Unit 1 Introduction

Unit Outcomes. Here you will learn

- to define a distributed system (DS) and describe and justify the characteristics of all DSs
- to classify DSs by various important aspects, appreciate the challenges in accurately specifying the behaviour of a DS
- about some common ways to distribute responsibility among networked nodes
- about network abstractions avialable to DS developers
- about basic tools for deploying and managing DSs

Further Reading: CDK2005 1, 2.3, 3, 4

Contents

Definition of Distributed System

A Distributed System (DS) is

- a software system
  - deployed on networked multiple computers (nodes)
  - making these nodes cooperate towards a shared goal
  - nodes communicate only by exchanging messages

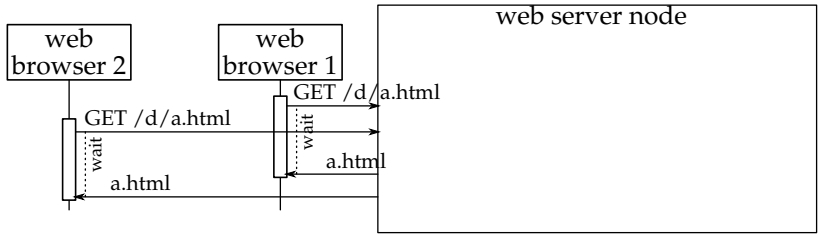
Some common DSs

- phone networks (phones and exchanges)
- email (email clients, SMTP servers and mail-box servers)
- World-Wide Web (web servers and browsers)
- Internet search engines (high-performance clusters)
- enterprise systems (database servers and clients)
-
-
-
-
-

Characteristics of a DS
Concurrency

- messages arrive concurrently, are processed concurrently

  example:



- multi-threading inside a node usually necessary
- access to internal data must be synchronised

No global clock

- impossible to 100% synchronise clocks among nodes
- approximate synchronisation using Cristian protocol:
  - estimating message delays:
  - using radio or satellite signals:
- more generally: a node cannot fully know others' state

Independent failures

- failures occur in components of DS
- how to communicate failures?
  - when node dies completely
    - only by lack of response
  - when network or its part fails silently
    - slow or broken?
    - nodes may be stranded, DS fragmented

Classification of DSs
Purpose

- information transfer over distance
- resource sharing
- performance enhancement
  - processing of large data
  - complex scientific computation
- protection against node failure by duplicating resources

Scalability

- how does a DS cope with increasing load?
- load = eg count of nodes, requests, users, ...
- perfectly scalable DS:
  - can be adapted to take any load
  - without modifying protocols or sw
  - cost of hw grows proportionally: $O(n)$
  - decrease in performance logarithmic: $O(\log(n))$ time increase for basic ops

Security — attacks

- *security* = level of resistance to attacks
- *attack* = attempt to make a DS behave in unintended ways
- typical goals of attacks:
  - unauthorised access to information
  - unauthorised alteration of data
  - total system break-down

Security — attack techniques

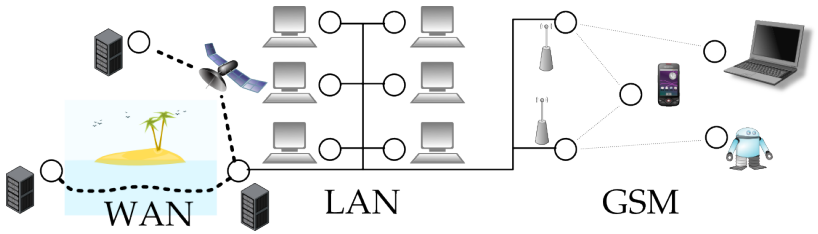
- typical techniques used in attacks:
  - deploy a *rogue node* to the DS
  - *spy* on messages in the DS
  - *modify* messages in the DS
  - *flood* the DS with legitimate but meaningless messages
  - send *malicious mobile code*

Security — levels and solutions

- various levels of security in DS by combinations of:
  - relying on private network
  - authenticating nodes using central authority
  - secure channels — no understanding without a key
  - combating isolated internal attacks (eg by voting techniques)
  - analysing any incoming mobile code
- beware: for some attacks no known cure

Heterogeneity support

- ? is a DS deployable through a variety of:
  - network types
  - hardware platforms + operating systems



Openness

- how easy is it to extended and adapt a DS?
- a DS is *open* to the extreme if
  - all its protocols and APIs are public and well-documented
  - it provides well-documented access methods to all its resources
  - it is well-designed for extensibility, eg:
    - good decomposition (low coupling, cohesion, layers of abstraction)
    - no unnecessary complexity
  - it supports heterogeneity of networks, OSs and hardware
  - it is extensible in any common programming language

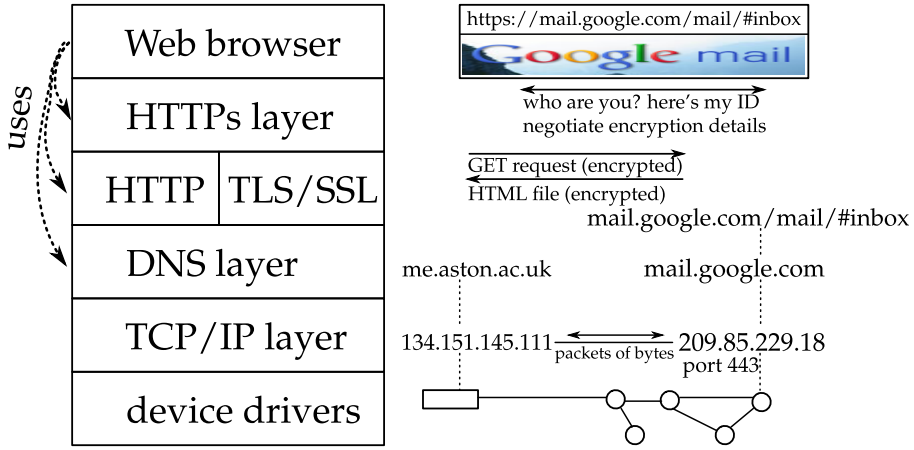
Transparency (1/2)

- transparency = certain aspect is not detectable (a tangible measure of abstraction)
- what to hide?
  - *access tr.*: local or remote?
  - *location tr.*: location in the net
  - *mobility tr.*: is it in a fixed location or moving?
  - *failure tr.*: failures of node hw/sw or communication hw/sw

Transparency (2/2)

- what to hide?
  - *performance tr.*: overall load of DS
  - *scaling tr.*: size and power of DS
  - *replication tr.*: using multiple copies of a single resource
  - *concurrency tr.*: doing something concurrently

Network abstraction
Stacking abstraction layers



- network = nodes + links + communication facilities

Role of protocols

- *protocol* is a specification of
  - message formats
  - valid sequences of messages
- that allow multiple nodes cooperate to achieve a goal
- protocol usually defined in terms of lower-level protocol(s)
  layers of protocols
- abstract networks defined by a set of protocols
- eg Internet =
  - TCP/IP: reliable messaging + numerical addressing of nodes
  - DNS: hierarchical and human-friendly naming of nodes

Internet protocols for DS developers

- TCP provides logical channels between pairs of remote nodes
  - as if they were directly linked
  - can stream sequences of bytes of any length either way
  - hiding IP packets from users
- establishing a TCP channel:
  - one computer listens, the other initiates
  - multiple listeners → listening on *ports* 1–
- DNS lesser-known facts:
  - several DNS names can share one IP
  - one DNS name can refer to multiple computers
  - DNS-IP mapping sensitive to location

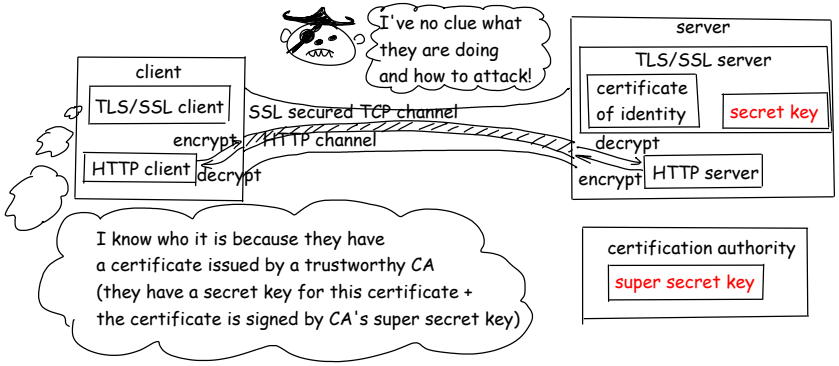
Web layer — resources and HTTP

- resources = passive or active entities accessible over Internet
- can be viewed as logical nodes/sub-nodes for a DS
- URL — resource locator `http://a.com/d.pdf?page=1#fig1`

  scheme, authority (DNS name + TCP port), path, query, fragment
- URI = URL or URN, URN has no DNS or TCP information, only a name, eg `urn:isbn:0-486-27557-4`
- HTTP: a request-response protocol for manipulating resources
- most common requests and their purpose:
  - GET: obtain a remote resource
  - POST: initiate operation on a remote resource
  - PUT: store a local resource at a remote location
  - DELETE: permanently remove a remote resource

Web layer — HTTPS

- HTTP over a secure channel:

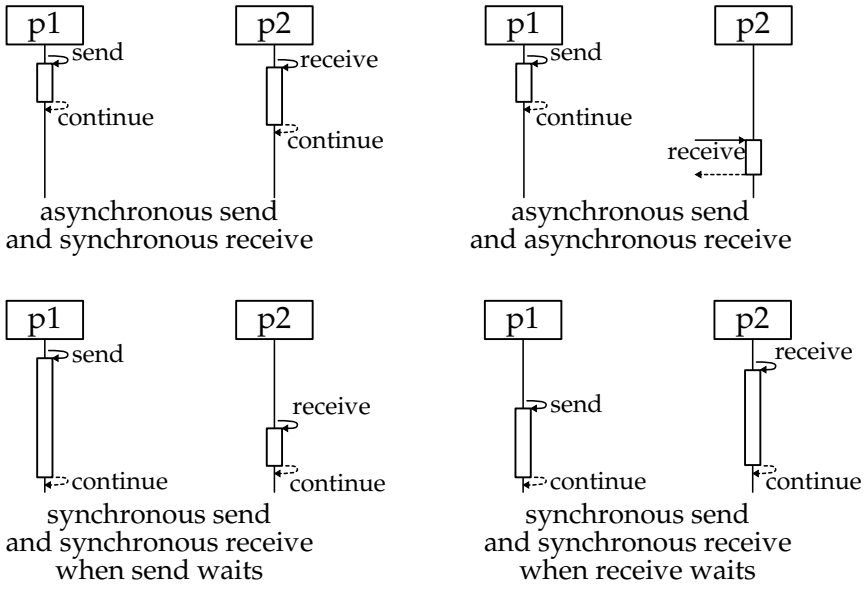


- protocol based on HTTP, TLS/SSL, TCP/IP, DNS

Network abstraction
Middleware — network abstraction

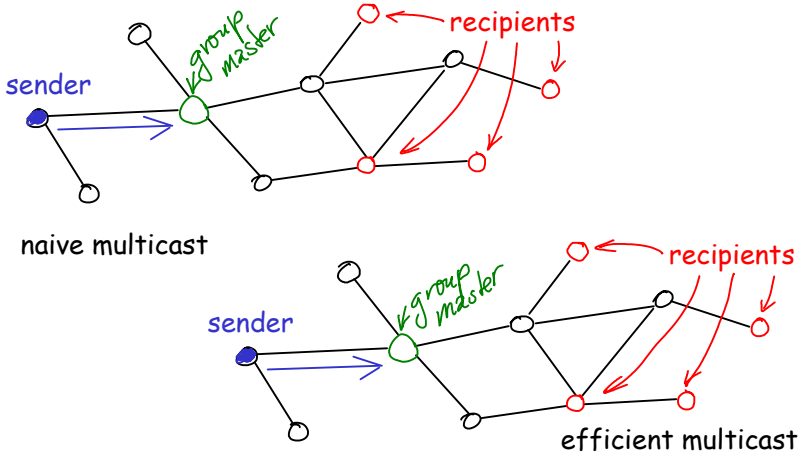
- characterised by
  - addressing logical nodes, not physical nodes
    - eg processes, resources, objects
    - results in location & access transparency
  - convenient messaging, transporting typed structured data
    - types of data declared
      → greater openness and PL heterogeneity
    - synchronisation of communicating nodes

Synchronisation of communicating nodes



Multicast

- most efficient when implemented at TCP/IP level
- sometimes available at middleware level



Remote administration
Why remote administration?

- install/update/configure/deploy a node's software
- read logs local to a node to diagnose a problem
- tweak some data files to fix a problem or simulate a problem
- monitor a node's load to tune a DS algorithm

Remote shell

- command line is simple but powerful — easy remote operation
- SSH: security achieved via secured TCP channels
- demo: install prg, view load, search files, rename files
- advantages/disadvantages?

Remote execution of GUI applications

- X Windows
  - thin-client: very efficient remote graphical clients
  - can have secure connection via SSH
  - not for MS Windows applications, only X-Windows applications
- Remote desktop
  - desktop completely taken over by remote computer
  - powerful but slow unless on very fast network
  - works on all OS, common in MS Windows administration

Learning Outcomes

Learning Outcomes. You should now be able to

- define a DS
- name, describe and justify DS characteristics
- name and define important DS classifying aspects
- evaluate a given well-known DS against a set of classifying aspects
- list several levels of network abstraction behind a typical DS; for each level identify the key concepts that are relevant to a DS developer
- define and correctly use terms: TCP port, TCP channel, URL, secure channel, multicast
- explain difference between synchronous and asynchronous send
- give the main idea of an efficient multicast implementation
- start remote programs, copy files using SSH