CSCI-1680
APIs

Nick DeMarinis

Based partly on lecture notes by Rodrigo Fonseca, Scott Shenker and John Jannotti
TCP officially due tonight (Tuesday, Nov 21)
  – Office hours 2-5pm (Me); 5-7pm (Alex); 7-9pm (Rhea)

  – Like with IP: you can continue to make small bugfixes after the deadline
    • OK: Fixing small bugs, README, capture files, code cleanup
    • Not OK: eg. implementing sendfile/recvfile, teardown, submitting untested code

  – Grading meetings: after break
Administrivia

Final project is online
• Group registration form: due tomorrow (11/29) by 5pm EST
• Brief proposal: due Friday 12/1, no late days permitted!!
  – We will review all of these over the weekend

• HW4 (probably last HW): out this week, due next week

• TCP grading: end of this week, early next week
  – Look for email today/tomorrow
Project examples

• Make your own iterative DNS resolver
• Build a simple HTTP server
• Make your own web API for something
• Implement Snowcast, etc. using RPCs (more next week)
• Extend your IP/TCP in some way…

These are only a few ideas!
HTTP request: a way to fetch (GET) or send (POST) some object
• Doesn’t need to be a web page
• Doesn’t need to be from a browser

⇒ Generic way to ask the server to do something => an API over the network!
How do programs communicate?

Need a protocol! We’ve seen lots of examples….
IP, TCP, ICMP, RIP, OSPF, BGP, DNS, HTTP, Snowcast …
How do programs communicate?

Need a protocol! We’ve seen lots of examples…. IP, TCP, ICMP, RIP, OSPF, BGP, DNS, HTTP, Snowcast …

⇒ What do protocols require?
Requirements for protocols

Data representation (headers, packet formats)

<table>
<thead>
<tr>
<th>Field</th>
<th>Length (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>4</td>
</tr>
<tr>
<td>IHL</td>
<td>4</td>
</tr>
<tr>
<td>Tos</td>
<td>8</td>
</tr>
<tr>
<td>Identification</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>13</td>
</tr>
<tr>
<td>Fragment Offset</td>
<td>16</td>
</tr>
<tr>
<td>Total Length</td>
<td>16</td>
</tr>
<tr>
<td>Protocol</td>
<td>8</td>
</tr>
<tr>
<td>Header checksum</td>
<td>16</td>
</tr>
<tr>
<td>Source address</td>
<td>32</td>
</tr>
<tr>
<td>Destination address</td>
<td>32</td>
</tr>
<tr>
<td>Options</td>
<td>0-40</td>
</tr>
<tr>
<td>Up to Data</td>
<td>65536</td>
</tr>
</tbody>
</table>

Semantics (when to send each message, how to handle errors)

Transmission Control Protocol (TCP)

Abstract

This document specifies the Transmission Control Protocol (TCP). TCP is an important transport-layer protocol in the Internet protocol stack, and it has continuously evolved over decades of use and growth of the Internet. Over this time, a number of changes have been made.

A Standard for the Transmission of IP Datagrams on Avian Carriers

Status of this Memo

This memo describes an experimental method for the encapsulation of IP datagrams in avian carriers. This specification is primarily useful in Metropolitan Area Networks. This is an experimental, not recommended standard. Distribution of this memo is unlimited.

Overview and Rational

Avian carriers can provide high delay, low throughput, and low
Requirements for protocols

Data representation (headers, packet formats)
Requirements for protocols

Data representation (headers, packet formats)

Semantics (when to send each message, how to handle errors)

Must be specific enough to interoperate (support multiple architectures, byte orders, languages, locales …)
When you made a custom protocol...

Client to Server Commands

The client sends the server messages called **commands**. There are two commands the client can send the server, in the following format:

```
Hello:
    uint8 commandType = 0;
    uint16 udpPort;

SetStation:
    uint8 commandType = 1;
    uint16 stationNumber;
```

A **uint8** is an unsigned 8-bit integer; a **uint16** is an unsigned 16-bit integer. Your programs **MUST** use **network byte order**. So, to send a **Hello** command, your client would send exactly three bytes to the server: one for the command type and two for the port.
When you made a custom protocol...

Client to Server Commands

The client sends the server messages called **commands**. There are two commands the client can send the server, in the following format:

**Hello:**

```go
type struct

messageType uint8
udpPort uint16
```

**SetStation:**

```go
type struct

messageType uint8
stationNumber uint16
```

A **uint8** is an unsigned 8-bit integer; a **uint16** is an unsigned 16-bit integer. All programs MUST use network **byte order**. So, to send a message to the server, the client sends exactly three bytes to the server: one for the command type, one for the server port (server programs can use this information to determine which part of the network the client is on), and the last byte for the station number.

Here is an example of a custom protocol for a guessing game:

```go
// Guessing game example (lecture 3!!)
type struct

GuessMessage {
    MessageType uint8
    Number uint16
}

func (m *GuessMessage) Marshal() []byte {
    buf := new(bytes.Buffer)
    err := binary.Write(buf, binary.BigEndian, m.MessageType)
    if err != nil {
        ...
    }
    err = binary.Write(buf, binary.BigEndian, m.Number)
    if err != nil {
        ...
    }
    return buf.Bytes()
}
```
When you made a custom protocol...

Client to Server Commands

The client sends the server messages called **commands**. There are two commands the client can send the server, in the following format:

- **Hello**:
  - `uint8 commandType = 0;`
  - `uint16 udpPort;`
- **SetStation**:
  - `uint8 commandType = 1;`
  - `uint16 stationNumber;`

A `uint8` is an unsigned 8-bit integer; a `uint16` is an unsigned 16-bit integer. These are network byte order. So, to send commands, you must send exactly three bytes to the server: one for the command type, one for the UDP port, and one for the station number.

// Guessing game example (lecture 3!!)

type struct GuessMessage {
    MessageType uint8
    Number uint16
}

func (m *GuessMessage) Marshal() []byte {
    buf := new(bytes.Buffer)
    err := binary.Write(buf, binary.BigEndian, m.MessageType)
    if err != nil {
        ... 
    }
    err = binary.Write(buf, binary.BigEndian, m.Number)
    if err != nil {
        ... 
    }
    return buf.Bytes()
}

All the protocols you’ve made so far (+IP, TCP, RIP, ...): manually packing bytes into buffers
All the protocols you’ve been writing so far: manually loading bytes into buffers

This is useful for learning:
• How protocols work under the hood
• How fundamental Internet protocols actually work

But if your job is to build applications, is this what you should be doing?

Almost certainly not.
How SHOULD you write a protocol outside this class?

And why?
How SHOULD you write a protocol outside this class?

And why?

* At least, how to start thinking about it
Typical application goal: make an API for something

What you have: some servers/services that live somewhere in the cloud
Typical application goal: make an API for something

What you have: some servers/services that live somewhere in the cloud
=> Might be distributed, might not

Want: end-user to be able to use your app
- Read data
- Write/upload data

Diagram:
- Client
  - do_things()
  - Response/error
- Your App
Challenges/Requirements
- Heterogeneous devices (desktop/mobile, different OSes)
- Application will change
- Number of user devices will scale
- Number of services/services will scale too!
Would like to have a generic API for interacting with application services

=> Flexible to changes
=> Easy to scale
=>
Why doesn’t this work?

Client to Server Commands

The client sends the server messages called **commands**. There are two commands the client can send the server, in the following format:

```
Hello:
  uint8 commandType = 0;
  uint16 udpPort;

SetStation:
  uint8 commandType = 1;
  uint16 stationNumber;
```

A `uint8` is an unsigned 8-bit integer; a `uint16` is an unsigned 16-bit integer. So, to send a `SetStation` command, you must send exactly three bytes to the server: one for the command type and two for the station number.

---

// Guessing game example (lecture 3!!)

type struct GuessMessage {
    MessageType uint8
    Number uint16
}

func (m *GuessMessage) Marshal() []byte {
    buf := new(bytes.Buffer)
    err := binary.Write(buf, binary.BigEndian, m.MessageType)
    if err != nil {
        ...}
    err = binary.Write(buf, binary.BigEndian, m.Number)
    if err != nil {
        ...}
    return buf.Bytes()```
Usually, build on existing tools that can define the API for you
Usually, build on existing tools that can define the API for you

=> Creates **endpoints** where you write code to perform actions

=> Don’t need to worry about serializing/deserializing messages

=> Build on existing protocols to handle scaling
   (eg. HTTP proxies, load balancing, caching, etc.)
Concepts: endpoints
HTTP APIs

- Endpoints at various URLs
- Usually: Request data with GET, upload with POST
- Client authenticates/passes inputs data with headers, cookies
- Response normally JSON, XML, or other self-describing format

GET /component/do_some_action

200 OK + (Data)
An example
curl -X GET 'https://www.gradescope.com/courses/567871/memberships.csv'
-H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0) Gecko/20100101 Firefox/118.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8'
-H 'Accept-Language: en-US,en;q=0.5'
-H 'Accept-Encoding: gzip, deflate, br'
-H 'Referer: https://www.gradescope.com/courses/567871/memberships'
-H 'DNT: 1'
-H 'Connection: keep-alive'
-H 'Cookie: remember_me=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX; _stripe_mid=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXY
-H 'Upgrade-Insecure-Requests: 1'
-H 'Sec-Fetch-Dest: document'
-H 'Sec-Fetch-Mode: navigate'
-H 'Sec-Fetch-Site: same-origin'
-H 'Sec-Fetch-User: ?1'
$ curl https://api.github.com/users/ndemarinis
{
    "login": "ndemarinis",
    "id": 1191319,
    "node_id": "MDQ6VXNlcjExOTEzMTk=[",
    "gravatar_id": "",
    "url": "https://api.github.com/users/ndemarinis",
    "type": "User",
    "site_admin": false,
    "name": "Nick DeMarinis",
    "blog": "https://vty.sh",
    "twitter_username": null,
    "public_repos": 10,
    ...
}
Why is this useful?

• HTTP is ubiquitous
• Lots of existing tools to scale HTTP
  – Cookies etc. for user authentication
  – Proxies/load balancers

Why use JSON/etc vs. a binary encoding?
HTTP Example

> telnet www.cs.brown.edu 80
Trying 128.148.32.110...
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 24 Mar 2011 12:58:46 GMT
Server: Apache/2.2.9 (Debian) mod_ssl/2.2.9 OpenSSL/0.9.8g
ETag: "840a88b-236c-49f3992853bc0"
Accept-Ranges: bytes
Content-Length: 9068
Vary: Accept-Encoding
Connection: close
Content-Type: text/html

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

...
What if you need more flexibility?
Generic view: Remote Procedure Call (RPC)

- Procedure calls are a well understood mechanism
  - Transfer control and data on a single computer
- Idea: make distributed programming look the same
  - Have servers export interfaces that are accessible through local APIs
  - Perform the illusion behind the scenes
- 2 Major Components
  - Protocol to manage messages sent between client and server
  - Language and compiler support
    - Packing, unpacking, calling function, returning value
Stub Functions

• Local stub functions at client and server give appearance of a local function call
  • client stub
    – marshalls parameters -> sends to server -> waits
    – unmarshalls results -> returns to client
  • server stub
    – creates socket/ports and accepts connections
    – receives message from client stub -> unmarshalls parameters -> calls server function
    – marshalls results -> sends results to client stub
The network between the calling process and the called process has much more complex properties than the backplane of a computer. For example, it is likely to limit message sizes and has a tendency to lose and reorder messages. The computers on which the calling and called processes run may have significantly different architectures and data representation formats. Thus, a complete RPC mechanism actually involves two major components:

1. A protocol that manages the messages sent between the client and the server processes and that deals with the potentially undesirable properties of the underlying network.

2. Programming language and compiler support to package the arguments into a request message on the client machine and then to translate this message back into the arguments on the server machine, and likewise with the return value (this piece of the RPC mechanism is usually called a stub compiler).

Figure 5.12 schematically depicts what happens when a client invokes a remote procedure. First, the client calls a local stub for the procedure, passing it the arguments required by the procedure. This stub hides the fact that the procedure is remote by packaging the arguments into an RPC protocol message, sending it to the server, and then translating the return value back into the arguments on the client machine.
Some examples

- gRPC
- Apache Thrift
- JSON-RPC
- XML-RPC, SOAP
- ...
Design questions
Describing data
Example: gRPC

• IDL-based, defined by Google
  – Protocol Buffers as IDL

• User specifies services, calls
  – Single and streaming calls
  – Support for timeouts, cancellations, etc

• Transport: based on HTTP/2
gRPC

• Generates stubs in many languages
  – C/C++, C#, Node.js, PHP, Ruby, Python, Go, Java
  – These are interoperable
• Transport is http/2
Protocol Buffers

- Defined by Google, released to the public
  - Widely used internally and externally
  - Supports common types, service definitions
  - Natively generates C++/Java/Python/Go code
    - Over 20 other supported by third parties
  - Efficient binary encoding, readable text encoding
- Performance
  - 3 to 10 times smaller than XML
  - 20 to 100 times faster to process
Protocol Buffers Example (for a file)

```cpp
message Student {
  required String name = 1;
  required int32 credits = 2;
}

Student s;
  s.set_name("Jane");
  s.set_credits(20);
  fstream output("students.txt", ios::out | ios::binary);
  s.SerializeToOstream(output);

Student s;
  fstream input("students.txt", ios::in | ios::binary);
  s.ParseFromIstream();
```
protobuf: Binary Encoding

• Variable-length integers
  – 7 bits out of 8 to encode integers
  – Msb: more bits to come
  – Multi-byte integers: least significant group first

• Signed integers: zig-zag encoding, then varint
  – 0:0, -1:1, 1:2, -2:3, 2:4, ...
  – Advantage: smaller when encoded with varint

• General:
  – Field number, field type (tag), value

• Strings:
  – Varint length, unicode representation
Apache Thrift

- Originally developed by Facebook
- Used heavily internally
- Supports (at least): C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, and Ocaml
- Types: basic types, list, set, map, exceptions
- Versioning support
- Many encodings (protocols) supported
  - Efficient binary, json encodings
Conclusions

• Unless you *really* want to optimize your protocol for performance, use an IDL

• Parsing code is easy to get (slightly) wrong, hard to make fast—only want to do this once!

• Which one should you use?
Which data types?

• Basic types
  – Integers, floating point, characters
  – Some issues: endianness (ntohs, htons), character encoding, IEEE 754

• Flat types
  – Strings, structures, arrays
  – Some issues: packing of structures, order, variable length

• Complex types
  – Pointers! Must flatten, or serialize data structures
At the next level are flat types—structures and arrays. While flat types might at first not appear to complicate argument marshalling, the reality is that they do. The problem is that the compilers used to compile application programs sometimes insert padding between the fields that make up the structure so as to align these fields on word boundaries. The marshalling system typically packs structures so that they contain no padding.

At the highest level, the marshalling system might have to deal with complex types—those types that are built using pointers. That is, the data structure that one program wants to send to another might not be contained in a single structure, but might instead involve pointers from one structure to another. A tree is a good example of a complex type that involves pointers. Clearly, the data encoder must prepare the data structure for transmission over the network because pointers are implemented by memory addresses, and just because a structure lives at a certain memory address on one machine does not mean it will live at the same address on another machine. In other words, the marshalling system must serialize (flatten) complex data structures.

▶ In summary, depending on how complicated the type system is, the task of argument marshalling usually involves converting the base types, packing the structures, and linearizing the complex data structures, all to form a contiguous message that can be transmitted over the network. Figure 7.3 illustrates this task.

Conversion Strategy
Once the type system is established, the next issue is what conversion strategy the argument marshaller will use. There are two general options: canonical intermediate form and receiver-makes-right. We consider each, in turn.
Problem

• Two programs want to communicate: must define the protocol
  – We have seen many of these, across all layers
  – E.g., Snowcast packet formats, protocol headers

• Key Problems
  – Semantics of the communication
    • APIs, how to cope with failure
  – Data Representation
  – Scope: should the scheme work across
    • Architectures
    • Languages
    • Compilers...?
Data Schema

• How to parse the encoded data?
• Two Extremes:
  – Self-describing data: tags
    • Additional information added to message to help in decoding
    • Examples: field name, type, length
  – Implicit: the code at both ends “knows” how to decode the message
    • E.g., your Snowcast implementation
    • Interoperability depends on well defined protocol specification!
    • very difficult to change
Presentation Formatting

• How to represent data?
• Several questions:
  – Which data types do you want to support?
    • Base types, Flat types, Complex types
  – How to encode data into the wire
  – How to decode the data?
    • Self-describing (tags, type-length-value)
    • Implicit description (the ends know)
• Several answers:
  – Many frameworks do these things automatically
Stub Generation

- 2 Main ideas:
  - Introspection-based
    - E.g., Java RMI
  - Independent specification: IDL
    - IDL – Interface Description Language
      - describes an interface in a language neutral way
    - Separates logical description of data from
      - Dispatching code
      - Marshalling/unmarshalling code
      - Data wire format
Web browser

DNS
domain: example.com?

Webserver example.com

DNS resolves to an IP address

Web browser sends a GET request

GET /page.html

200 OK + (Content of page.html)

Web browser receives and renders the HTML

Welcome!

Server returns response (in this case, with HTML)