TCP: Transmission Control Protocol

- TCP, a Transport Layer protocol, provides reliability:
  - Takes care of resending lost, damaged, delayed packets.
  - Sorts packets into the original order they were sent in.

- TCP also provides virtual connections:
  - The underlying IP (Network) Layer is connectionless, but is used to realize continuous connections.
  - These connections give other internet technologies the means to exchange finite, ordered bit streams.

- TCP supports having many such applications, by providing each with a port number, e.g.:

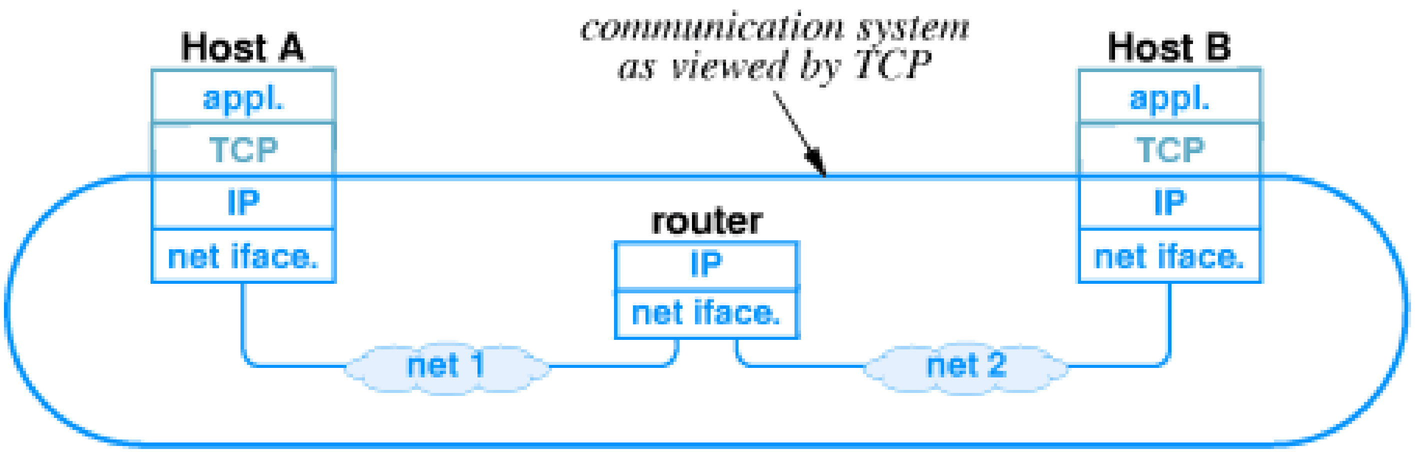
  21 = *FTP server (file transfer)*

  25 = *SMTP server (e-mail)*

  etc. *(range: 0-65535).*

TCP uses the abstraction provided by IP

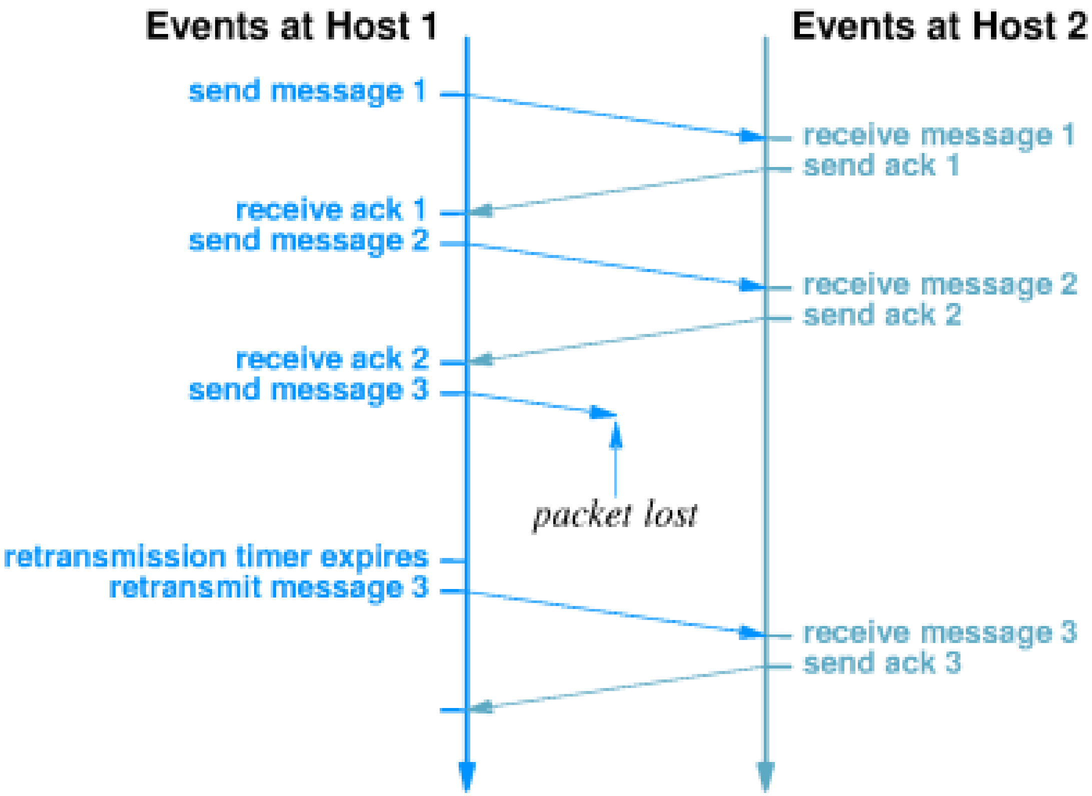
- TCP uses IP’s services to abstract from underlying networks:
  - It sees just a single internetwork.
  - It knows nor cares about the underlying network infrastructure:



TCP: implementing reliability

- TCP uses *acknowledgments* to track the delivery of packets.

  When a packet has not been acknowledged in due time, TCP resends. *Simplified:*



← In TCP, packets are called *segments*.

Q: *Why would this term make sense?*

TCP: implementing virtual connections

- On a source host, TCP:
  - Initiates a connection.
  - Then splits data (from the Application Layer) into smaller packets (a.k.a. *datagrams* or *segments*).
  - Then adds its own header to each segment.
  - Then passes the segments to the Internet Layer (IP).
  - Retransmits segments if receipt not acknowledged in time.
  - Ends the connection when done.

- On a target host, TCP:
  - Sends an acknowledgment for each received segment.
  - Assembles received segments in the correct order.
  - Rebuilds the bit stream from the segments.
  - Passes it to the appropriate application on the correct port.

TCP: introduces the client/server model

- Server: an IP address + TCP port *providing* a service.

- Client: an IP address + TCP port *using* a service.

- Examples:
  - A *mail client* (e.g. Outlook) retrieving e-mail from a *mail server* (e.g. Exchange).
  - An *FTP client* sending files to an *FTP server*.
  - A *telnet client* connecting to a *telnet server*.
  - Etc.!

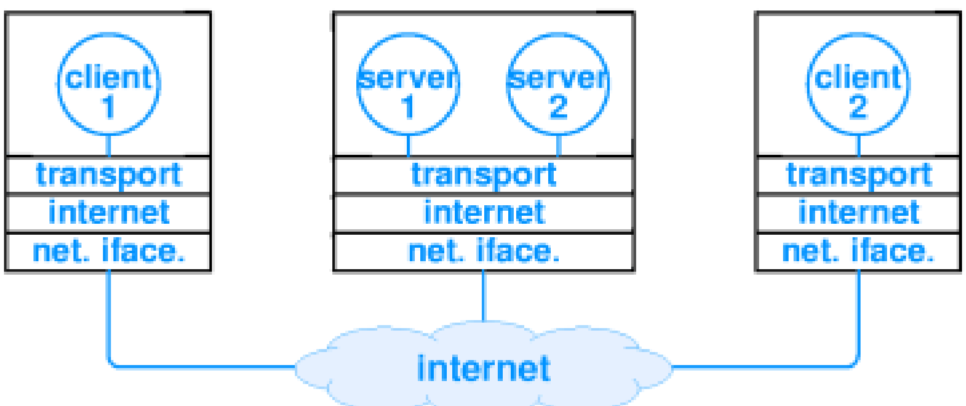
- *Servers are often passive:* wait for requests from clients to come in.

TCP: introduces the client/server model

- A machine can run multiple servers (*applications, programs*) with each performing I/O via a different TCP port.

- Each server may use a different Application Layer protocol (e.g. telnet, FTP, SMTP).

- Multiple clients may be contacting multiple servers on one machine:



UDP: User Datagram Protocol

__Ap__plication layer

__Tr__ansport layer

__Ne__twork layer

__Da__ta link layer

__Physical__ layer

Q: Where do you think UDP is located? ↑

A: Like TCP, UDP is a *Transport Layer* protocol.

UDP: comparison with TCP

- Similarities with TCP:
  - Also runs on top of IP (naturally).
  - Also is a very widely used Internet protocol.
  - Also provides port numbers to applications.

UDP: comparison with TCP

- Differences with TCP:
  - Is connectionless: does not provide virtual connections.
  - Is unreliable:
    - packets may get lost or seriously delayed;
    - packets are not sorted into their original order.
  - ...But, is generally faster than TCP.

UDP: why is it faster?

- Providing reliable connections (e.g. TCP) has a **cost**.

- Cost is in terms of time/computations:
  - *Handshake* exchanges to set up and terminate a connection.
  - Acknowledging, waiting for, and re-transmitting segments.
  - Processing of the complex TCP headers.

UDP: why is it faster?

- Consider TCP headers:

TCP Header

<table>
<tr><th>Offsets</th><th>Octet</th><th colspan="8">0</th><th colspan="8">1</th><th colspan="8">2</th><th colspan="8">3</th></tr>
<tr><th>Octet</th><th>Bit</th><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th>16</th><th>17</th><th>18</th><th>19</th><th>20</th><th>21</th><th>22</th><th>23</th><th>24</th><th>25</th><th>26</th><th>27</th><th>28</th><th>29</th><th>30</th><th>31</th></tr>
<tr><td>0</td><td>0</td><td colspan="16">Source port</td><td colspan="16">Destination port</td></tr>
<tr><td>4</td><td>32</td><td colspan="32">Sequence number</td></tr>
<tr><td>8</td><td>64</td><td colspan="32">Acknowledgment number (if ACK set)</td></tr>
<tr><td>12</td><td>96</td><td colspan="4">Data offset</td><td colspan="3">Reserved
0 0 0</td><td>NS</td><td>CWR</td><td>ECE</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="16">Window Size</td></tr>
<tr><td>16</td><td>128</td><td colspan="16">Checksum</td><td colspan="16">Urgent pointer (if URG set)</td></tr>
<tr><td>20
...</td><td>160
...</td><td colspan="32">Options (if Data Offset > 5,padded at end with "0" bytes if necessary)
...</td></tr>
</table>

↑ Significant computational processing required during communication.

UDP: why is it faster?

- Consider UDP datagrams:

offset (bits)	0 – 15	16 – 31
0	Source Port Number	Destination Port Number
32	Length	Checksum
64+	Data	

The UDP header consists of 4 fields, each of which is 2 bytes (16 bits).[1] The use of two of those is optional in IPv4 (pink background in table). In IPv6 only the source port is optional (see below).

↑ Simple format, *very* little overhead.

⇒ Is *cheap* in terms of computational processing required during communication.

UDP: when is it used?

- Mailservers, file transfers, etc.: *need to be reliable!*

⇒ **Q:** So who uses UDP? Why use an unreliable protocol, when there is TCP?

⇒ **A:** Applications that:
  · Need to be fast.
  · Have single packet communications (*no ordering required*).
  · Can handle lost packets well.

- Some examples:
  · voice over IP (VoIP)
  · real-time multiplayer games
  · streaming media (e.g. music, video)
  · Open Sound Control (OSC).

DNS: Name resolution

- **Problem:** The – many – IP addresses...
  ...are (still) not easy to remember – for us humans.
  ...give you no clue about what is provided via them.

- **Solution:** Name resolution:
  - Host owner: picks a *name* that is informative & easy to remember.
  - ...and submits it to a "central list" of (*name, IP address*) pairs.
  - Other hosts can now use the name & look up the address in the list.

↑ *Early version:* a single name server implements (access to) the list.
(+) Hosts/users remember *just one* IP address: of the name server.
(–) Does not scale!
- Lookup time: *processing a list with millions of entries*.
- Traffic load: *processing millions of requests per second*.
- Redundancy: *What if the server crashes?*

DNS: Name resolution

- **Current solution:** DNS, which is:
  - An Application Layer protocol for name resolution.
  - A *distributed* name server architecture. There are:

    ...ordinary name servers:
    - provide direct name resolution
    - but are only responsible for a *part* of the list.

    ...and root name servers:
    - do not provide direct name resolution
    - instead maintain a *list of name servers*
    - response to requests: IP address of *name server* that knows.

  (+) Hosts/users remember *just* the IP addresses of root servers.

  (+) Does scale!
  - Lookup time: *the server types now each maintain shorter lists* ↑
  - Traffic load: *request/reply traffic now is distributed across servers*.
  - Redundancy: *multiple root servers* and *full list not at one place*.

DNS: Name resolution

- **(!) However – key point to actually make this work:**
  - A root name server has to work out which other server to refer you to *simply by looking at the name you requested*.

  ⇒ We need to somehow distribute the host names across the name servers in a way that is *stored inside the names themselves*...

- Luckily, we can look at everyday life:
  - Consider *first* and *family* names, e.g. Alice Combs, Bob Combs.
  - "Let's put all the Combses in a single name server."
  - For Internet hosts, the notation becomes alice.com, bob.com.

  ⇒ **Problem solved!**
  - ".com" is a "domain" ⇒ and DNS is the Domain Name System.
  - Domains expand *hierarchically:* e.g. advice.bob.com, gossip.bob.com

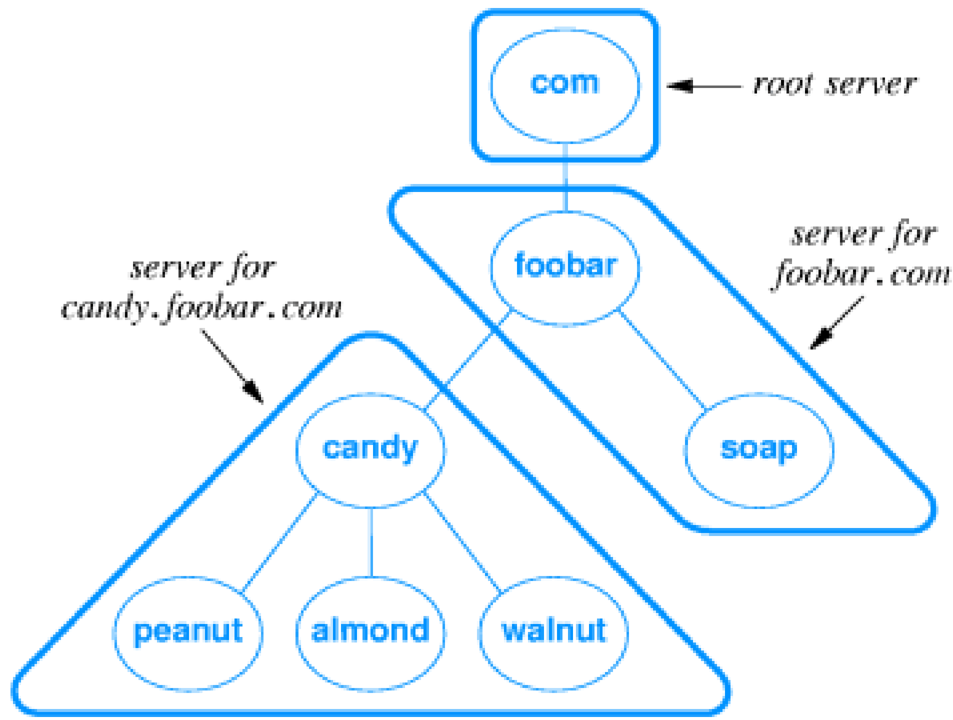
DNS: a few concrete examples

- Suppose you want to start using the IP address associated with walnut.candy.foobar.com.

- Your local DNS server does not hold the requested *domain name*, and so the request is forwarded to a DNS *root server*.

- It then returns which DNS *name server* to contact next for the requested domain name...

  ...and this repeats. →



- *To also study & understand:* the everyday DNS request/reply scenario discussed in the required reading material.

What's next?

- We now know why Internet host names have *this.dotted.structure*:
  - Because of DNS.
  - Had to strike a balance in human/machine “readability”.

⇒ We are now at the end of the *internetworking* arc. We can:
  - digitally connect to any machine on a worldwide internet (...*thanks to IP*),
  - using just its name (...*thanks to DNS*),
  - and reliably send arbitrary bit streams to and from it (...*thanks to TCP*).

Next: in the early 1990s, the *above technologies* were used by *someone* to come up with a *project*.