Computer Networks

Exercise Session 09

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

January 12, 2024

General Schedule

All exercises will follow this general schedule

- Identify potential understanding problems
  $\rightarrow$ Ask your questions
  $\rightarrow$ Recap of the lecture
- Address the understanding problems
  $\rightarrow$ Answer your questions
  $\rightarrow$ Repeat certain topics
- Walk through the exercises/solutions $\rightarrow$ Some hints and guidance
  $\rightarrow$ Work time or presentation of results

Network Layer: Addressing

You have seen ...

- the purpose and format of IPv4 and IPv6 addresses
- the original classes of IPv4 networks, what CIDR and what subnets are
- how to connect private networks to the Internet using NAT
- that IP datagrams can be fragmented if they are too big for a single frame on the data link layer
- why a successor for IPv4 was needed and how IPv6 tackles the challenges

IP Packet Structure

You have seen . . .

- the packet structure of IPv4 packets
- the packet structure of IPv6 packets

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ `/24` → `255.255.255.0`
- `10.1.2.3/24` is different from `10.1.2.3/16`[1]

[1] Both addresses cannot be used in the same network.

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ `/24` → `255.255.255.0`
- `10.1.2.3/24` is different from `10.1.2.3/16`[1]
  - ```
    00001010 00000001 00000010 00000011 AND
    11111111 11111111 11111111 00000000
    ```
    versus
    ```
    00001010 00000001 00000010 00000011 AND
    11111111 11111111 00000000 00000000
    ```

[1] Both addresses cannot be used in the same network.
    ```

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ `/24` → `255.255.255.0`
- `10.1.2.3/24` is different from `10.1.2.3/16`[1]
    - `00001010 00000001 00000010 00000011 AND`
      `11111111 11111111 11111111 00000000`
      versus
      `00001010 00000001 00000010 00000011 AND`
      `11111111 11111111 00000000 00000000`
- Subnet masks are often multiples of eight bits, but not always
  e.g., `10.21.42.83/28`

[1] Both addresses cannot be used in the same network.

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ `/24` → `255.255.255.0`
- `10.1.2.3/24` is different from `10.1.2.3/16`[1]
  - `00001010 00000001 00000010 00000011 AND`
    `11111111 11111111 11111111 00000000`
    versus
    `00001010 00000001 00000010 00000011 AND`
    `11111111 11111111 00000000 00000000`
- Subnet masks are often multiples of eight bits, but not always e.g., `10.21.42.83/28`
  - What's the subnet mask for this address?

[1] Both addresses cannot be used in the same network.

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous

  ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
  - E.g.,
    `ip addr add 192.168.7.3/24 dev wlan0`
  - Reminder: CIDR notations specifies the number of masked bits

    ⇒ `/24` → `255.255.255.0`
- `10.1.2.3/24` is different from `10.1.2.3/16`[1]
  - ```
    00001010 00000001 00000010 00000011 AND
    11111111 11111111 11111111 00000000
    ```
    versus
    ```
    00001010 00000001 00000010 00000011 AND
    11111111 11111111 00000000 00000000
    ```
- Subnet masks are often multiples of eight bits, but not always e.g., `10.21.42.83/28`
  - What's the subnet mask for this address?

  → `/28` → `11111111 11111111 11111111 11110000` → `255.255.255.240`

[1] Both addresses cannot be used in the same network.
    ```

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ /24 → 255.255.255.0
- 10.1.2.3/24 is different from 10.1.2.3/16[1]
  - `00001010 00000001 00000010 00000011 AND`
    `11111111 11111111 11111111 00000000`
    versus
    `00001010 00000001 00000010 00000011 AND`
    `11111111 11111111 00000000 00000000`
- Subnet masks are often multiples of eight bits, but not always e.g., 10.21.42.83/28
  - What's the subnet mask for this address?
  - → /28 → 11111111 11111111 11111111 11110000 → 255.255.255.240
  - What's the network address?

[1] Both addresses cannot be used in the same network.

Exercise 01: IPv4 Addressing

- An IPv4 address without a subnet mask is ambiguous
  - ⇒ Tools like *iputils* (→ `ip`) require the IPv4 address in CIDR notation
    - E.g.,
      `ip addr add 192.168.7.3/24 dev wlan0`
    - Reminder: CIDR notations specifies the number of masked bits
      ⇒ /24 → 255.255.255.0
- 10.1.2.3/24 is different from 10.1.2.3/16[1]
  - 00001010 00000001 00000010 00000011 AND
    11111111 11111111 11111111 00000000
    versus
    00001010 00000001 00000010 00000011 AND
    11111111 11111111 00000000 00000000
- Subnet masks are often multiples of eight bits, but not always
  e.g., 10.21.42.83/28
  - What's the subnet mask for this address?
  - → /28 → 11111111 11111111 11111111 11110000 → 255.255.255.240
  - What's the network address?
  - → 10.21.52.80/28

[1] Both addresses cannot be used in the same network.

Exercise 2: Forwarding Process

- The OS uses a forwarding table (or forwarding information base (FIB)) to select the appropriate interface for sending a packet
- The selector is the destination IP address of the outgoing (or forwarded) packet
- The FIB contains at least two columns:
  - The destination network address
  - The interface
- Optionally it may contain a gateway
- The OS performs a longest prefix match on the selector

Exercise 2: Longest Prefix Matching

- The longest (*best*) matching prefix from the FIB is chosen
- The destination IP address is compared bit by bit with the network addresses in the FIB
- The number of compared bits depends on the prefix length of the FIB entry
- The longest matching prefix is selected and the according interface will be chosen
- There is typically a default entry (`0.0.0.0/0` for IPv4) that always matches

Exercise 2: Inspect the FIB

- On Linux you can query your routing table with *iputils* (→ `ip route show` or simply `ip r`)
- On Windows and Linux you can also use `netstat -r[n]`
- The result may look like this:

```
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         10.51.0.1       0.0.0.0         UG        0 0          0 wlan0
10.2.0.0        0.0.0.0         255.255.255.0   U         0 0          0 enp0s31f6
10.51.0.0       0.0.0.0         255.255.0.0     U         0 0          0 wlan0
192.168.0.0     0.0.0.0         255.252.0.0     U         0 0          0 wlan0
```

Exercise 3: Subnetting

IP address	172.21.240.90	10101100	00010101	11110000	01011010
Class B	255.255.0.0	11111111	11111111	00000000	00000000
Subnet mask	255.255.255.224	11111111	11111111	11111111	11100000

Exercise 3: Subnetting

IP address	172.21.240.90	10101100 00010101 11110000 01011010
Class B	255.255.0.0	11111111 11111111 00000000 00000000
Subnet mask	255.255.255.224	11111111 11111111 11111111 11100000
Subnet ID	1922	~~10101100 00010101~~ 11110000 010~~00000~~

Exercise 3: Subnetting

IP address	172.21.240.90	10101100 00010101 11110000 01011010
Class B	255.255.0.0	11111111 11111111 00000000 00000000
Subnet mask	255.255.255.224	11111111 11111111 11111111 11100000
Subnet ID	1922	~~10101100 00010101~~ 11110000 01~~000000~~

- **IP address `AND` (`NOT` subnet mask) = host ID**

IP address	172.21.240.90	10101100 00010101 11110000 01011010
Subnet mask	255.255.255.224	11111111 11111111 11111111 11100000
Inverse subnet mask	000.000.000.31	~~00000000 00000000 00000000 000~~11111
Host ID	26	~~00000000 00000000 00000000 000~~11010

Exercise 4: IPv4 Checksum

RFC 791, page 14

„The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header. For purposes of computing the checksum, the value of the checksum field is zero".

- To calculate the checksum of the packet, the sum of each 2 byte word inside the header must be calculated. The checksum field itself is skipped here!
  `4500 + 0034 + B612 + 4000 + 4006 + 0A00 + 008B + 5BC6 + AEE0 = 2907D`
- Next, the result of the calculation is converted to binary:
  `2907D` $\Longrightarrow$ `10 1001 0000 0111 1101`
- The first two bits are the carry and need to be added to the rest of the value:
  `10 + 1001 0000 0111 1101 = 1001 0000 0111 1111`
- Next, every bit of the result is flipped to obtain the checksum:
  ```
     1001 0000 0111 1111
  => 0110 1111 1000 0000
  ```
- The result `0110 1111 1000 0000` is equal to the value `6F80` in hexadecimal notation, as already shown in the original IP packet header.

IPv4: Verify checksum

- To verify a checksum, the same procedure is used as above, with a single exception: The original header checksum is not omitted.
  `4500 + 0034 + B612 + 4000 + 4006 + 6F80 + 0A00 + 008B + 5BC6 + AEE0 = 2FFFD`
- Next, the result of the calculation is converted to binary:
  2FFFD $\Longrightarrow$ 10 1111 1111 1111 1101
- The first two bits are the carry and need to be added to the rest of the value:
  `10 + 1111 1111 1111 1101 = 1111 1111 1111 1111`
- Next, every bit of the result is flipped:
  ```
     1111 1111 1111 1111
  => 0000 0000 0000 0000
  ```
- This indicates: No error detected! Any result, which is $\neq 0$ indicates: Error!

Source: RFC 791 and Wikipedia

Exercise 5: Address Types and Spaces

- Private addresses (unique local addresses in IPv6)
  - "have no global meaning"[2]
  - "routing information [. . .] shall not be propagated"[2] in the Internet, and
  - "packets with private source or destination addresses should not be forwarded"[2]
- May be forwarded inside a LAN (→ *link-local addresses* are never forwarded)
- Edge routers ideally filter traffic using address from private address space

[2] RFC 1918

Exercise 5: Address Types and Spaces

- Private addresses (unique local addresses in IPv6)
  - "have no global meaning"[2]
  - "routing information [. . .] shall not be propagated"[2] in the Internet, and
  - "packets with private source or destination addresses should not be forwarded"[2]
- May be forwarded inside a LAN (→ *link-local addresses* are never forwarded)
- Edge routers ideally filter traffic using address from private address space

Pinging broadcast addresses

```
user@host> ping -b 10.0.34.255
PING 10.0.34.0 (10.0.34.0) from 10.0.34.197 : 56(84) bytes of data.
64 bytes from 10.0.34.197: icmp_seq=1 ttl=64 time=0.049 ms
64 bytes from 10.0.34.236: icmp_seq=1 ttl=255 time=0.163 ms (DUP!)
64 bytes from 10.0.34.206: icmp_seq=1 ttl=255 time=0.211 ms (DUP!)
64 bytes from 10.0.34.196: icmp_seq=1 ttl=255 time=0.213 ms (DUP!)
64 bytes from 10.0.34.181: icmp_seq=1 ttl=255 time=0.220 ms (DUP!)
64 bytes from 10.0.34.174: icmp_seq=1 ttl=255 time=0.243 ms (DUP!)
64 bytes from 10.0.34.133: icmp_seq=1 ttl=255 time=0.245 ms (DUP!)
```

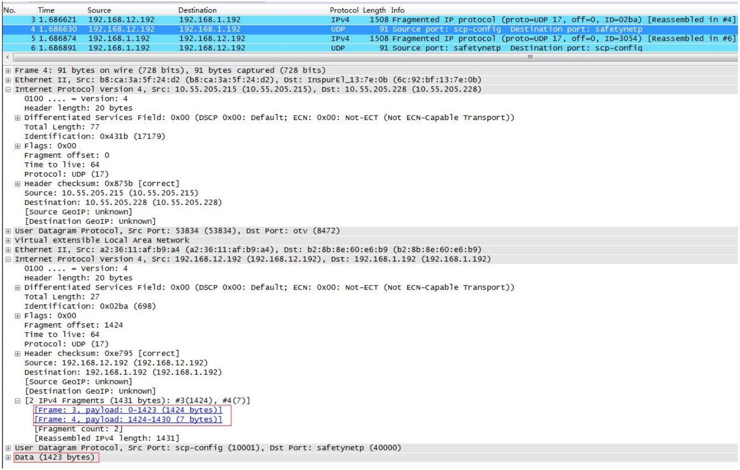
[2] RFC 1918

Exercise 6: Fragmenting IP Packets

- Any router can fragment (unless the `DF` bit is not set)
- Only the receiver reassembles
- In IPv4:
  - Any router "must be able to forward a datagram of 68 octets without further fragmentation"[3]
  - Any host "must be able to receive a datagram of 576 octets either in one piece or in fragments to be reassembled"[3]
- "IPv6 requires that every link in the internet have an MTU of 1280"[4] octets or greater

[3] RFC 791
[4] RFC 2460

Exercise 6: Fragmenting IP Packets



Source: https://hustcat.github.io/