Computer Networks - Network Layer (Data Plane)

JJ McCauley

4/22/25

Chapter 4's Problems: 8, 9, 17

8 Longest Prefix Matching in Datagram Forwarding

We are given a datagram network using 32-bit host addresses, assuming that a router has four links (numbered 0-3), and packets are forwarded to the link interfaces as:

Destination Address Range	**Link Interface**
11100000 00000000 00000000 00000000 through 11100000 00111111 11111111 11111111	0
11100000 01000000 00000000 00000000 through 11100000 01000000 11111111 11111111	1
11100000 01000001 00000000 00000000 through 11100001 01111111 11111111 11111111	2
otherwise	3

8.1 Forwarding Table with Five Entries, Longest Prefix Matching, and Correct Link Forwarding Interfaces

In order to complete this question, we must first understand that each interface handles traffic for a specific range of destination addresses. Additionally, *longest prefix matching* means that, for any given destination address, the router matches the entry with the most bits in common (from the left).

Within the table, we can interpret the address ranges as range starts and range ends. For example, interface 0 has a range start of 11100000 00000000 00000000 00000000, and a range end of 11100000 00111111 11111111 11111111. Using this, we can then find the CIDR Prefixes in the form **A.B.C.D/N**.

8.1.1 Determining *N* (Number of Shared Bits)

To determine this, we must determine how many bits each address shares:

Destination Address Range	Bits Shared
11100000 00000000 00000000 00000000 through **11100000 00**111111 11111111 11111111	10
11100000 01000000 00000000 00000000 through **11100000 01000000** 11111111 11111111	16
11100000 01000001 00000000 00000000 through **1110000**1 01111111 11111111 11111111	7
otherwise	-

These numbers of bit shared gives us the N in the CIDR notation of **A.B.C.D/N**. In order to find the first part (A.B.C.D), we must convert all of these ranges into dotted decimals.

8.1.2 Determining *A.B.C.D* (Dotted Decimals of IPs)

In order to find the A.B.C.D portion of the prefix, we will find the dotted decimal of the shared bits.

For instance, we will walk through interface 0. For this address range, there are 10 shared bits, being:

11100000 00

Then, we will divide up the segments into octets (8-bit segments). Any segment that does not match, we will simply zero out. This gives us *Octet 1* as 11100000, *Octet 2* as 01000000, and *Octet 3 & 4* as 00000000.

We can then use convert these binary octets to decimals, to get *Octet 1* as **224**, *Octet 2* as **64**, and *Octet 3 & 4* remain as **0**. Putting this into the CIDR notation, we get **224.64.0.0**. By doing these calculations on the remaining interfaces, we can derive the table:

Destination Address Range	Dotted Decimals (Tentative CIDR Notation, excluding N)
11100000 00000000 00000000 00000000 through **11100000 00**111111 11111111 11111111	224.64.0.0
11100000 01000000 00000000 00000000 through **11100000 01000000** 11111111 11111111	224.0.0.0
11100000 01000001 00000000 00000000 through **1110000**1 01111111 11111111 11111111	224.0.0.0
otherwise	0.0.0.0

8.1.3 Putting it all together - Forwarding Table

Now that we know the N value and the $A.B.C.D$ format, we can combine them to get the following notations:

Destination Address Range	Dotted Decimals (Tentative CIDR Notation, excluding N)
11100000 00000000 00000000 00000000 through **11100000 00**111111 11111111 11111111	224.64.0.0/10
11100000 01000000 00000000 00000000 through **11100000 01000000** 11111111 11111111	224.0.0.0/16
11100000 01000001 00000000 00000000 through **1110000**1 01111111 11111111 11111111	224.0.0.0/7
otherwise	0.0.0.0/0

However, **there is an issue here**. We are asked to *provide five table entries*, with there currently only being *four*. In order to satisfy this constraint, we can *break down interface 2* into two separate entries with masks (N) of 8. This is because we already have two more-specific entries (interfaces 0 and 1), which observe a smaller range of bit differences. Additionally, interface 8 only observes the first 7 bits, we can provide separate cases for the 8th bit. This not only satisfies the constraint, but also provides more specific routes for the bits which could improve performance.

Looking at interface two, we see that the range is ultimately 224.0.0.0 to 225.225.225.225 (since the 8th and all following bits are not 'matching'). We can therefore break this into two separate ranges of 224.225.225.225 to 225.225.225.225, resulting in the additional entries of:

224.0.0.0/8 (Interface 2)
225.0.0.0/8 (Interface 2)

Using these new entries, we can create the **final forwarding table** of:

Prefix	**Mask (*N*)**	**Interface**
224.0.0.0	/10	0
224.64.0.0	/16	1
224.0.0.0	/8	2
225.0.0.0	/8	2
0.0.0.0	/0	3 (default)

8.2 Forwarding Decisions via Longest-Prefix Matching

We are asked to determine the appropriate link interfaces of datagrams using the following destination addresses.

11001000 10010001 01010001 01010101
11100001 01000000 11000011 00111100
11100001 10000000 00010001 01110111

Since we have already described the **forwarding table** (above), the process to determine these interfaces is very simple. This process will only involve two steps: converting the binaries to decimals (prefixes) and mapping to the correct prefix/interface.

Firstly, we will map convert each binary segment to their respective decimal prefixes such that:

11001000 10010001 01010001 01010101 = **200.145.81.85**
11100001 01000000 11000011 00111100 = **225.64.195.60**
11100001 10000000 00010001 01110111 = **225.128.17.119**

Then, we must map each decimal prefix to each entry in the forwarding table, checking the entries in order of longest prefix (largest mask, n). Below is a table demonstrating this:

`Prefix`	**224.64.0.0/16**	**224.0.0.0/10**	**224.0.0.0/8**	**225.0.0.0/8**	**0.0.0.0/0**
`200.145.81.85`	X	X	X	X	Match
`225.64.195.60`	X	X	X	Match	
`225.128.17.119`	X	X	X	Match	

As shown above, each entry is checked (by mask) until an appropriate one is reached. Therefore, by matching these decimal prefixed with the **forwarding table above**, we can derive the following **interface mapping**:

11001000 10010001 01010001 01010101 maps to **Interface 3 (default)**
11100001 01000000 11000011 00111100 maps to **Interface 2**
11100001 10000000 00010001 01110111 maps to **Interface 2**

9 Address Range Mapping for 8-bit Forwarding Table

We are given a datagram network with an 8-bit forwarding addresses and the following forward table (assuming the router uses longest prefix matching):

Prefix Match	**Interface**
00	0
010	1
011	2
10	2
11	3

In order to find the associated range of destination host addresses and the number of addresses in each range, we firstly must understand that this is representing 8-bit numbers (*xxxxxxxx*). Using this information, we can determine the ranges (and therefore the counts), then map those onto the appropriate interfaces using the provided forwarding table.

Firstly, to retrieve the counts, we must find the minimum and maximum potential values of the binary. We can do this but subbing in 0 and 1 to find the minimum and maximum, respectively.

For example, the first prefix match (00xxxxxx) can be 00000000 as the minimum (decimal value of 0) or 00111111 as the maximum (decimal value of 63). Therefore, we know the range is 0-63, making the count 64 (0 and 63 inclusive). Under this same logic, we can derive the following values:

`Prefix`	**Addresses Covered**	**Decimal Range**	**Count**
`00`	00xxxxxx	0–63	64
`010`	010xxxxx	64–95	32
`011`	011xxxxx	96–127	32
`10`	10xxxxxx	128–191	64
`11`	11xxxxxx	192–255	64

Now, we can map these ranges and counts onto their respective interfaces, such that:

Interface 0's range is 0-63 with a count of 64 **Interface 1**'s range is 64-95 with a count of 32 **Interface 2**'s range is 96-127 with a count of 32 **Interface 2**'s range is also 128-191 with a count of 64 **Interface 0**'s range is 192-255 with a count of 64

By combining both of Interface 2's metrics, we obtain the **final table** of:

`Interface`	**Prefix(es)**	**Decimal Range**	**Count**
`0`	00	0–63	64
`1`	010	64–95	32
`2`	011, 10	96–191	96
`3`	10	192–255	64

17 Datagram Count for Sending a 5-Million Byte MP3

We are tasked with determining the number of datagrams required to send a 5-million byte MP3 file, given that datagrams are limited to 1500 bytes (including the header) and there is a 20-byte IP header.

This problem is extraordinarily simple once we understand the context. Firstly, we must calculate the amount of data in each diagram. Since the only additional overhead to the data would be the IP header, this can be calculated as:

$$\text{Data per Datagram} = \text{Max Datagram Size} - \text{IP Header Size} = 1500 - 20 = 1480 \text{ bytes}$$

Then, since we know that total file size of the MP3 file is 5,000,000 bytes, we can directly determine the number of datagrams with the equation:

$$\text{Datagram Count} = \frac{\text{Total File Size}}{\text{Datagram Size}} = \frac{5000000 \text{ bytes}}{1480 \text{ bytes}} \approx 3378.38$$

Since we can not have a partial datagram, this will be *rounded up* to the nearest whole number. Therefore, **3379 datagrams** are required to send this 5,000,000 byte MP3 file with payloads of 1480 bytes each.