CSCI-GA.2250-001

Operating Systems

Networking

Hubertus Franke
frankeh@cs.nyu.edu
Source: Ganesh Sittampalam

NYU
TCP/IP protocol family

- IP : Internet Protocol
  - UDP : User Datagram Protocol
    - RTP, traceroute
  - TCP : Transmission Control Protocol
    - HTTP, FTP, ssh
What is an internet?

- A set of interconnected networks
- The Internet is the most famous example

- Networks can be completely different
  - Ethernet, ATM, modem, …
  - (TCP/)IP is what links them
What is an internet? (cont)

• *Routers* (nodes) are devices on multiple networks that pass traffic between them
• Individual networks pass traffic from one router or endpoint to another
• TCP/IP hides the details as much as possible
OSI: Open Systems Interconnect

**OSI and Protocol Stack**

<table>
<thead>
<tr>
<th>OSI Model</th>
<th>TCP/IP Hierarchy</th>
<th>Protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7th</strong> Application Layer</td>
<td>Application Layer</td>
<td>HTTP</td>
</tr>
<tr>
<td><strong>6th</strong> Presentation Layer</td>
<td></td>
<td>SMTP</td>
</tr>
<tr>
<td><strong>5th</strong> Session Layer</td>
<td></td>
<td>POP3</td>
</tr>
<tr>
<td><strong>4th</strong> Transport Layer</td>
<td>Transport Layer</td>
<td>FTP</td>
</tr>
<tr>
<td><strong>3rd</strong> Network Layer</td>
<td>Network Layer</td>
<td>TCP</td>
</tr>
<tr>
<td><strong>2nd</strong> Link Layer</td>
<td>Link Layer</td>
<td>UDP</td>
</tr>
<tr>
<td><strong>1st</strong> Physical Layer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Link Layer**: includes device driver and network interface card
**Network Layer**: handles the movement of packets, i.e. Routing
**Transport Layer**: provides a reliable flow of data between two hosts
**Application Layer**: handles the details of the particular application
Packet Encapsulation

- The data is sent down the protocol stack
- Each layer adds to the data by prepending headers
IP

- Responsible for end to end transmission
- Sends data in individual packets
- Maximum size of packet is determined by the networks
  - Fragmented if too large
- Unreliable
  - Packets might be lost, corrupted, duplicated, delivered out of order
IP addresses

• 4 bytes
  – e.g. 163.1.125.98
  – Each device normally gets one (or more)
  – In theory there are about 4 billion available

• But…
Routing

• How does a device know where to send a packet?
  – All devices need to know what IP addresses are on directly attached networks
  – If the destination is on a local network, send it directly there
Routing (cont)

• If the destination address isn’t local
  – Most non-router devices just send everything to a single local router
  – Routers need to know which network corresponds to each possible IP address
Routing
Allocation of addresses

• Controlled centrally by ICANN
  – Fairly strict rules on further delegation to avoid wastage
    • Have to demonstrate actual need for them
• Organizations that got in early have bigger allocations than they really need
IP packets

- Source and destination addresses
- Protocol number
  - 1 = ICMP, 6 = TCP, 17 = UDP
- Various options
  - e.g. to control fragmentation
- Time to live (TTL)
  - Prevent routing loops
We only looked at the IP addresses, TTL and protocol #
IP Routing

- Routing Table
  - Destination IP address
  - IP address of a next-hop router
  - Flags
  - Network interface specification
UDP

- Thin layer on top of IP
- Adds packet length + checksum
  - Guard against corrupted packets
- Also source and destination *ports*
  - Ports are used to associate a packet with a specific application at each end
- Still unreliable:
  - Duplication, loss, out-of-order-ness possible
# UDP datagram

![UDP datagram diagram]

<table>
<thead>
<tr>
<th>Field</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>16-bit port number identifying originating application</td>
</tr>
<tr>
<td>Destination Port</td>
<td>16-bit port number identifying destination application</td>
</tr>
<tr>
<td>Length</td>
<td>Length of UDP datagram (UDP header + data)</td>
</tr>
<tr>
<td>Checksum</td>
<td>Checksum of IP pseudo header, UDP header, and data</td>
</tr>
</tbody>
</table>
Typical applications of UDP

– Where packet loss etc is better handled by the application than the network stack
– Where the overhead of setting up a connection isn’t wanted

• VOIP
• NFS – Network File System
• Most games
TCP

• Reliable, full-duplex, connection-oriented, stream delivery
  – Interface presented to the application doesn’t require data in individual packets
  – Data is guaranteed to arrive, and in the correct order without duplications
    • Or the connection will be dropped
  – Imposes significant overheads
Applications of TCP

- Most things!
  - HTTP, FTP, ...

- Saves the application a lot of work, so used unless there’s a good reason not to
TCP implementation

• Connections are established using a *three-way handshake*
• Data is divided up into packets by the operating system
• Packets are numbered, and received packets are acknowledged
• Connections are explicitly closed
  – (or may abnormally terminate)
TCP Packets

- Source + destination ports
- Sequence number (used to order packets)
- Acknowledgement number (used to verify packets are received)
TCP Segment

<table>
<thead>
<tr>
<th>Field</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Identifies originating application</td>
</tr>
<tr>
<td>Destination Port</td>
<td>Identifies destination application</td>
</tr>
<tr>
<td>Sequence Number</td>
<td>Sequence number of first octet in the segment</td>
</tr>
<tr>
<td>Acknowledgment #</td>
<td>Sequence number of the next expected octet (if ACK flag set)</td>
</tr>
<tr>
<td>Len</td>
<td>Length of TCP header in 4 octet units</td>
</tr>
<tr>
<td>Flags</td>
<td>TCP flags: SYN, FIN, RST, PSH, ACK, URG</td>
</tr>
<tr>
<td>Window</td>
<td>Number of octets from ACK that sender will accept</td>
</tr>
<tr>
<td>Checksum</td>
<td>Checksum of IP pseudo-header + TCP header + data</td>
</tr>
<tr>
<td>Urgent Pointer</td>
<td>Pointer to end of “urgent data”</td>
</tr>
<tr>
<td>Options</td>
<td>Special TCP options such as MSS and Window Scale</td>
</tr>
</tbody>
</table>

You just need to know port numbers, seq and ack are added
TCP: Data transfer

Client

- Send Packet 1
- Start Timer

- ACK would normally arrive at this time

- Time Expires

- Retransmit Packet 1
- Start Timer

- Receive ACK 1
- Cancel Timer

Host

- Packet should arrive
- ACK should be sent

- Receive Packet 1
- Send AXK 1

Timer

- Timer

Packet Lost
IPv6

• 128 bit addresses
  – Make it feasible to be very wasteful with address allocations

• Lots of other new features
  – Built-in autoconfiguration, security options, …

• Not really in production use yet
General Structure
Interaction IP Stack / NIC
Linux Tx/Rx Ring handling
TCP/IP Details

- The sliding window serves several purposes:
  - (1) it guarantees the reliable delivery of data
  - (2) it ensures that the data is delivered in order,
  - (3) it enforces flow control between the sender and the receiver.
Flow Control

• Max Send and Receive Buffer sizes

• In order delivery to the consumer

• Acknowledgement of reception

• Retransmit when ack is not received in RTT (RoundTripTime) setting
Congestion Control

• Slow Start
  - Start with 1 congestion window and then doubling it

• Fast Retransmit
  - When out of order packet is received immediately ACK

• Fast Recovery
Device Driver Details

Receive

Application

OS

socket Rx

netif_receive_skb

NET RX Softirq

NIC Rx

Switch

Port B Rx

send

pause/resume queue

Transmit

Application

OS

socket Tx

netif_send_skb

NET Driver

NIC Tx

Switch

Port B Tx

send

pause/resume queue
**Slab-Cache**

- The primary motivation for slab allocation:
  - initialization and destruction of kernel data objects can actually outweigh the cost of allocating memory for them.
  - As object creation and deletion are widely employed by the kernel, mitigating overhead costs of initialization can result in significant performance gains.
  - "object caching" was therefore introduced in order to avoid the invocation of functions used to initialize object state.

- Group same dynamically allocated objects under one "allocator" object

- E.g. skb_buff_alloc()
Some useful general “stuff”

- **General implementation**

  ```
  struct kmem_cache
  size
  order
  name
  ...
  cpu_slab
  ```

- **Other features:**
  - Coloring → Cache utilization
Linux Example of slab caches

- `#> slabtop`