Characterization of Distributed Systems

Prof. Dave Bakken

Cpt. S 464/564 Lecture

Textbook, Chapter 1, plus extras

August 28+30, 2000

Administrative Items

- Handouts today
  - Syllabus
  - Student Survey
  - Paper “A Note on Distributed Computing”
  - Paper “Interview: Fred B. Schneider on Distributed Computing”
    - Testable for 564 students only
  - Lecture slides for first two lectures (this document!)
- Textbooks are not in yet! (Hopefully by Wednesday.)
- Conventions in these slides
  - Key terms defined are underlined
  - Items for extended discussion are in red
    - Or URLs underlined and in red by PowerPoint
  - `Code fragments or something else you might type are in a typewriter font (Courier New)`

Outline of Topics

1. **Introduction**
2. Examples of Distributed Systems
3. Resource Sharing and the Web
4. Challenges
5. Example Local vs. Remote Procedure Call

Introduction

- A distributed system is "one in which hardware or software components located at networked computers communicate and coordinate their actions only by message passing"
  - Very broad definition
  - Lots of examples
  - Lots of kinds
  - Note: I abbreviate "Distributed System" by "DS"
- "You know you have one when the crash of a computer you've never heard of stops you from getting any work done." Leslie Lamport
- Examples of DSs:
  –
  –
  –
  –
  –
  –

Advantages of Distributed Systems

- Share resources (key)
- Share devices
- Better hardware cost/performance than supercomputers, multiprocessors
- Allows access from many remote users using their simple PCs
- Allows for incremental growth (if done right)
- Increases reliability and availability (if done right)
- Some applications and services are inherently distributed
- Can spread the load of a given service much more easily
- Can potentially increase security (!!!???)

Consequences of Distributed Systems

- Concurrency
  - Concurrent use of low-level resources: processing, storage (memory+disk), communications
  - Mutual exclusion and other synchronization required
  - Access to resources for a given user often best-effort
- No global clock
  - Cannot often know the exact ordering of events: which happened first
- Independent failures
  - No longer “all or none” failures for your program!
  - Some computers still running, while others failed or partitioned
  - Failure of a component you are using may not be a clean failure
- “I don’t think we’re in Kansas anymore, Toto!”

Outline of Topics

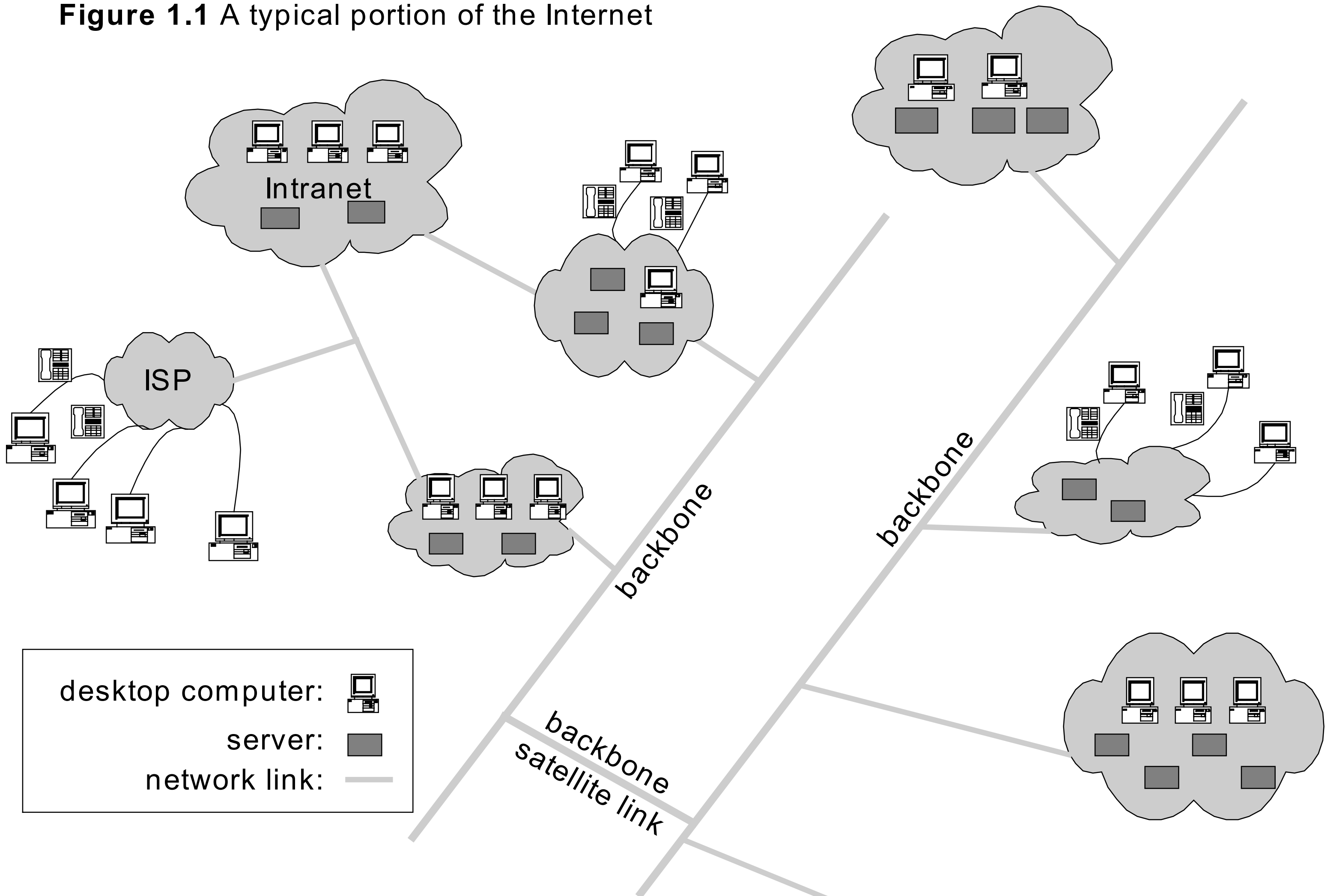
1. Introduction
2. **Examples of Distributed Systems**
3. Resource Sharing and the Web
4. Challenges
5. Example Local vs. Remote Procedure Call

DS Example 1: The Internet

- Large number of connected computers
- Common protocols
- Common addressing
- Common set of basic services
- It is a very large distributed system!
- Q: What defines “The Internet”

The Internet (cont.)

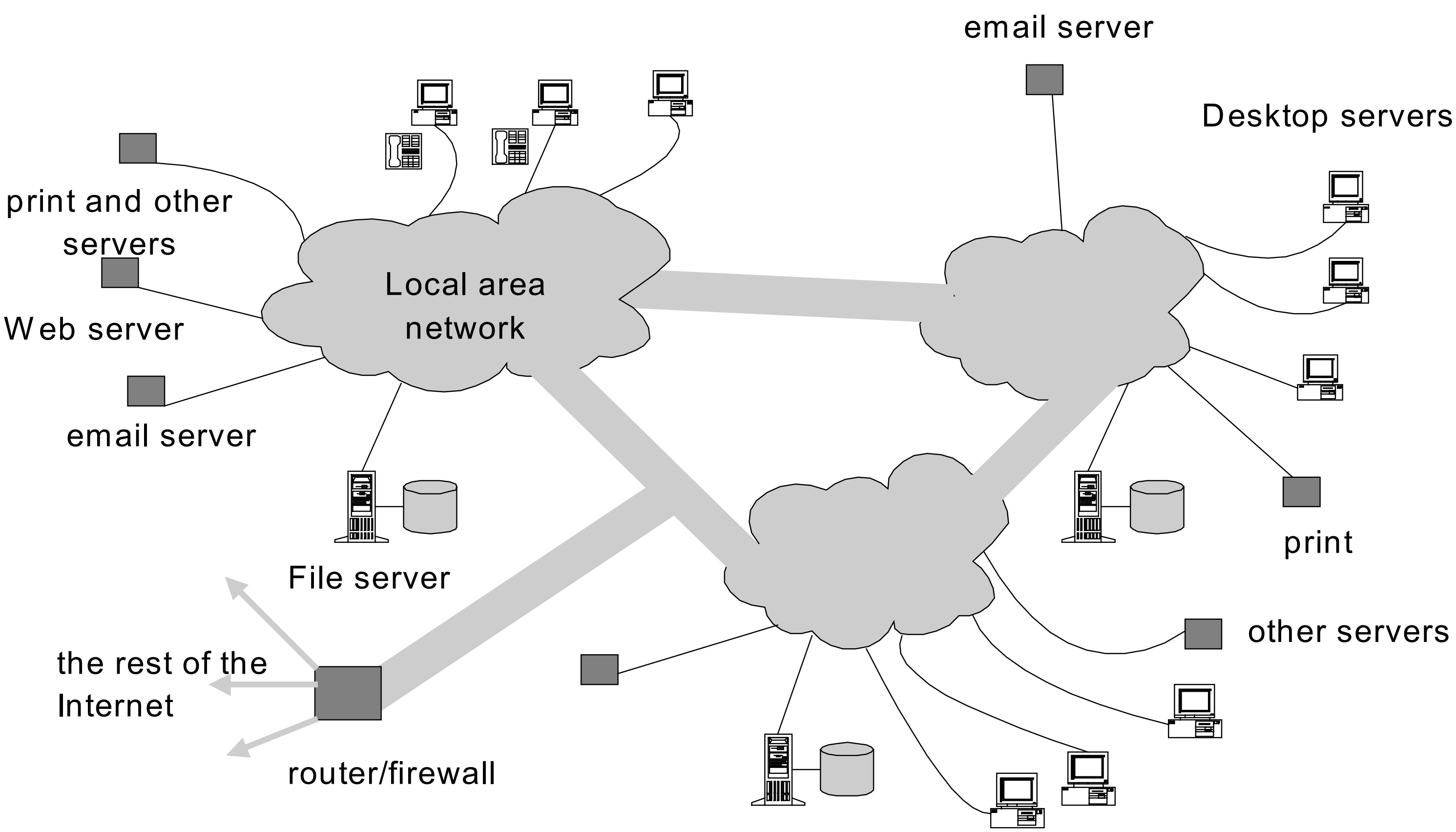
Figure 1.1 A typical portion of the Internet



DS Example 2: Intranets

- <u>Intranet</u> == "a portion of the Internet that is separately administered and has a boundary that can be configured to enforce local security policies"

Figure 1.2 A typical intranet



Intranets (cont.)

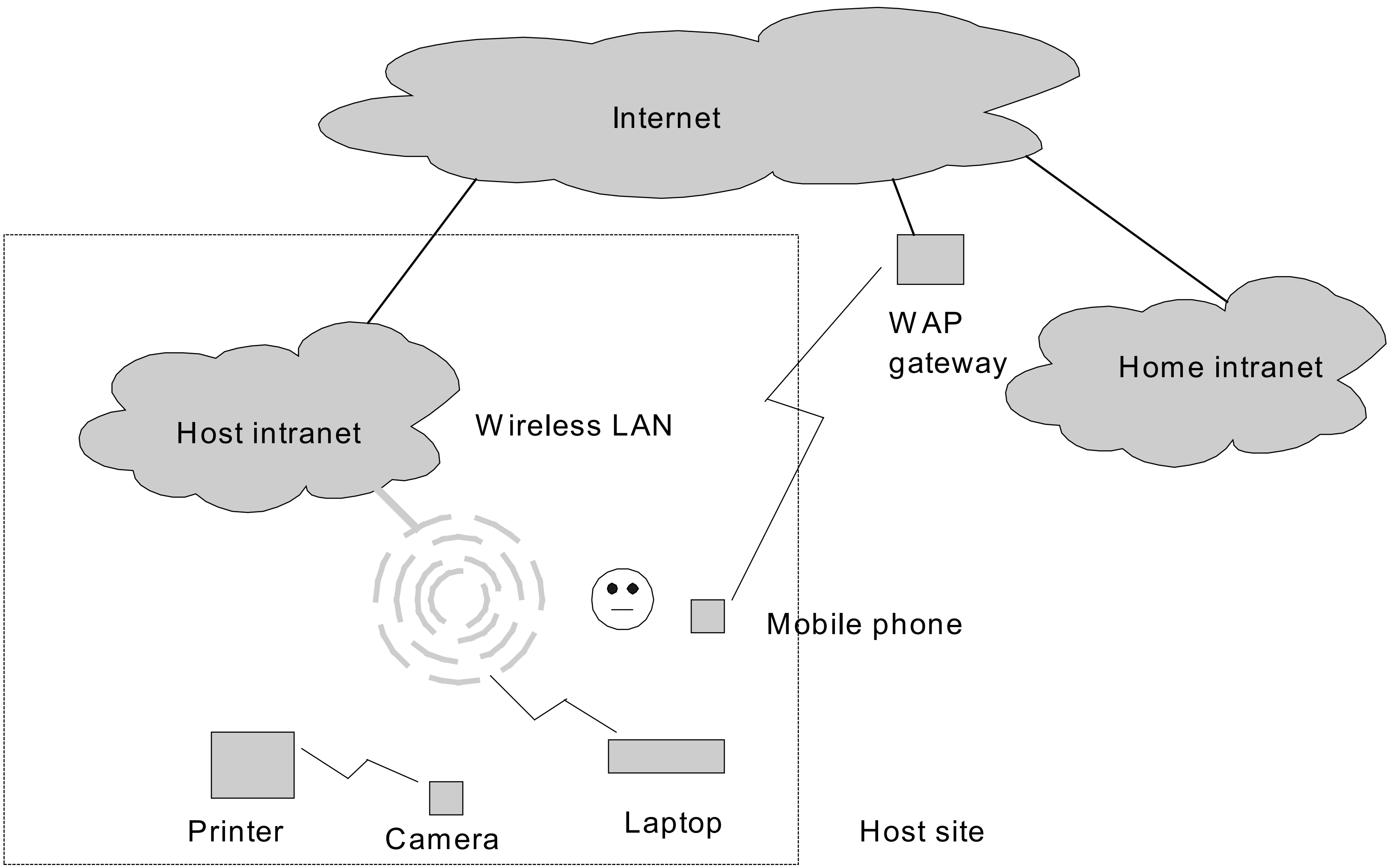
- Usually protected by a firewall
- Issues involving design of components in an intranet
  - File services are needed
  - Firewalls tend to impede legitimate access to services
    - Need finer-grained security mechanisms
  - Cost of software installation and support is very important!

DS Example 3: Mobile and Ubiquitous Computing

- Lots of small and/or portable computing devices are networked
  - Laptop computers
  - Personal Digital Assistants (PDAs) like a Palm Pilot
  - Mobile phones
  - Video cameras
  - Wearable devices (smart watch, "Dick Tracy" ring)
  - Devices embedded in appliances
- <u>Mobile computing</u>
  - Ability to perform computing tasks when user is moving from his normal environment
  - Also known as (aka) "nomadic computing"
- <u>Ubiquitous computing</u>
  - Harnessing plentiful small and cheap computing devices that are (seemingly) everywhere
  - Only really useful if they
    - Are networked
    - Can easily be used in programs by novices
- Mobile and ubiquitous computing overlap, but not 100%

Mobile and Ubiquitous Computing (cont.)

Figure 1.3 Portable and handheld devices in a distributed system



Outline of Topics

Resource Sharing and the Web

- Its easy to overlook benefits of resource sharing!
- Great to share low-level resources (storage, etc), services, and peripherals
- But more important to share higher-level services like databases, etc.
- Patterns of resource sharing vary quite widely in
  - Geographic scope
  - How closely users work together
- Service == "a distinct part of a computer system that manages a collection of related resources and presents their functionality to users and applications"
  - Examples of distributed services….
- Server == "a running program (a process) on a networked computer that accepts requests from programs running on other computers to perform a service and responds appropriately"
  - A <u>client</u> *invokes an operation* on a server; "remote invocation"
  - "Client" and "server" usually refer to processes, sometimes to objects, occasionally principals/users

The World Wide Web (WWW)

- System for publishing and accessing resources and services across the Internet
  - WWW is probably what >90% of non-techies think of as the Internet!
- Began at CERN in Switzerland in 1989 to share documents
- Uses idea of <u>hypertext</u>: non-linear ways to use a document via <u>links</u>
  - Hypertext has been around since 1945…
- The web is an <u>open system</u>: can be extended
  - Based on freely available standards for communication and documents
  - Can add new kinds of resources to the web
- Web is based on 3 technologies:
  - HyperText Markup Language (HTML)
  - Uniform Resource Locators (URLs)
  - HyperText Transport Protocol (HTTp)d

HyperText Markup Language (HTML)

- Used to specify
  - Text and images that make up contents of a web page
  - How these are laid out and formatted
- A simple but low-level language
  - Often not directly programmed but generated
- Contains structured items specified by <u>tags</u>
  - Headings: `<h1>, <h2>, …`
  - Paragraphs: `<p>`
  - Images: `<img…>`
  - Links: `<a href … >`
  - ….
- Stored on a web server

Uniform Resource Locators (URLs)

- Used to identify a resource so a browser can locate it
- Format: `scheme : scheme-specific-location`
- Scheme declares what type of URL this is
- Scheme-specific-location tells where that scheme can find it
- Examples
  - `mailto:` bakken@eecs.wsu.edu
  - http://www.eecs.wsu.edu/~bakken
  - ftp://ftp.parc.xerox.com/Programs/ILU.exe
  - Other schemes: `nntp`, `telnet`
- Extensible by
  - Adding a new scheme
  - Developing and disseminating helper applications or plugins for the scheme
- HTTP URL is most common…
  - http://servername[:port][/pathNameOnServer][?arguments]
  - Anchors — # — are part of HTLM spec., not URL spec.

HyperText Transfer Protocol (HTTP)

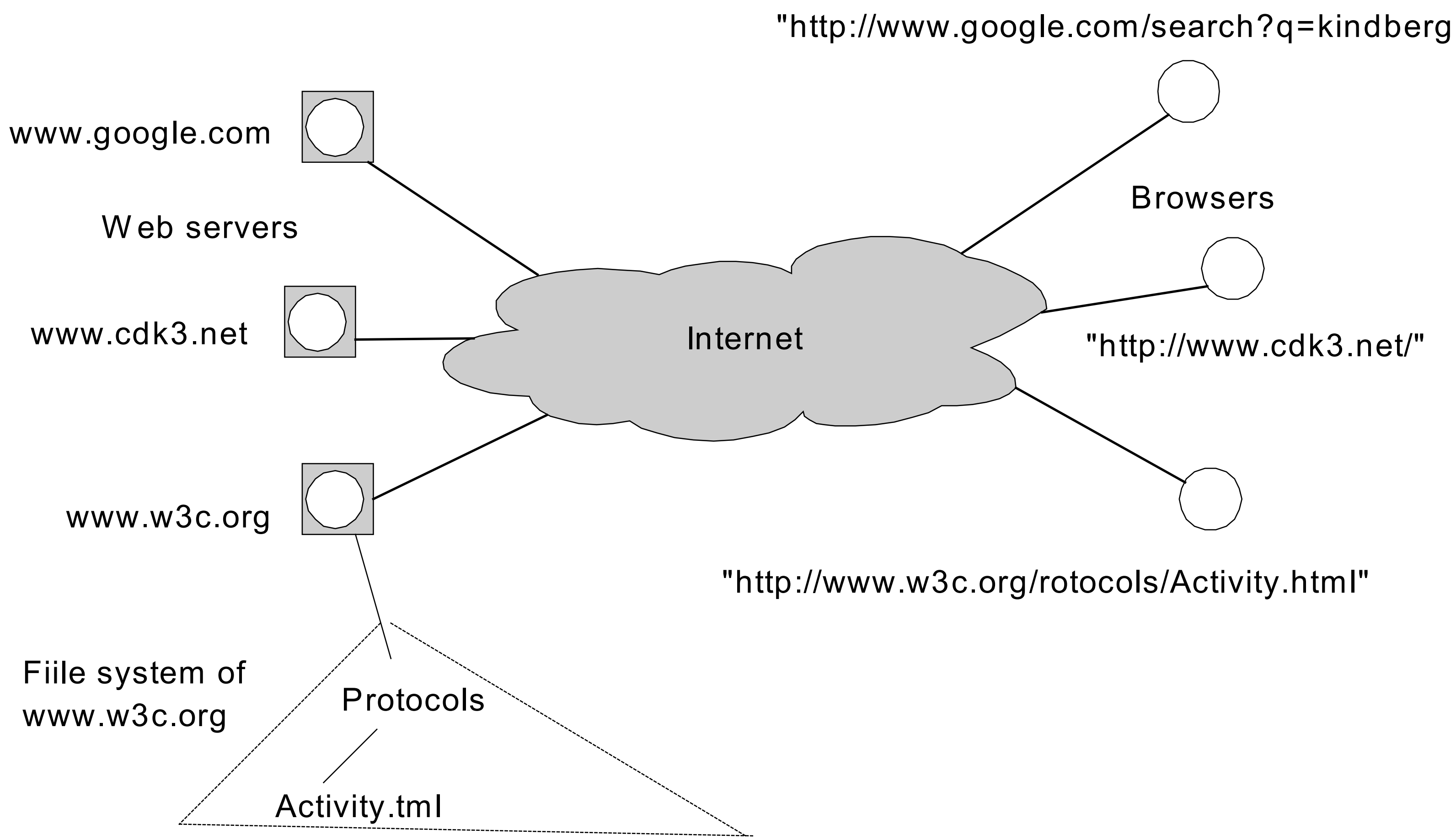
- HTTP is the protocol browsers and other clients interact with web servers
- Very low-level and dumb…. (version 1.0 anyway)
- Request-reply protocol
- Preferred content types (ones a browser can handle) sent with request
- One resource (html for the page, image, etc) per request
- Very simple access control
  - Configuration file to deny or allow access based on client’s host or domain
  - Passwords

Advanced Features of the WWW

- Services and dynamic content
  - URL points to a program, not a file, that processes client's inputs
    - Often with input from a form
    - Often html is sent back for the client to present to the user
  - Program is often called a Common Gateway Interface (CGI)
- Downloaded code
  - JavaScript or Java applet to run on client machine
  - Can provide
    - Error checking locally before a request goes over the network
    - Much richer and quicker graphics than possible across web
- MetaData and the Web
  - Searching by an ASCII string is not always useful….
  - MetaData (data about data) is being developed and deployed
  - Resource Description Language standardized for this, not yet widely used
  - Semi-structured data (Prof. Curtis Dyreson)
- XML (later this course)
  - Meta-language for describing data
  - Helps make data portable between applications

WWW Example

Figure 1.4 Web servers and web browsers



Outline of Topics

1. Introduction
2. Examples of Distributed Systems
3. Resource Sharing and the Web
4. **Challenges**
    1. **Heterogeneity**
    2. **Openness**
    3. **Security**
    4. **Scalability**
    5. **Failure Handling**
    6. **Concurrency**
    7. **Transparency**
5. Example Local vs. Remote Procedure Call

Heterogeneity

- Heterogeneity == variety or difference
- In DSs this is in a number of dimensions
  - Networks
  - Hardware
  - Operating Systems
  - Programming Languages
  - Different implementations of the same standard or protocol
- Dealing with this difference is very difficult!
- Middleware == a layer of software above the operating system which
  - Provides a programming abstraction (a higher-level building block) across the network
  - Masks the heterogeneity in above dimensions (some or all of them)
- Middleware examples
  - CORBA
  - DCOM/COM+
  - Java Virtual Machine (JVM)

Openness

- <u>Openness</u> == “the characteristic that determines whether the system can be extended and re-implemented in various ways”
- Key interfaces are published
- Hardware extensions: adding new computers to the DS
- Software extensions: adding new services or new implementations of existing services

Security

- Some information has high value to its owners (and others!)
- Dimensions of security
  - Confidentiality
  - Integrity
  - Availability
- Confidentiality
  - Protection of disclosure of information to unauthorized parties
  - I.e., lack of invalid reading of data
- Integrity
  - Protection against unauthorized alteration
  - I.e., lack of invalid writing of data
- Availability
  - Protection against valid users being able to access the data or service
  - I.e., lack of denial of service
- Denial of service attacks are an increasing problem
- Security of mobile code is crucial, too

Scalability

- System is scalable if it is still useable/effective with an increase in
  - Number of resources
  - Number of users
- Scalability challenge #1: controlling the cost of physical resources
  - Extending the system should be doable at reasonable cost
  - I.e, O(n)
- Scalability challenge #2: controlling the performance loss
  - The more resources, the more the cost to access one of them
  - But hopefully it can grown O(log n) at worst
- Scalability challenge #3: preventing software resources from running out
  - E.g,. 32-bit IP addresses
- Scalability challenge #4: avoiding performance bottlenecks
  - Centralized servers or "hot spots" in network do not scale
  - E.g., Domain Name Service (DNS) replicated and cached
- Scalability goal: design system so that system and existing application software do not need to change when growth occurs

Failure Handling

- Failures in a non-distributed system (a PC not networked) are often total failures (or can be treated as such)
- DS failures are often partial failures: some components fail while others remain in operation
- Technique #1: failure detection: Finding out one happened!
- Technique #2: failure masking: Hide or lessen impact of failure
  - Retransmit a message (redundancy in time)
  - Replicate the data (redundancy in space)
- Technique #3: failure toleration: write middleware and applications to explicitly deal with a partial failure (rather than ignoring the possibility)
- Technique #4: failure recovery: recover the state of the server after it has failed

Concurrency

- A DS has a lot of sharing happening (concurrently, not serially)
  - That is its *raison d'être*!
  - Sharing of applications
  - Sharing of servers
  - Sharing of low-level resources
  - ….
- Bottom line: any shared component must be written so that it operates correctly in a concurrent environment

Transparency

- Transparency == concealing distribution from the user and/or application
- Different kinds of transparencies:
  - Access transparency: local and remote resources can be accessed using an indentical operation
  - Location transparency: resources can be accessed without knowledge of their location
  - Concurrency transparency: several processes can operate concurrently using a shared resource or service without interference between them (or being aware of the concurrency in any other way)
  - Replication transparency: multiple copies can be deployed without the programmers or users having to be aware of it
  - Failure transparency: users and application programs can complete their activities despite hardware or software failures

Transparency (cont.)

- Different kinds of transparencies (cont.)
  - <u>Mobility transparency</u>: clients and systems resources can move within the system without users or applications being aware (or having to handle it)
  - <u>Performance transparency</u>: the system can be reconfigured to improve performance as loads vary, without the client or user having to be aware
  - <u>Scaling transparency</u>: the system and applications can expand in scale without the system structure or application programs being aware
- Note: middleware not only supports heterogeneity but also transparency
- Examples
  - GUI for file system (files and folders) which has same look and feel for local and remote files
  - Library for file operations where API is same for local and remote files
  - Library for file operations which makes you use ftp for remote files but procedure call for local ones
  - URLS
    - Location transparent?
    - Mobility trasparent?
    - Replication transparent?

Transparency (cont.)

- So, then, is transparency always
  - Feasable?
  - Desirable?
  - Free?
- Its always a tradeoff, usually against performance, and sometimes against other dimensions of transparency
- Understanding these tradeoffs is a key goal for this course!

Outline of Topics

Example Local Call

Caller:

```
// declare and init stuff
x = new int [100];
Y = new util;
Flag = y.sort(x, 100);
```

Callee:

```
// declare and init stuff
Int util:sort(int [] a, int max) {
     // implementation of sort
   return status;
}
```

- Potential assumptions:
  - Procedure call conventions between caller (“client”) and callee
  - In same address space (on same computer)
  - In same programming language (usually)
  - Written by same programmer (often, not always)
  - Can transfer data and control quickly, effectively in zero time
  - Both fail, or neither do (for the most part)
- None of these assumptions are true in a distributed system!

Example Remote Call

Caller:

```
// declare and init stuff
x = new int [100];
Y = new util.bind();
Flag = y.sort(x, 100);
…
```

```
// "proxy" or "stub"
// generated by middleware
int util:sort(int [] a, int max){
  // put a[], max into struct
  // send message with struct
  // receive message w/ struct
  // copy from struct to a[],
  // status
    return status;
}
```

Callee:

```
// declare and init stuff
int util_impl:sort(int [] a,
                     int max)      {
      // implementation of sort
    return status;
}
```

```
// "skeleton" generated
// by middleware compiler
  …
  // receive message with struct
  // copy from struct to a[], max
  flag = z.sort(a, max)

 // copy a[], flag into struct
 // send message with struct
 …
```

Many Assumptions do not Hold!

- Not a local procedure call, so need more help
- Not in same programming language (can't assume this)
- Not written by same programmer
- Not always in the same administrative domain
- Latency for transfer of control and data can be large and, worse, unpredictable
- Partial failures
- Membership of the system (the computers in its collection) can change
- Unreliable or insecure communication

- One more important item……

Washington State University

One Important Conclusion!

GO Cougs!

(DeColor???) the Cardinal!

Cougars