Protocols and Layering

- We need abstractions to handle all this system complexity

  A protocol is an agreement dictating the form and function of data exchanged between parties to effect communication

- Two parts:
  - Syntax: format -- where the bits go
  - Semantics: meaning -- what the words mean, what to do with them

- Examples:
  - IP, the Internet protocol; TCP and HTTP, for the Web
  - You can make up your own
Layering and Protocol Stacks

- Layering is how we combine protocols
  - Higher level protocols build on services provided by lower levels
  - Peer layers communicate with each other

You -> Yahoo!
Layering Mechanics

- Encapsulation and de(en)capsulation

Analogy: A packet is an envelope.
- What’s written on the outside is the header
- What’s contained on the inside is the payload
- The payload may, itself, be another envelope
- Each layer understands (and acts on) the writing on the outside, but doesn’t understand what it contains – just delivers it.
Example – Layering at work
A Packet on the Wire

- Starts looking like an onion!

- This isn’t entirely accurate
  - ignores segmentation and reassembly, Ethernet trailers, etc.

- But you can see that:
  - layering adds overhead
  - one protocol’s header is another protocol’s data
### What’s Inside a Packet (detailed view)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet Header:</td>
<td>FROM=00:30:65:0a:ea:62, TO=00:30:64:9a:11:22, SIZE=200,…</td>
</tr>
<tr>
<td>IP Header:</td>
<td>FROM=128.95.1.32, TO=28.2.5.1, SIZE=200-SIZEOF(Ehdr)</td>
</tr>
<tr>
<td>TCP Header:</td>
<td>FROM=Port 5000, TO=Port 80, Byte#=23, SIZE=200-SIZEOF(Ehdr)-SIZEOF(IPHdr)</td>
</tr>
<tr>
<td>HTTP Hdr:</td>
<td>HTTP v.1.0, Internet Explorer v5.1,…</td>
</tr>
<tr>
<td>Good Stuff:</td>
<td>GET <a href="http://www.google.com">http://www.google.com</a></td>
</tr>
</tbody>
</table>
More Layering Mechanics

- Multiplexing and demultiplexing in a protocol graph

Diagram:
- SMTP
- HTTP
- TCP
- UDP
- IP
- ARP
- Ethernet

Demux key:
- TCP port number
- IP protocol field
- 802.2 identifier
# Internet Protocol Framework

<table>
<thead>
<tr>
<th>Model</th>
<th>Protocols</th>
<th>The “narrow waist”</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Many (HTTP,SMTP)</td>
<td>email, WWW, phone…</td>
</tr>
<tr>
<td>Transport</td>
<td>TCP / UDP</td>
<td>SMTP, HTTP, RTP…</td>
</tr>
<tr>
<td>Network</td>
<td>IP</td>
<td>TCP, UDP…</td>
</tr>
<tr>
<td>Link</td>
<td>Many (Ethernet, …)</td>
<td>ethernet, PPP…</td>
</tr>
</tbody>
</table>

- **Model**:
  - Application
  - Transport
  - Network
  - Link

- **Protocols**:
  - Many (HTTP, SMTP)
  - TCP / UDP
  - IP
  - Many (Ethernet, …)

- **The “narrow waist”**:
  - email, WWW, phone…
  - SMTP, HTTP, RTP…
  - TCP, UDP…
  - ethernet, PPP…
  - CSMA async, sonet…
  - copper, fiber, radio…
OSI “Seven Layer” Reference Model

Their functions:
- Up to the application
- Encode/decode messages
- Manage connections
- Reliability, congestion control
- Routing
- Framing, multiple access
- Symbol coding, modulation
Protocol Standards

- Different functions require different protocols
- A “standard” protocol is one that has been carefully specified so that different implementations can interoperate.
  - Standardized: screws, batteries
  - Not standardized: Appliances, furniture
- Thus there are many protocol standards
  - E.g., IP, TCP, UDP, HTTP, DNS, FTP, SMTP, NNTP, ARP, Ethernet/802.3, 802.11, RIP, OSPF, 802.1D, NFS, ICMP, IGMP, DVMRP, IPSEC, PIM-SM, BGP, ...
- Organizations: IETF, IEEE, ITU
- IETF (www.ietf.org) specifies Internet-related protocols
  - RFCs (Requests for Comments)
  - “We reject kings, presidents and voting. We believe in rough consensus and running code.” – Dave Clark.
What goes in a standard?

- Is it, e.g., for 802.11:
  - A) The messages exchanged and their meaning
  - B) The services that the higher layer can invoke
  - C) What you need to know to implement it well
  - D) All of the above.
Key Design Question
What functionality goes in which layer?

- The “End to End Argument” (Reed, Saltzer, Clark, 1984):

  Functionality should be implemented at a lower layer only if it can be correctly and completely implemented. (Sometimes an incomplete implementation can be useful as a performance optimization.)

- Example: reliability needs to be done by the endpoints for correctness; add it over individual links just to help performance

- Tends to push functions to the endpoints, which has aided the extensibility of the Internet.
Pros of the End-to-End principle

- Fosters innovation by anyone without being a “big player”
  - The middle of the network is (historically) hard to change

- Don’t impose functionality on applications that don’t need it
  - E.g., VoIP doesn’t *want* reliability inside the network

- Favors simplicity. Complex middle = complex interface.
  - Can improve robustness
  - Plus many complicated things fail ..
Cons of the End-to-End principle

- Doesn’t work well with security, e.g., firewalls
  - Articulated in a time when we trusted each other 😊

- End-points are also hard to change *en masse*
  - New versions of TCP can’t “just be deployed”
  - Leads to middleboxes in the Internet

- Loss of efficiency or duplication of effort if taken literally
  - Just relax a bit ...