Principles of Software Construction: Objects, Design, and Concurrency

Introduction to networking and distributed systems

Christian Kästner Bogdan Vasilescu

School of Computer Science

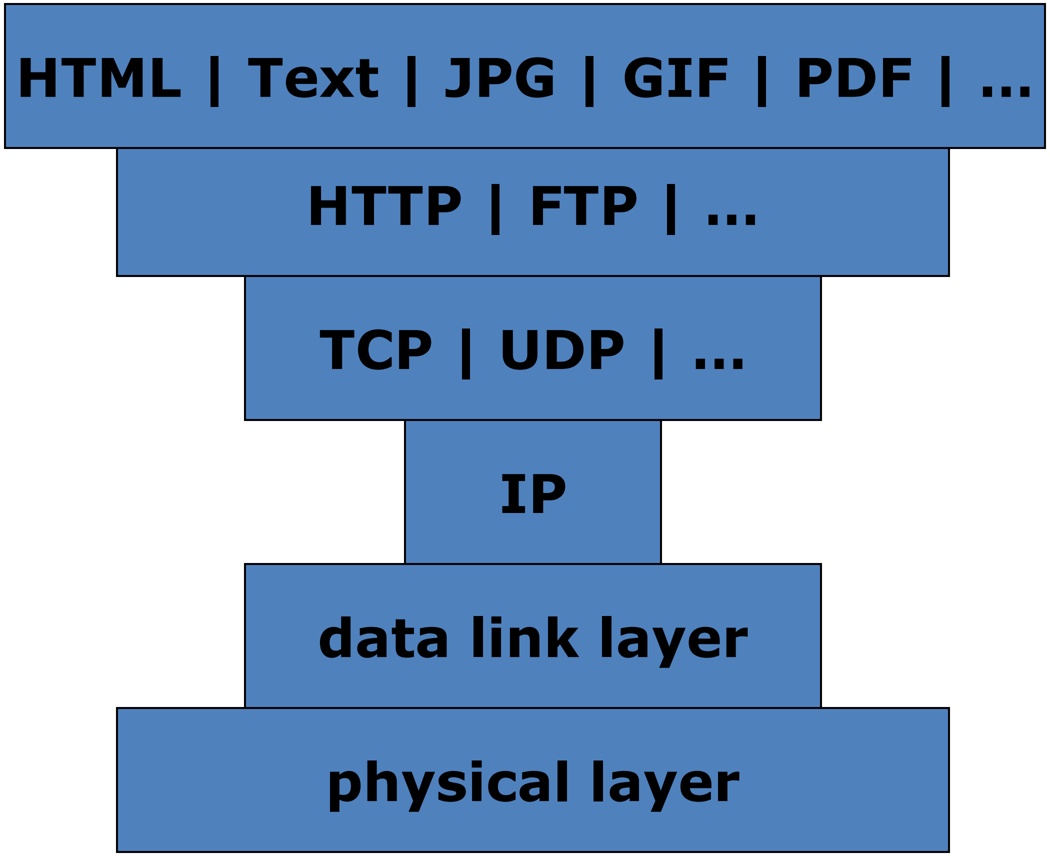
institute for SOFTWARE RESEARCH

Administrivia

- HW 6 out tomorrow morning - concurrency
- No reading for next week (Carnival!)

JAVA NETWORKING BASICS

Layers of a network connection



Internet addresses

- For IP version 4 (IPv4), **host address** (IP address) is 4 bytes
  - e.g., 216.58.217.78
  - ~4 billion distinct addresses
- **Hostnames** mapped to host IP addresses via DNS
- **Port** is a 16-bit number (0 – 65535), assigned conventionally
  - e.g., port 80 is the standard port for web servers

- For IP version 6 (IPv6), IP address is 16 bytes
  - e.g., 3601:557:901:ecc0:1180:9217:a491:b6c2
  - ~3×10^{38} possible addresses

MAC Addresses

- 48-bit hardware-specific ID
  - Associated with the Network Interface “Card” (NIC)
- Centrally administered
- Globally unique*
- Isomorphism from host name to IP address to MAC address
  - But don’t count on it!
  - MAC address spoofing
  - NAT
  - etc.

Packet-oriented and stream-oriented connections

- UDP: User Datagram Protocol
  - Conectionless
  - Discrete packets of data (**datagrams**)
  - Unreliable (but usually pretty reliable)
  - *Does* detect data corruption, via packet checksum
- TCP: Transmission Control Protocol
  - Reliable data **stream**
  - Session-oriented
  - Ordered sequence of bytes
  - Error-checked – a lot going on under the covers!

What is a socket?

- An **endpoint** in a network connection
  - Used to send and/or receive data
- Transport protocol: TCP or UDP (or Raw IP, but not in Java)
- Socket address: local IP address and port number
  - And possibly remote address
- **Sockets make network I/O feel like file I/O**
  - Support read, write, open, and close operations
  - Consistent with Unix philosophy "Everything's a file."
  - History: first appeared In Berkeley (BSD) Unix in 1983
- Java model is a bit different from underlying Unix model
  - Glosses over *socket pairs*
  - Adds notion of server socket (*factory pattern*)

TCP networking in Java – `java.net`

- **IP Address – `InetAddress`**

```
static InetAddress getByName(String host);
static InetAddress getByAddress(byte[] b);
```

- **Ordinary socket – `Socket`**

```
Socket(InetAddress addr, int port);
InputStream  getInputStream();
OutputStream getOutputStream();
void         close();
```

- **Server socket – `ServerSocket`**

```
ServerSocket(int port);
Socket       accept();
void         close();
```

…

Crappy socket demo – chat program (1/2)

Main program – client and server

```
public static void main(String[] args) throws IOException {
    Socket socket;
    if (args.length == 2) {  // We're the client
        InetAddress host = InetAddress.getByName(args[0]);
        int port = Integer.parseInt(args[1]);
        socket = new Socket(host, port);
    } else {  // We're the server
        int port = Integer.parseInt(args[0]);
        ServerSocket serverSocket = new ServerSocket(port);
        socket = serverSocket.accept();
    }

    InputStream socketIn = socket.getInputStream();
    new Thread(() -> copyLines(socketIn, System.out)).start();

    copyLines(System.in, socket.getOutputStream());
}
```

Crappy socket demo – chat program (2/2)

Utility function to copy lines from an input stream to an output stream

```
private static void copyLines(InputStream in, OutputStream out) {
    BufferedReader reader =
        new BufferedReader(new InputStreamReader(in));
    PrintWriter writer = new PrintWriter(out, true);

    // Read a line at a time from reader and copy to writer
    try {
        String line;
        while ((line = reader.readLine()) != null) {
            writer.println(line);
        }
    } catch (IOException e) {
        System.out.println("IO error: " + e);
    }
}
```

Serialization and Sending Objects

- **TODO**

CRASH COURSE ON DISTRIBUTED SYSTEMS

What is a distributed system?

- Multiple system components (computers) communicating via some medium (the network) to achieve some goal
- "Concurrent" (shared-memory multiprocessing) vs. Distributed
  - **Agents**: Threads vs. Processes
    - Processes typically spread across multiple computers
    - Can put them on one computer for testing
  - **Communication**: changes to Shared Objects vs. Network Messages

- (remember the Actor model)

What is a distributed system?

Another definition

Received: by jumbo.dec.com (5.54.3/4.7.34) id AA09105
Date: Thu, **28 May 87** 12:23:29 PDT
From: lamport (**Leslie Lamport**)
Message-Id: <8705281923.AA09105@jumbo.dec.com>
To: src-t
Subject: distribution

There has been considerable debate over the years about what constitutes a distributed system. It would appear that the following definition has been adopted at SRC:

> **A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.**

[Remainder omitted]

Why build a distributed system?

- Unlimited scaling
  - Can be used for capacity or speed
- Geographical dispersion – people and data around the world
- Robustness to failures including physical catastrophes

Challenges

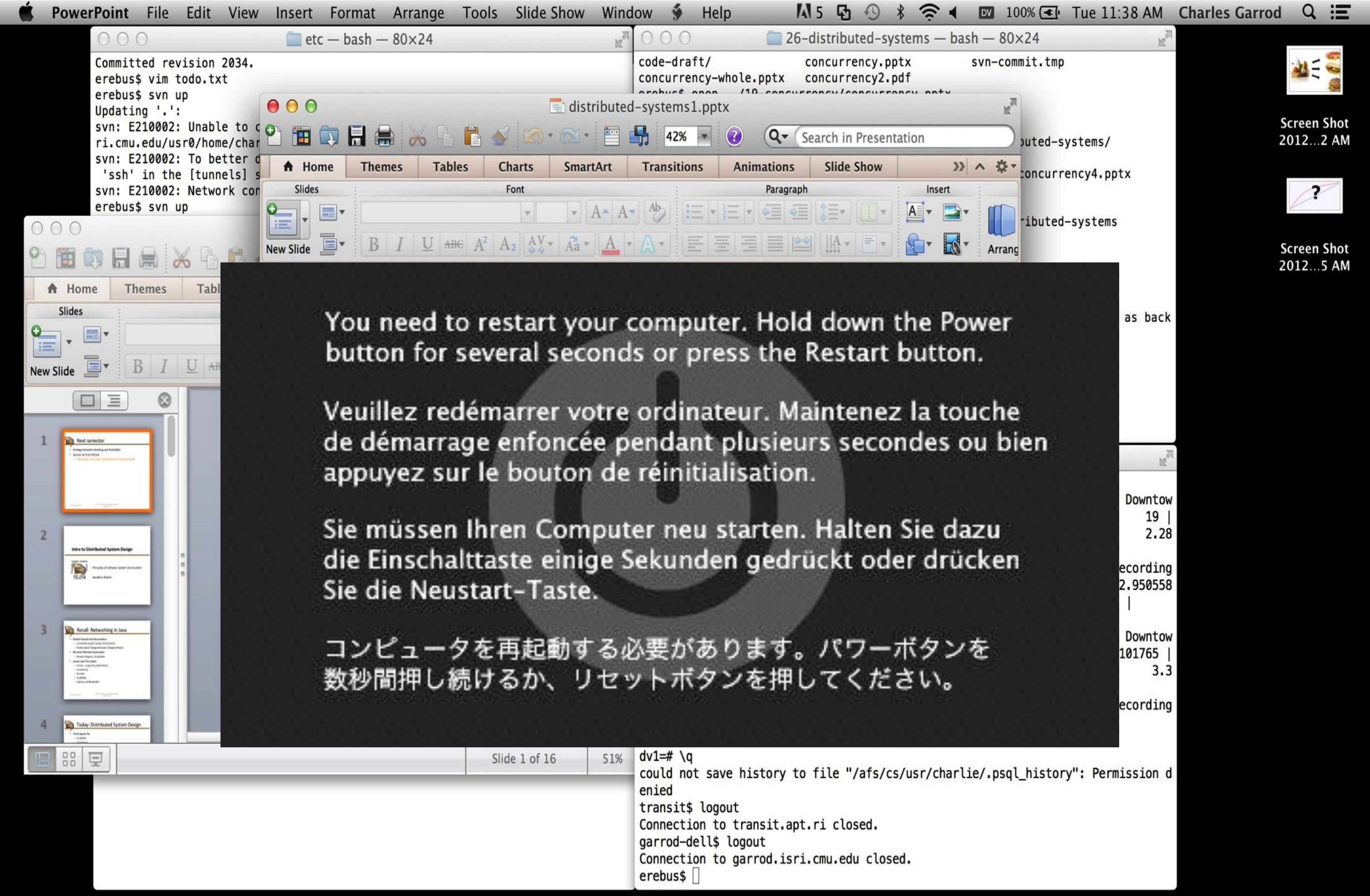
- Scale
- Concurrency
- Geography
- Failures
- Heterogeneity
- Security

Higher levels of abstraction

- Application-level communication protocols
  - HTTP, HTTPS, FTP, etc.
- Frameworks for remote computation
  - Remote Procedure Call (RPC)
  - Java Remote Method Invocation (RMI)
- Common distributed system architectures and primitives
  - e.g., distributed consensus, transactions, replication
- Complex computational frameworks
  - e.g., distributed map-reduce

Metrics of success

- Reliability – works well
  - Often in terms of availability: fraction of time system is working
    - 99.999% available is "5 nines of availability"
- Performance – works fast
  - Low latency
  - High throughput
- Scalability – adapts well to increased demand
  - Ability to handle workload growth


PowerPoint File Edit View Insert Format Arrange Tools Slide Show Window Help
Tue 11:38 AM Charles Garrod
etc — bash — 80×24
Committed revision 2034.
erebus$ vim todo.txt
erebus$ svn up
Updating '.':
erebus$ svn up
26-distributed-systems — bash — 80×24
code-draft/ concurrency.pptx svn-commit.tmp
concurrency-whole.pptx concurrency2.pdf
distributed-systems1.pptx
Home Themes Tables Charts SmartArt Transitions Animations Slide Show
Slides Font Paragraph Insert
New Slide
Search in Presentation
You need to restart your computer. Hold down the Power button for several seconds or press the Restart button.
Veuillez redémarrer votre ordinateur. Maintenez la touche de démarrage enfoncée pendant plusieurs secondes ou bien appuyez sur le bouton de réinitialisation.
Sie müssen Ihren Computer neu starten. Halten Sie dazu die Einschalttaste einige Sekunden gedrückt oder drücken Sie die Neustart-Taste.
コンピュータを再起動する必要があります。パワーボタンを数秒間押し続けるか、リセットボタンを押してください。
Slide 1 of 16
51%
dv1=# \q
could not save history to file "/afs/cs/usr/charlie/.psql_history": Permission denied
transit$ logout
Connection to transit.apt.ri closed.
garrod-dell$ logout
Connection to garrod.isri.cmu.edu closed.
erebus$
Screen Shot 2012…2 AM
Screen Shot 2012…5 AM

Types of failure behaviors

- Fail-stop
- Other halting failures
- Communication failures
  - Send/receive omissions
  - Network partitions
  - Message corruption
- Data corruption
- Performance failures
  - High packet loss rate
  - Low throughput
  - High latency
- Byzantine failures

Common bogus assumptions about failures

- Behavior of others is fail-stop
- Network is reliable
- Network messages are not corrupt
- Failures are independent
- Local data is not corrupt
- Failures are reliably detectable

Some distributed system design principles

- The end-to-end principle
  - When possible, implement functionality at the end nodes (rather than middle nodes) of a distributed system
  - Must confirm success at endpoints; little benefit in redundant work along path
  - Build reliable systems from unreliable parts
  - Canonical example: TCP atop UDP
- The robustness principle (AKA Postel’s law)
  - “Be conservative in what you send, be liberal in what you accept”
- Avoid single points of failure with redundancy
  - Data replication
  - Error detecting / correcting codes (e.g., checksums, Hamming codes)
- Balance load by sharding

Summary

- Network programming in Java is easy compared to C
  - We've seen a simple TCP program
  - UDP is equally easy
- Distributed systems provide scalability and reliability
- But they also provide complexity and headaches
- Abstractions to reduce the complexity:
  - Protocols – UDP, TCP, HTTP
  - Computational primitives – RPC, transactions
  - Computational frameworks – mapreduce