Computer Networks *(Redes de Computadores - RCOMP)* – 2024/2025

Laboratory Class 04 (PL04 – 3 hours)

Project 1/Sprint 1 review presentations. Classless IPv4 addressing. IPv4 networks dimensioning with classless addressing (VLSM). Practical exercises.

1. Classless IPv4 addressing

Classful addresses (8-, 16-, and 24-bits prefix-lengths) lead to severe addresses wasting, in fact, often the real needs don't match these prefix-lengths. Some examples:

A two nodes network (a dedicated connection) – will require 256 addresses (a C class network)

A network with three hundred nodes – will require 65536 addresses (a B class network)

With classless IPv4 addressing, other prefixes can be used, this allows a far better adjustment to the real needs.

All principles for classful addressing are also valid for classless addressing, however:

- The network mask (prefix-length) can no longer be established from the value of the address's first bits, therefore, to identify a network, the network prefix length must be included along with the network address itself.

- The four-octet dot-decimal representation of the network mask is still used, but it's far less convenient for addresses analysis because now prefix lengths may not match the octets' limits. Most address analysis will have to be done in binary representation.

Network masks expressed in dot-decimal representation are now hard to interpret, for instance to represent a 27-bits long prefix, the network mask is 255.255.255.224, this is easier to read from the binary representation (network bits in red): **11111111.11111111.11111111.11100000**.

As an alternative to dot-decimal network masks, the CIDR (Classless Inter-Domain Routing) representation can be used. In CIDR notation the network prefix-length is added to the address, separated by a slash. So, in the above case, it would be **/27**.

Most address analysis must now be performed in binary representation. In general, addresses and masks must be converted to binary representation, analysis performed and then results converted back to dot-decimal representation.

Usually, the binary analysis will be focused on the octet where the network prefix length is located, so there is no need to represent all 32 bits in binary, only that octet. Let's take an example:

Network: 194.120.8.0/21

With this prefix, the leftmost 21 bits are used to identify the network, and thus, the remaining 11 bits are used to identify nodes within that network.

The number of addresses in the network (address space) is $2^{11} = 2048$, the number of valid node addresses this network can hold is 2046 nodes (2048-2)

The network prefix lies on the **third octet**, so the binary analysis is relevant only on that octet.

The network mask is: **11111111.11111111.11111000.00000000**, using the binary representation for the third octet only, we have: **255.255.(11111000)$_2$.0**, in dot-decimal **255.255.248.0**.

The network address is **194.120.(00001000)$_2$.0**, in dot-decimal **194.120.8.0**

The first valid node address is **194.120.(00001000)$_2$.1**, in dot-decimal **194.120.8.1**.

The broadcast address is **194.120.(00001111)$_2$.255**, in dot-decimal **194.120.15.255**.

1.1. Manual conversion from binary to decimal and vice versa

Manually converting an octet from its binary representation to the decimal representation is fairly simple. Each bit represents a decimal value, these decimal values are the powers of two, thus, the less significant bit decimal value is one, and each following bit doubles the decimal value of the previous bit (Table 1).

Table 1 - Decimal values of each bit in an octet

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

To convert from binary to decimal you can just sum the decimal values of all bits with value one. For instance, **01100111** can be converted to decimal representation by calculating the sum:

64+32+4+2+1 = 103

Converting from decimal to binary is the reverse operation, you will try to deduct each bit decimal value from the original decimal representation, starting with the most significant bit. If deducting is possible, it means that bit is one, otherwise is zero.

Let´s take an example: **202**

202 – 128 = 74 (this means bit with decimal value 128 is one)

74 – 64 = 10 (this means bit with decimal value 64 is also one)

10 – 32 would be negative (this means bit with decimal value 32 is zero)

10 – 16 would be negative (this means bit with decimal value 16 is also zero)

10 – 8 = 2 (bit with decimal value 8 is one)

2 – 4 would be negative (bit with decimal value 4 is also zero)

2 – 2 = 0 (bit with decimal value 2 is one)

0 – 1 would be negative (bit with decimal value 1 is zero)

So, the binary representation of **202** is **11001010**

1.2. Practical exercises

For each given network, expressed in CIDR notation, present the network mask in dot-decimal notation, the network's first valid node address and the broadcast address:

a) **170.20.0.0/22**

b) **200.100.20.192/26**

c) **120.64.0.0/12**

d) **191.123.90.104/30**

e) **138.20.64.0/18**

2. IPv4 networks dimensioning with classless addressing (VLSM).

IP network dimensioning consists of establishing the appropriate network mask (prefix-length) to meet the maximum number of nodes the network must support.

Each network mask sets the number of bits that will be available to identify nodes within it and, therefore, the maximum number of nodes it can have. We will be calling **address block** to the set of different addressed within a network, remember however that, from those, two addresses are reserved and cannot be used as node address.

The biggest prefix-length applicable in IPv4 is 30 bits, this means only two bits are left to identify addresses within such a network, the block size is four and the number of valid node addresses is only two. This prefix is commonly used for dedicated connections between routers where only two valid node addresses are in fact required.

For each bit reduced in the prefix-length the **block size doubles**, inversely for each bit added to the prefix-length, the **block size is reduced to half**. Keeping in mind some significant prefix-length values (highlighted in red at Table 2), it's easy to reach the block size for any prefix-length.

Table 2 - Some IPv4 prefix-lengths and resulting addresses block sizes

Network prefix-length	Addresses block size	Addresses block size in dot-decimal	Number of valid node addresses
/30	**$2^2 = 4$**	**0.0.0.4**	**4 - 2 = 2**
/29	$2^3 = 8$	0.0.0.8	8 - 2 = 6
/28	$2^4 = 16$	0.0.0.16	16 - 2 = 14
/27	$2^5 = 32$	0.0.0.32	32 - 2 = 30
/26	$2^6 = 64$	0.0.0.64	64 - 2 = 62
/25	$2^7 = 128$	0.0.0.128	128 - 2 = 126
/24	**$2^8 = 256$**	**0.0.1.0**	**256 - 2 = 254**
/23	$2^9 = 512$	0.0.2.0	512 - 2 = 510
/22	$2^{10} = 1024$	0.0.4.0	1024 - 2 = 1022
/21	$2^{11} = 2048$	0.0.8.0	2048 - 2 = 2046
/20	$2^{12} = 4096$	0.0.16.0	4096 - 2 = 4094
/19	$2^{13} = 8192$	0.0.32.0	8192 - 2 = 8190
/18	$2^{14} = 16384$	0.0.64.0	16384 - 2 = 16382
/17	$2^{15} = 32768$	0.0.128.0	32768 - 2 = 32766
/16	**$2^{16} = 65536$**	**0.1.0.0**	**65536 - 2 = 65534**
(…)	(…)		(…)

Expressing the block size in dot-decimal is convenient for prefix-lengths below 24 bits, as we will see ahead when the next network address is calculated by adding to the current network address the block size.

By adding one bit to the prefix-length, a block is split into two blocks of half the size of the original block. One resulting half-size block starts at the same address as the initial block and the other at the middle of the initial block. This is always valid whatever the initial block may by.

By reducing one bit to the prefix, two blocks are merged to form a new block with double size, this is usually referred to as blocks aggregation.

Of course, reducing one prefix-length bit (aggregation) over two existing address blocks is not always valid, it depends on the address blocks.

Example:

Blocks (networks) 120.10.5.0/24 and 120.10.6.0/24 cannot be aggregated by reducing the prefix to 23 bits

Let's try:

120.10.5.0/24 - 120.10.(0000 0101).0

120.10.6.0/24 - 120.10.(0000 0110).0

Reducing the prefix-length to 23 bits results in:

120.10.5.0 - 120.10.(0000 0100).0 , thus, 120.10.4.0/23

120.10.6.0 - 120.10.(0000 0110).0 , thus, 120.10.6.0/23

We can see they end up in different 23 prefix-length address blocks, not the same. So this aggregation is not valid.

However, other aggregations are valid:

120.10.4.0/23 = 120.10.4.0/24 + 120.10.5.0/24

And also: 120.10.6.0/23 = 120.10.6.0/24 + 120.10.7.0/24

And also: 120.10.4.0/22 = 120.10.4.0/23 + 120.10.6.0/23

To summarize: any address block can always be split into two (with half the size each), however, two equal size address blocks may or may not be aggregable into a single double size block.

Once we know the address block size, knowing each network address it's easy. To get the next network's address you can simply add the address block's size to the network's address, thus, network addresses can be assigned sequentially.

Example:

Given the 190.130.0.0/24 addresses block, use it to establish several IPv4 networks capable of holding up to 20 nodes each.

The best fitted prefix for up to 20 nodes is the 27 bits prefix length, resulting in address blocks of 32 addresses each (up to 30 nodes)

First network - 190.130.0.0/27

Second network – 190.130.0.32/27 (0 + 32)

Third network – 190.130.0.64/27 (32 + 32)

Fourth network – 190.130.0.96/27 (64 + 32)

Fifth network – 190.130.0.128/27 (96 + 32)

(…)

Once prefix-lengths are established, the problem consists in taking the provided address space (addresses block) and split it into several non-overlapping address valid spaces (addresses blocks) one for each network.

Because each address space or addresses block is also a valid IPv4 network, the split of a bigger network into several smaller networks is also known as **subnetting**. And those resulting networks are also called **subnets**.

Often, different sizes will be required for each subnet, so different prefix-lengths must be used for each, this is called **Variable Length Subnet Mask** (VLSM).

We already know every address block (network) can always be split into two blocks of half size each, consequently, **wherever a block starts at that same address also starts any smaller block** (bigger prefix-length, smaller network).

This means, even with VLSM, we can still use the sequential assignment strategy as far as **bigger blocks** (smaller prefixes) **are assigned first**.

In the sequential assignment strategy, we get the next network address by adding to the current network address its block size, this results in the starting point of the next address block with the same size (same prefix-length), and thus at that same point also starts a block with any bigger prefix-length.

Let's test this:

Given the 190.130.0.0/24 addresses block we want to assign addresses for three networks: one with up to 20 nodes, another with up to 60 nodes and yet another with up to 100 nodes.

As requirements for each network are different, we will be using VLSM

For the, up to 20 nodes network – a 32 addresses block – 27 bits prefix

For the, up to 60 nodes network – a 64 addresses block – 26 bits prefix

For the, up to 100 nodes network – a 128 addresses block – 25 bits prefix

The sequential assignment strategy can be used, **as far as we start by the bigger blocks**:

First, the 128 addresses block:	190.130.0.0/25	
Second, the 64 addresses block:	190.130.0.128/26	(0 + 128)
Finally, the 32 addresses block:	190.130.0.192/27	(128 + 64)

If we disregard the rule **bigger blocks first**, the solution may not be valid:

First, a 64 addresses block:	190.130.0.0/26	
Second, a 128 addresses block:	190.130.0.64/25	(0 + 64)

The problem is the second block doesn't exist. Notice that, with a 25 bits prefix, the bit with decimal value 64 is beyond the network prefix, and thus, it should be zero when representing the network address.

When managing prefixes with less than 24 bits, block sizes will be above 255 addresses, then to add the address block size to the current network address it's more convenient to express the block size in dot-decimal representation as well (as presented in the table before).

Example for prefixes less than 24 bits

Given the 190.130.128.0/17 addresses block, use it to assign network addresses to three networks with the following networks capacities: up to 500 nodes, up to 1000 nodes and up to 2000 nodes

500 nodes – 512 addresses block – 23 bits prefix

1000 nodes – 1024 addresses block – 22 bits prefix

2000 nodes – 2048 addresses block – 21 bits prefix

512 in binary is 10 00000000, in dot-decimal notation 2.0

1024 in binary is 100 00000000, in dot-decimal notation 4.0

2048 in binary is 1000 00000000, in dot-decimal notation 8.0

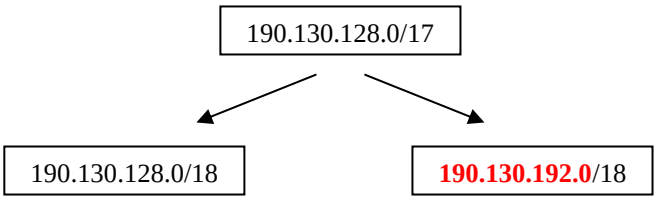
As before, the sequential assignment strategy can be used if we follow the rule **bigger blocks first**.

2048 addresses block	190.130.128.0/21	
1024 addresses block	190.130.136.0/22	**(128.0 + 8.0)**
512 addresses block	190.130.140.0/23	**(136.0 + 4.0)**

This technique of establishing each subnet size (number of addresses) and getting the next subnet address by adding the subnet size to the subnet address is always valid, as far as the **bigger blocks first** rule is applied. Though other techniques can be used, one most popular is by representing subnets as a binary tree. This comes from the fact that each block is split into two blocks when one bit is added to the prefix-length.

One advantage of this approach is it provides a better overall view of the address space being used, but in fact it's similar to the previous method.

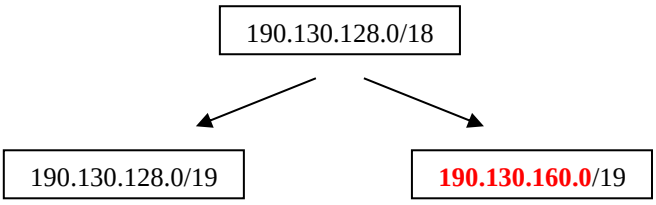
Let's use the same example as above, having already established the required prefix-lengths for each subnet, we can start splitting the provided block to the point we need, this should be done step by step, meaning advancing one bit on the prefix-length at a time:



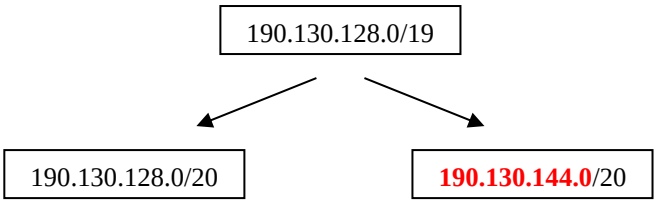
When splitting the original network (190.130.128.0/17) into two subnets, by adding one bit to the prefix-length, one resulting network will have the same address value (190.130.128.0/18). To get the other subnet address value, it's just a matter of adding the block size, from the table: **the /18 prefix-length creates a block with size: 0.0.64.0**, so it will be **190.130.128.0 + 0.0.64.0 = 190.130.192.0**.

The first biggest subnet to be assigned is using a 21 bits prefix-length, so we are not quite there yet.

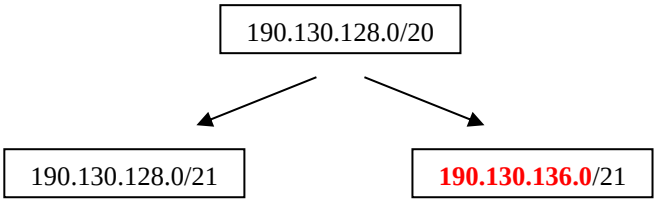
We may leave the 190.130.192.0/18 block for later use if needed and proceed by splitting the 190.130.128.0/18 block in two.



The /19 prefix-length creates blocks with size: 0.0.32.0, so the second subnet address is 190.130.160.0/19, again we may leave this second block for later use and focus on the first one:

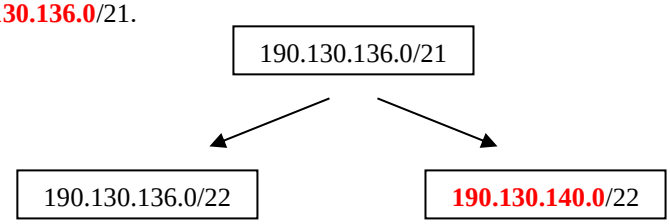


The /20 prefix-length creates blocks with size: 0.0.16.0, so the second subnet address is 190.130.144.0/20, again we may leave this second block for later use and focus on the first one:

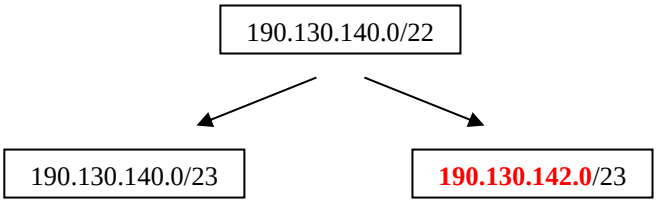


Now we have a block size matching the biggest subnet to be assigned, so it will be assigned address **190.139.128.0/21**. The second block address is obtained by adding the block size (0.0.8.0).

To assign addresses to the remaining two subnets we could use any of the block left being unused (**190.130.192.0**/18; **190.130.160.0**/19; **190.130.144.0**/20; **190.130.136.0**/21). For the sake of simplifications when latter creating routing tables it's a good practice to use the block that is nearest to the already assigned, so we may choose block **190.130.136.0**/21.



The first block is ready to be assigned to one additional network (/22), so the network with prefix /22 is assigned address **190.130.136.0/22**. The second block address is obtained by adding the block size 0.0.4.0. Keeping the same methodology, we will now pick block 190.130.140.0/22 for the remaining network to be assigned (/23).



So, the last network to be assigned will have address: **190.130.140.0/23**

Several address blocks were left unused, as leftovers, and could be used to assign addresses to additional networks, they are: **190.130.192.0/18; 190.130.160.0/19; 190.130.144.0/20; 190.130.142.0/23**.

It this resolution, a step-by-step style was used, one branch at a time, but the entire binary tree may also be drawn at once, though often it may become more confusing if you get out of drawing space.

3. Practical exercises

3.1. Use the 173.30.60.128/25 addresses block to assign network addresses to two networks capable of supporting up to 30 nodes and one network capable of supporting up to 8 nodes.

3.2. Define network addresses for networks A, B and C within the 180.30.0.0/20 addresses block. Network A must support up to 2000 nodes, network B up to 500 nodes and network C up to 200 nodes.

3.3. Reconsidering a problem from two weeks ago

Consider again the diagram in Figure 1 **representing several IPv4 networks interconnected by three routers.**

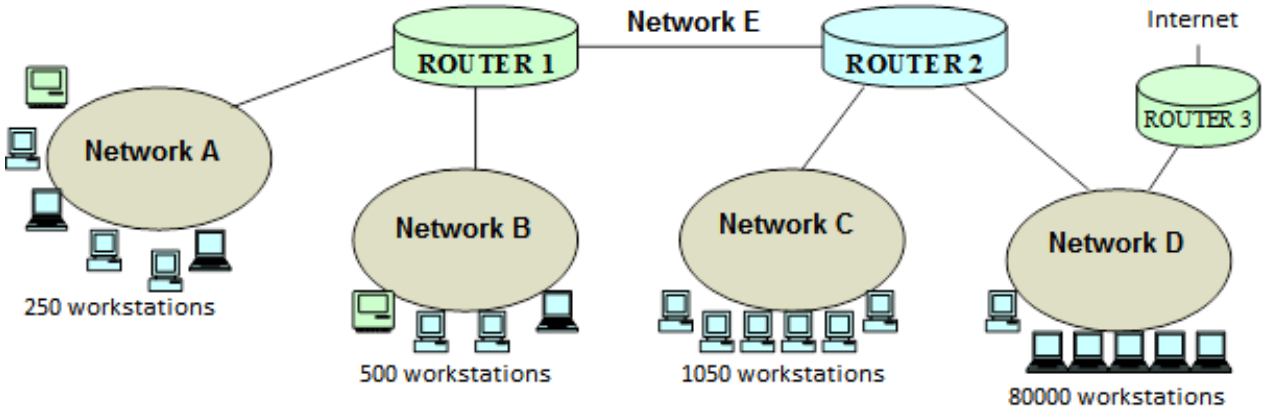


Figure 1 - Networks layout

a) Use the **10.48.0.0/14** addresses block to assign addresses to all represented networks (A; B; C; D and E), they must be capable of housing the represented number of workstations, find a solution using the least possible addresses from the provided addresses block.

b) Accordingly, set the IPv4 node addresses for each router interface.

c) Define each router's static routing table.

Important remarks:

- The image presents the number of workstations to be supported on each network, however, when establishing the number of nodes to be supported, connected routers must be taken in account.
- In this layout the dedicated connection between Router 1 and Router 2 is clearly identified as a network and referred on the text, nevertheless, even if that was not the case, this exercise resolution should always encompass it.