

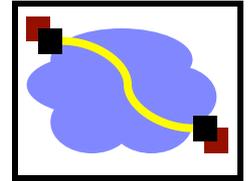
416 Distributed Systems

Networks review; Day 2 of 2

And start of RPC

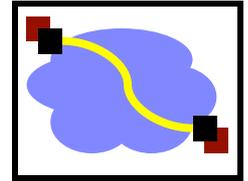
Jan 13, 2016

Last Time

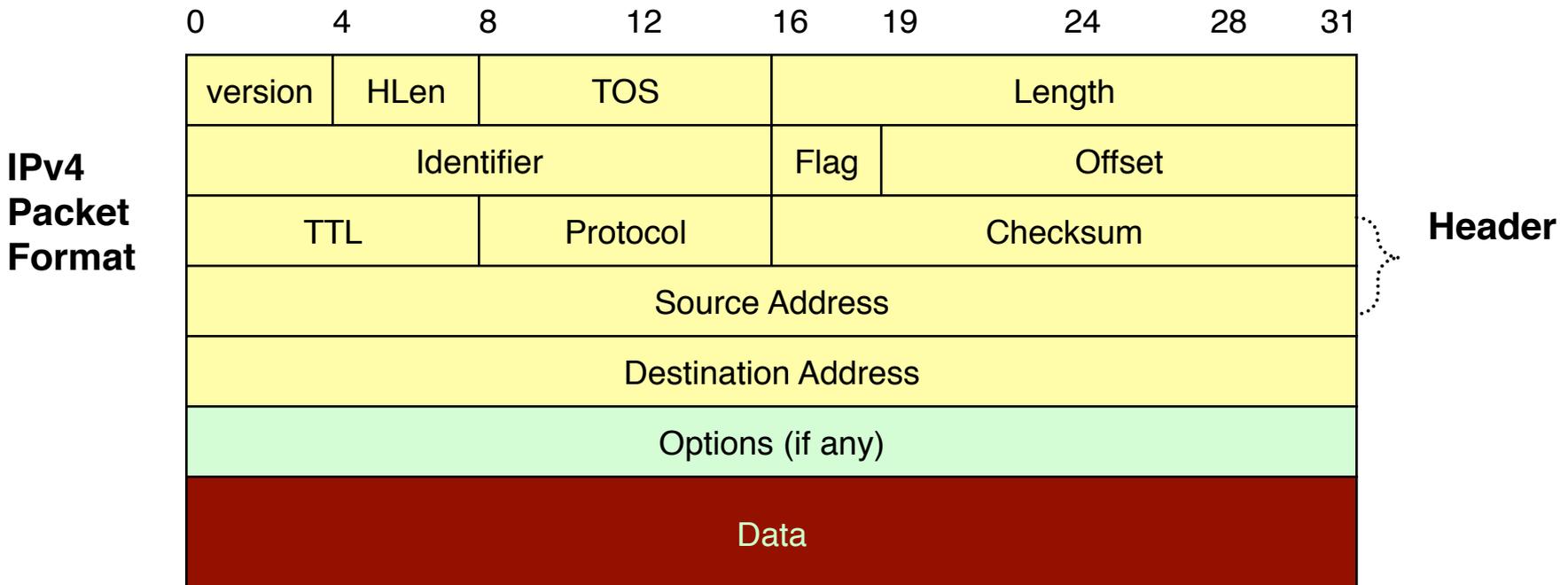


- Modularity, Layering, and Decomposition
 - Example: UDP layered on top of IP to provide application demux (“ports”)
- Resource sharing and isolation
 - Statistical multiplexing - packet switching
- Dealing with heterogeneity
 - IP “narrow waist” -- allows many apps, many network technologies
 - IP standard -- allows many impls, same proto

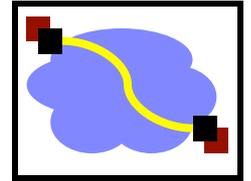
IP Packets/Service Model



- Low-level communication model provided by Internet
- Datagram
 - Each packet self-contained
 - All information needed to get to destination
 - No advance setup or connection maintenance
 - Analogous to letter or telegram



Goals [Clark88]



0 Connect existing networks

initially ARPANET and ARPA packet radio network

1. Survivability

ensure communication service even in the presence of network and router failures

2. Support multiple types of services

3. Must accommodate a variety of networks

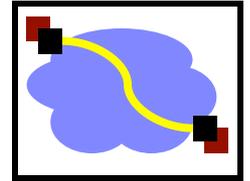
4. Allow distributed management

5. Allow host attachment with a low level of effort

6. Be cost effective

7. Allow resource accountability

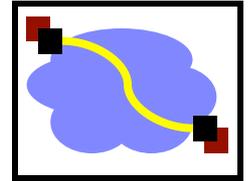
Goal 1: Survivability



- If network is disrupted and reconfigured...
 - Communicating entities should not care!
 - No higher-level state reconfiguration
- How to achieve such reliability?
 - Where can communication state be stored?

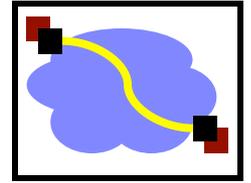
	Network	Host
Failure handing	Replication	“Fate sharing”
Net Engineering	Tough	Simple
Routing state	Maintain state	Stateless
Host trust	Less	More

Fate Sharing



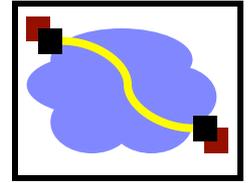
- Lose state information for an entity if and only if the entity itself is lost.
- Examples:
 - OK to lose TCP state if one endpoint crashes
 - NOT okay to lose if an intermediate router reboots
 - Is this still true in today's network?
 - NATs and firewalls
- Tradeoffs
 - Less information available to the network
 - Must trust endpoints more

Networks [including end points] Implement Many Functions



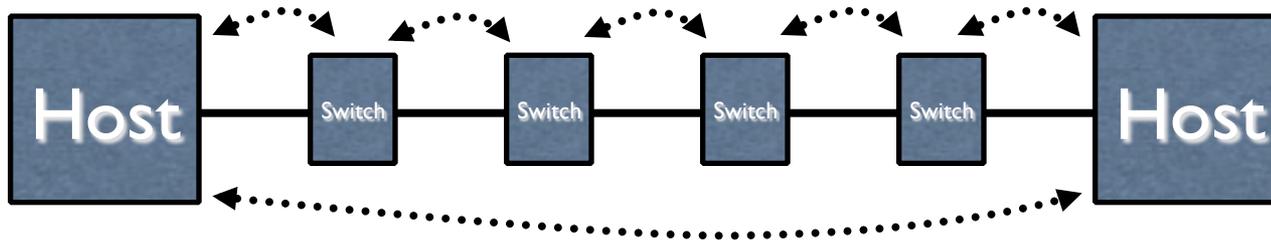
- Link
- Multiplexing
- Routing
- Addressing/naming (locating peers)
- **Reliability**
- Flow control
- Fragmentation
- Etc.....

Design Question



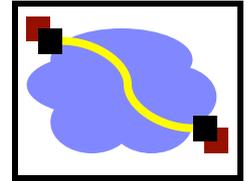
- If you want reliability, where should you implement it?

Option 1: Hop-by-hop (at switches)



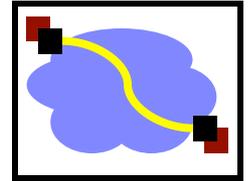
Option 2: end-to-end (at end-hosts)

Options



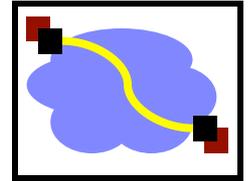
- Hop-by-hop: Have each switch/router along the path ensure that the packet gets to the next hop
- End-to-end: Have just the end-hosts ensure that the packet made it through
- What do we have to think about to make this decision??

A question



- Is hop-by-hop enough?
- [hint: What happens if a switch crashes? What if it's buggy and goofs up a packet?]

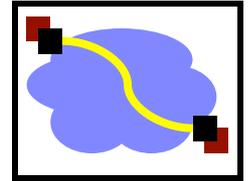
End-to-End Argument



- Deals with **where** to place functionality
 - Inside the network (in switching elements)
 - At the edges
- Guideline not a law
- Argument
 - If you have to implement a function end-to-end anyway (e.g., because it requires the knowledge and help of the end-point host or application), **don't implement it inside the communication system**
 - Unless there's a compelling performance enhancement

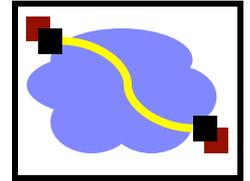
*Further Reading: "End-to-End Arguments in System Design."
Saltzer, Reed, and Clark.*

Questions to ponder



- If you have a whole file to transmit, how do you send it over the Internet?
 - You break it into packets (packet-switched medium)
 - TCP, roughly speaking, has the sender tell the receiver “got it!” every time it gets a packet. The sender uses this to make sure that the data’s getting through.
 - But by e2e, if you have to acknowledge the correct receipt of the entire file... **why bother acknowledging the receipt of the individual packets???**

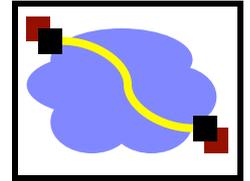
Questions to ponder



- If you have a whole file to transmit, how do you send it over the Internet?
 - You break it into packets (packet-switched medium)
 - TCP, roughly speaking, has the sender tell the receiver “got it!” every time it gets a packet. The sender uses this to make sure that the data’s getting through.
 - But by e2e, if you have to acknowledge the correct receipt of the entire file... **why bother acknowledging the receipt of the individual packets???**
- This is a bit of a trick question -- it’s not asking e2e vs in-network. :)

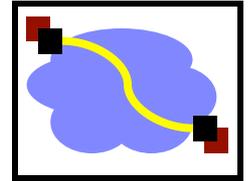
The answer: Imagine the waste if you had to retransmit the entire file because one packet was lost. Ow.

Internet Design: Types of Service



- **Principle:** network layer provides one simple service: best effort datagram (packet) delivery
 - All packets are treated the same
- Relatively simple core network elements
- Building block from which other services (such as reliable data stream) can be built
- Contributes to scalability of network
- No QoS support assumed from below
 - In fact, some underlying nets only supported reliable delivery
 - Made Internet datagram service less useful!
 - Hard to implement without network support
 - QoS is an ongoing debate...

User Datagram Protocol (UDP): An Analogy



UDP

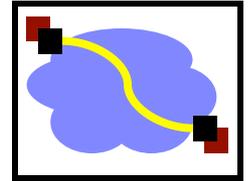
- Single socket to receive messages
- No guarantee of delivery
- Not necessarily in-order delivery
- Datagram – independent packets
- Must address each packet

Postal Mail

- Single mailbox to receive letters
- Unreliable 😊
- Not necessarily in-order delivery
- Letters sent independently
- Must address each letter

Example UDP applications
Multimedia, voice over IP

Transmission Control Protocol (TCP): An Analogy



TCP

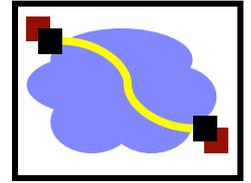
- Reliable – guarantee delivery
- Byte stream – in-order delivery
- Connection-oriented – single socket per connection
- Setup connection followed by data transfer

Telephone Call

- Guaranteed delivery
- In-order delivery
- Connection-oriented
- Setup connection followed by conversation

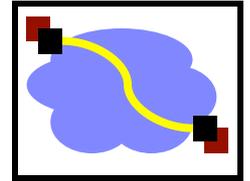
Example TCP applications
Web, Email, Telnet

Why not always use TCP?



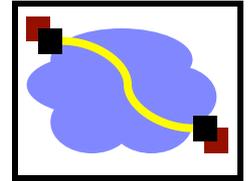
- TCP provides “more” than UDP
- Why not use it for everything??

Why not always use TCP?



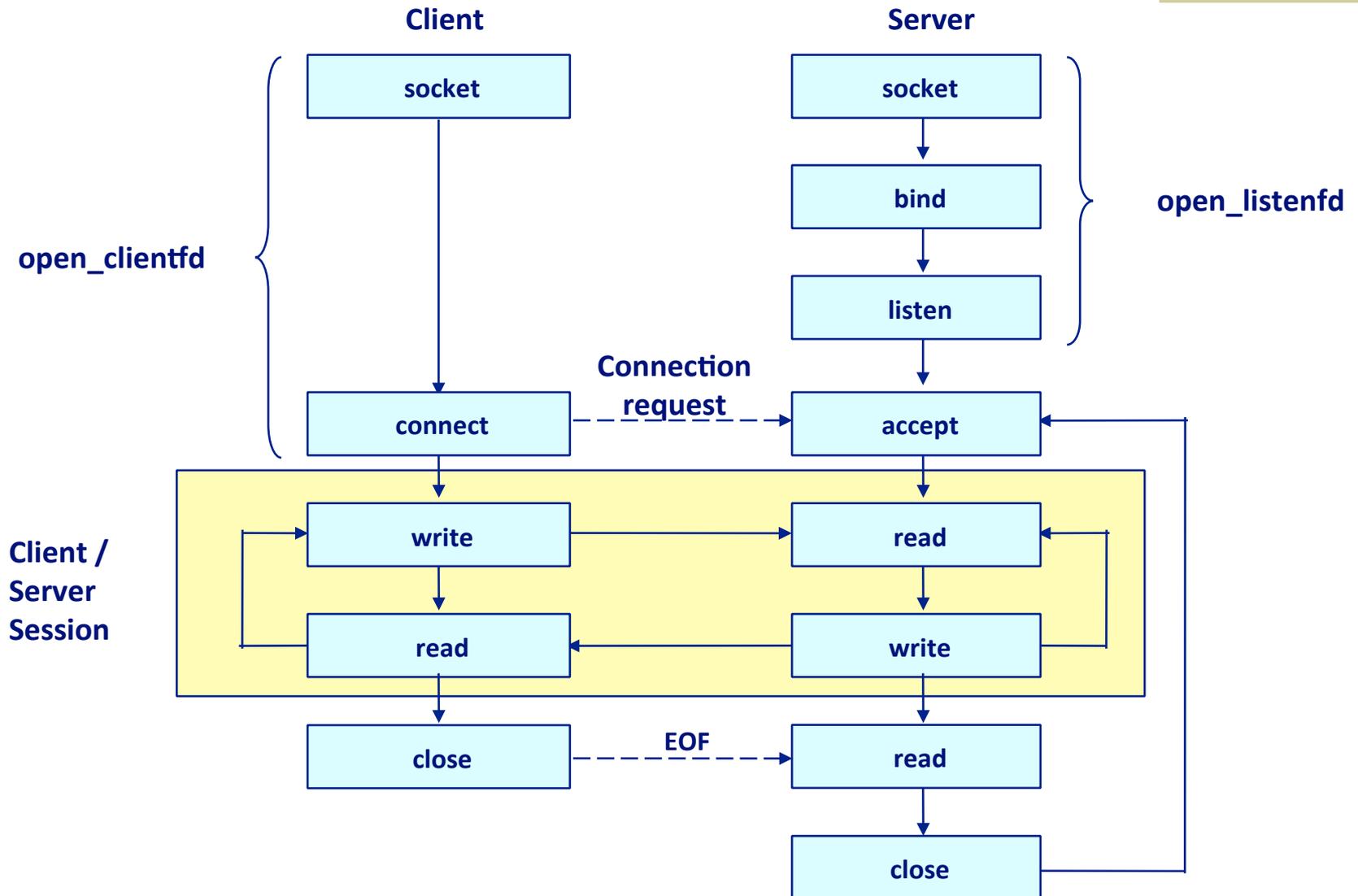
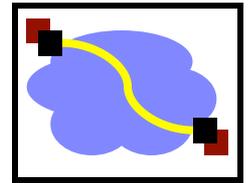
- TCP provides “more” than UDP
- Why not use it for everything??
- A: Nothing comes for free...
 - Connection setup (take on faith) -- TCP requires one round-trip time to setup the connection state before it can chat...
 - How long does it take, using TCP, to fix a lost packet?
 - At minimum, one “round-trip time” (2x the latency of the network)
 - That could be 100+ milliseconds!
 - If I guarantee in-order delivery, what happens if I lose one packet in a stream of packets?
 - Has semantics that may be too strong for the app (e.g., Netflix streaming)

Design trade-off

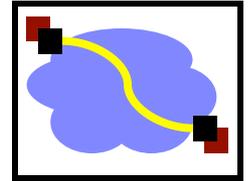


- If you're building an app...
- Do you need everything TCP provides?
- If not:
 - Can you deal with its drawbacks to take advantage of the subset of its features you need?
OR
 - You're going to have to implement the ones you need on top of UDP
 - Caveat: There are some libraries, protocols, etc., that can help provide a middle ground.
 - Takes some looking around

Socket API Operation Overview

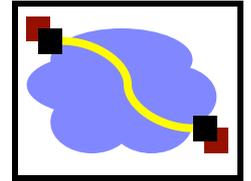


Blocking sockets



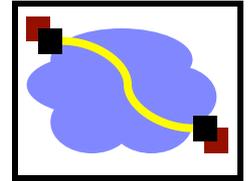
- What happens if an application write(s) to a socket waaaaay faster than the network can send the data?
- TCP figures out how fast to send the data...
- And it builds up in the kernel socket buffers at the sender... and builds...
- until they fill. The next write() call *blocks* (by default).
- What's blocking? It suspends execution of the blocked thread until enough space frees up...

In contrast to UDP

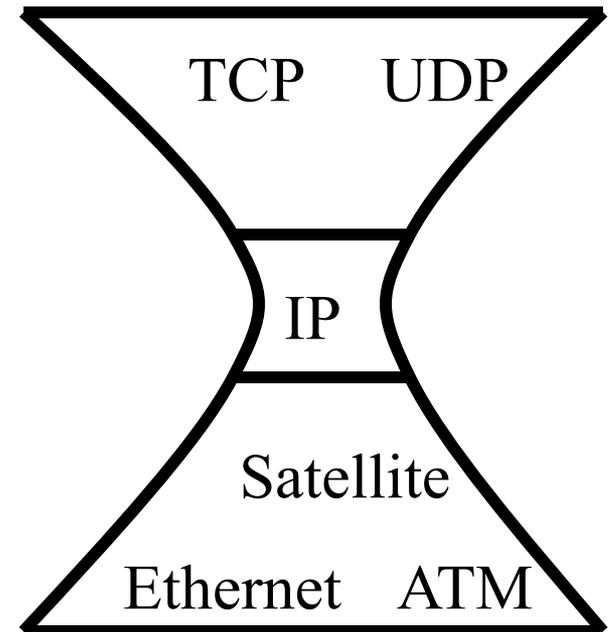


- UDP doesn't figure out how fast to send data, or make it reliable, etc.
- So if you write() like mad to a UDP socket...
- It often silently disappears. *Maybe* if you're lucky the write() call will return an error. But no promises.

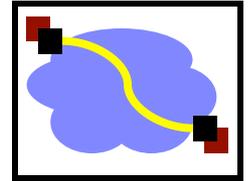
Summary: Internet Architecture



- Packet-switched datagram network
- IP is the “compatibility layer”
 - Hourglass architecture
 - All hosts and routers run IP
- Stateless architecture
 - no per flow state inside network

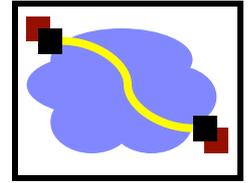


Summary: Minimalist Approach

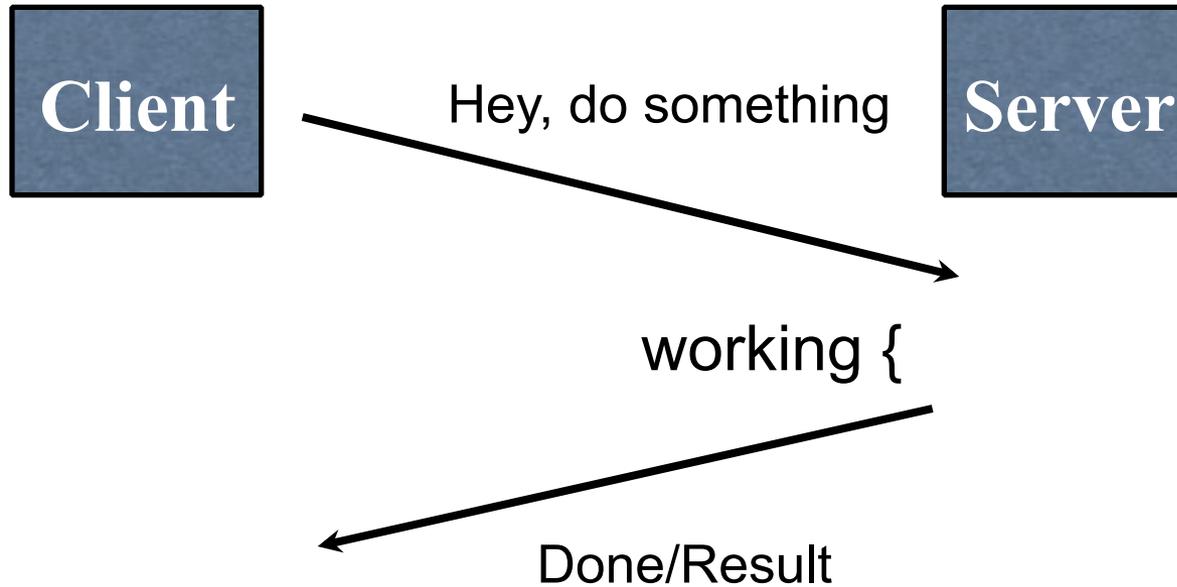
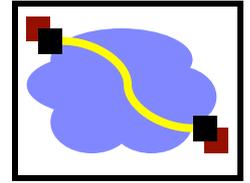


- Dumb network
 - IP provide minimal functionalities to support connectivity
 - Addressing, forwarding, routing
- Smart end system
 - Transport layer or application performs more sophisticated functionalities
 - Flow control, error control, congestion control
- Advantages
 - Accommodate heterogeneous technologies (Ethernet, modem, satellite, wireless)
 - Support diverse applications (telnet, ftp, Web, X windows)
 - Decentralized network administration

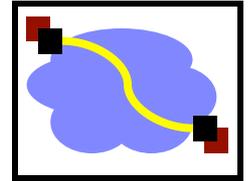
RPC: Remote Procedure Calls



Common communication pattern



Writing it by hand (in C)

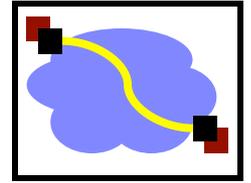


```
struct foormsg {
    u_int32_t len;
}

send_foo(char *contents) {
    int msglen = sizeof(struct foormsg) + strlen(contents);
    char buf = malloc(msglen);
    struct foormsg *fm = (struct foormsg *)buf;
    fm->len = htonl(strlen(contents));
    memcpy(buf + sizeof(struct foormsg),
           contents,
           strlen(contents));
    write(outsock, buf, msglen);
}
```

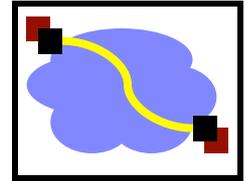
Then wait for response, etc.

RPC land



- RPC overview
- RPC challenges
- RPC other stuff

RPC



- A type of client/server communication
- Attempts to make remote procedure calls look like local ones

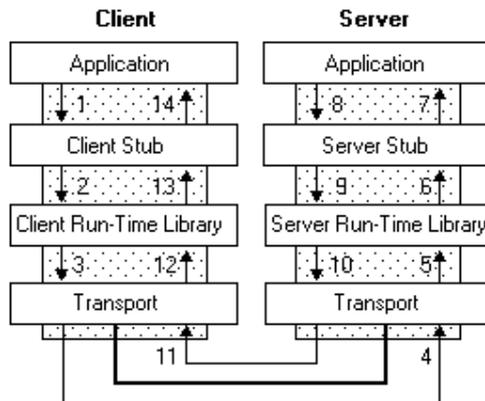
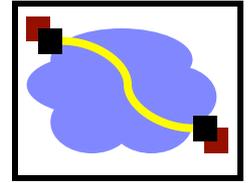


figure from Microsoft MSDN

```
{ ...  
  foo()  
}  
void foo() {  
  invoke_remote_foo()  
}
```

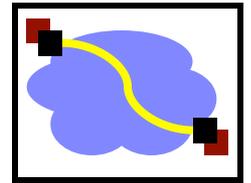
Go Example



- Need some setup in advance of this but...

```
// Synchronous call
args := &server.Args{7,8}
var reply int
err = client.Call("Arith.Multiply", args, &reply)
if err != nil {
    log.Fatal("arith error:", err)
}
fmt.Printf("Arith: %d*%d=%d", args.A, args.B, reply)
```

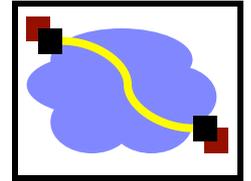
RPC Goals



- Ease of programming
- Hide complexity
- Automates task of implementing distributed computation
- Familiar model for programmers (just make a function call)

Historical note: Seems obvious in retrospect, but RPC was only invented in the '80s. See Birrell & Nelson, "Implementing Remote Procedure Call" ... or Bruce Nelson, Ph.D. Thesis

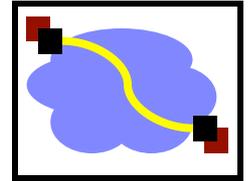
Remote procedure call



- A remote procedure call makes a call to a remote service look like a local call
 - RPC makes transparent whether server is local or remote
 - RPC allows applications to become distributed transparently
 - RPC makes architecture of remote machine transparent

Emphasis on **transparency**

Remote procedure call



- A remote procedure call makes a call to a remote service look like a local call
 - RPC makes transparent whether server is local or remote
 - RPC allows applications to become distributed transparently
 - RPC makes architecture of remote machine transparent

Emphasis on **transparency**

What are some problems with this transparency?