application / client (e.g., devices)
  - connection / delivery (e.g., cached content delivery)
  - data management / aggregation (e.g., service aggregation, federation, discovery, etc).
  - data / service (e.g., functionality-oriented services)

Middleware

- The "software glue" between operating systems and applications / systems
- Provide functionality-oriented APIs and tools
- Abstract complexity and hide underlying technologies
- Facilitate software (application) development

Distributed Systems Middleware

- Distributed software systems that
  - hide communication details (protocols, marshalling, etc)
  - hide heterogeneity of underlying systems (networks, hardware, operating systems, programming languages)
  - raise abstraction levels (high-level APIs)
  - can be seen as a form of virtualization
- Middleware extend programming models
  - local procedure calls → remote procedure calls
  - object-orientation → remove method invocation
  - event-based programming models
- Middleware abstract communication artefacts
  - APIs and type sets
  - (data) marshalling
  - protocols

Abstraction Perspective

- Protocols abstract heterogeneity in
  - networks
- Middleware abstract heterogeneity in
  - hardware
  - operating systems
  - platforms
  - programming languages
  - networks

Middleware Perspective

- Functionality oriented around specific use cases
  - e.g., peer-to-peer overlays for file sharing
- Uniform programming models (servers and applications)
  - RPC / RMI
  - remote event notification
  - distributed transaction processing
- Integration points for different communication technologies
- Bridges between different communication styles
- Different types and complexity levels of middleware exist
  - Message-Oriented Middleware (MOM)
  - Object Request Brokers (ORBs)
  - Enterprise Service Bus (ESB)