NETWORK ADDRESSES, ROUTING, AND MATH

WHY MATH?

When sending packets using IP addresses, each node must be able to distinguish between destination addresses in the local segment or addresses that are more than one hop away. The basic structure of IP addresses provides the basis of this distinction.

We've seen that IPv4 addresses are composed of 32 bits divided into two parts of varying lengths. The network part of the address will always begin from the most significant bit (MSB) position. The host portion of the address occupies the remaining bits down to the least significant bit (LSB) position.

Given a source address and a destination address, hosts can make decisions about forwarding by comparing the bits in the network portion of the two addresses. This is a simple equality/inequality comparison. The network stack will take one of two actions depending on the outcome of the comparison.

1) If the network prefix of the destination matches the source, the host will attempt to forward the packet to the destination on the local segment directly. In IPv4, ARP will be used to resolve the link layer address of the destination.
2) If the network prefix of the destination is not a match, the host will consult its routing table to determine the IP address of a router that can forward the packet. ARP will be used once again if the link layer address of the next hop is not already stored in the host's ARP table.

In the following sections, we'll look at how this test is implemented and what the implications are for assigning IP addresses. Our focus in this document is working with the numerical structure of IP addresses. Be aware that we may gloss over some protocol nuances to maintain this emphasis.

ADDRESS CALCULATIONS

It should go without saying that Layer 3 devices on high-speed networks incur a cost associated with this operation as hundreds of thousands or millions of packets pass through the stack each section. The efficiency of the Internet Protocol is contingent on how long it takes to obtain the prefix from an address and run a comparison. Fortunately, the structure outlined above lends itself to bitwise operations that can implemented efficiently in hardware or in software.

The term masking describes the process used beneath the hood to obtain the prefix from a specified address. Masking works by combining the address with a special sequence of mask bits using the bitwise AND operation. Depending on the value of the mask at each bit location, the bits in the original value will either be kept or set to zero. Based on the properties of the bitwise AND, we set a mask bit to one if we want to maintain the corresponding position from the original value or zero if we want to clear the position in the original value. IPv4 network masks are designed to keep bits associated with a network prefix and to clear bits associated with the host portion of the address, so we construct a 32-bit value such that the MSB positions corresponding to the prefix are set to one and the remaining bits are set to zero.

A network with a 16-bit prefix has a network mask composed of 16 ones followed by 16 zeros when read from left to right. We often notate this network as a /16 to indicate prefix length, but we can convey the same data by directly reporting that the network mask is equal to 255.255.0.0. Likewise, a 23-bit network prefix corresponds to a mask with 23 ones followed by 9 zeros. This /23 network has a mask of 255.255.254.0.

DAMN YOU, BASE-2

If it was easy for us to work directly with binary values, network configuration and operations like masking would not be much of a challenge. With very little practice, we could perform routing and forwarding operations with a glance.

Unfortunately, binary values are slightly verbose for our everyday operational needs. Imagine communicating 32-bit IPv4 or 128-bit IPv6 addresses to colleagues or creating configuration files where network addresses are written in long-form binary. Humans are much better equipped for working with values expressed in decimal (or even hexadecimal) since we can compress a long sequence of bits into a shorter sequence of digits.

The challenge of network address operations comes from the fact that our numeric conventions obscure the bit patterns from direct observation. Without a bit of attention, we'd be unlikely to realize that the decimal value of 224 represents three bits of masking on a single byte. As we work through examples, we'll learn to reduce the confusion that comes from the mismatch of numeric representations.

DON'T FEAR THE MASK

While IP address notation is not perfect,[1] the simplicity of the structure does lend itself to pattern recognition if we know a bit[2] about base-2 arithmetic. In other words, some focused practice will build the intuition we need to solve network-related problems with somewhat less cursing and outrage.

We've already established the relationship between network prefix length and network masks. Let's look more closely at how we translate back and forth between the decimal representation of a network mask and the prefix length.

Just like IPv4 addresses, IPv4 network masks are 32-bits in length. By convention, masks are written in dotted decimal to indicate the four octets of the address. We know that a mask, unlike standard addresses, will be composed of a sequence of ones followed by a sequence of zeros. If a given bit in any octet is set to one, the bits and octets to its left will also be one. If a bit in any octet is zero, the bits and octets to the right will also be zero. We never have reason to worry about masks that interleave alternating patterns of ones and zeros.

The implication of this pattern is that there are only eight possible states for any octet. This observation is true whether we are looking at masks in binary or in decimal. Worst case, we only need to memorize eight values to answer basic questions about network prefixes or masks. Moreover, a basic grasp of binary representation

[1] from the viewpoint of any student required to work with addresses in depth
[2] #laughtrack

provides enough grounding to solve these problems without having to commit a lot of numbers to memory or relying on external aids.

A single octet can encode 2^8 (256) possible values ranging from 0 to 255. These two values, corresponding to all zeros or all ones in binary representation, are the most common numbers to appear in IPv4 masks. Due to the structure of network prefixes, they are guaranteed to appear in three out of four octets in a valid mask. There are six more possible values for the remaining octet, assuming when the prefix length is not a clean multiple of eight bits.

When we divide the prefix length by 8 using integer division, our whole-number result tells us how many octets to set as 255 (beginning from the left). The remainder of the division tells us how many bits to account for in the next octet.

If we have one bit remaining, the decimal value of the octet will be 128 (2^7) since it is assigned to the most significant bit. Adding one more bit of prefix, will increase this part of the mask to 192. 192 is the sum of 128 and 64 (2^7 + 2^6). With each additional bit, we increase by half what we did the time before to account for the diminishing powers of two associated with each bit position.

	MSB							LSB
POS (N)	7	6	5	4	3	2	1	0
2^N	128	64	32	16	8	4	2	1

In the following table, we have generated the entire sequence by counting upward with the decreasing powers of two:[3]

PREFIX LEN	1	2	3	4	5	6	7	8
2^N	128	64	32	16	8	4	2	1
MASK (DEC)	128	192	224	240	248	252	254	255
MASK (HEX)	80	C0	E0	F0	F8	FC	FE	FF

If you pay close attention, you might notice an even simpler pattern. For a given remainder, the sum of the mask and the value of the bit at the corresponding bit position always equals 256. We can flip this property around to avoid adding four, five, or six powers of two at a time.

Taking the number of bits for a given octet, determine the power of two corresponding to the rightmost bit (least significant mask bit). For example, the power of two corresponding to the 5-bit prefix is 8 = 2^3. Subtracting this value from 256, we find the mask value for the related octet. Given the 5-bit prefix, our mask will be 256 - 8 = 248.

COUNTING NETWORKS AND HOSTS

[3] hexadecimal representation included for funsies and future reference with IPv6

Before we move on from network masks, we need to understand a couple more pieces of information that we can a subnet mask, i.e., the number of possible networks and hosts associated with the given mask.

Ignoring this business of network prefixes and subnet masks, we can generate 2^32 (approximately 4.3 billion) addresses, from the 32-bit address space of IPv4. For the reasons we've already discussed, we partition the address space into smaller networks by assigning some of those bits as a network prefix. We need to understand network size in terms of these subsets.

In the simplest of networks, we don't need much explanation to find the answer. By subtracting the network prefix length from 32 and evaluating the difference as a power of 2, we determine how many host addresses are available for each network ID. A prefix length of 16 leaves another 16 host bits for 2^16 = 65,536 values in the host part of the address. A prefix length of 24 leaves 8 bits for 256 unique addresses.

We can add another dimension to this conversation by remembering that organizations and individuals are only able to use a relatively small subset of the IPv4 address space to meet their network addressing needs. When we move beyond he simplest networks, we also learn that there are a variety of reasons an organization might want to divide their address space into smaller networks. In practice, we might want to divide a large address allocation into smaller subnets by adding some extra bits to the network prefix.

Taking these concepts together, we need to determine the right prefix length to provide enough subnets that will hold the number of hosts we want to give them. The math behind our answers is not any more complex than what we've encountered already. While we determine the number of hosts in a network or subnet by subtracting the prefix length from 32, we determine the number of subnets that are possible by subtracting the total allocation from our final prefix length. For example, we can divide a /24 network into 4 unique networks with a prefix of /26 (since 26 - 24 = 2 and 2^2 = 4).

As you gain practice with base-2, you'll find that you're often able to rely on mental math. You may still have trouble recalling slightly larger numbers without a calculator or pen/paper. As we did previously, we can often simplify our math (to complete it without external aids) by focusing on an octet at a time.

Let's apply this concept to estimate the number of addresses contained within /13 prefix. Rather than computing 2^19 directly, let's break the math down into smaller pieces. We know that we have two octets that contribute 256 addresses each and final octet contributing 8 (2^3) possible values. The total number of addresses is the product of all three values. While I would probably lean on a calculator to determine the exact number of hosts, I can see that the answer is roughly half a million hosts without breaking too much of a sweat. It's rare that I need much more accuracy when I'm dealing with numbers this large.

A NOTE ABOUT IPV6

When we turn our attention to IPv6 with its 128-bit address space, you'll realize that counting hosts is a thing of the past when we are guided to create prefixes that leave 2^64 possible addresses. The previous skills are still useful, however, to determine how many subnets we can create from a given allocation.

IPv6 allocations tend to live in the /60 to /48 range, leaving 68 to 80 bits of address space for each organization or individual to work with. IPv6 address space is so plentiful that even an individual can expect to receive a prefix in this ballpark. The magnitude of these numbers, however, does not impact the factors that lead us to split our address allocations into subnets.

Our main concern under IPv6 (when it comes to addressing) becomes determining how many /64 networks we can create from a larger allocation and whether we might be able to add an additional level of hierarchy to our address groupings. It should be clear that this system provides quite a bit of flexibility to IPv6 users. A home user obtaining a /60 prefix from an ISP can create 16 (2^4) subnets to organize devices living in the network.

A small business might receive a /56 prefix from an ISP, enabling them to design a LAN with 256 (2^8) subnets.

On an even larger scale, a large business may purchase a permanent allocation in the /48 prefix range. This allocation provides 65,536 (2^16) subnets and raises the question of further partitioning the address space according to geographical region or site by allocating part of the prefix for this purpose.

If we follow this pattern even further, we arrive at the /32 prefix that is allocated to large service providers. These service providers are interested in further delegating their space address space to their customers. Given a single /32 allocation, a ISP can divide the address space into more than 268 million /60 prefixes that it dispenses to residential customers.

FINDING PREFIXES

Despite becoming more fluent in translating between network prefixes and address masks or computing the size of a network, it's still challenging to for most of us to determine whether a packet is in a specific subnet when prefixes look alike, but don't align on 8-bit boundaries. For example, how would you determine whether 192.168.22.7 and 192.168.22.8 are valid members of the same network segment given a prefix length of 29? Likewise, how would you decide whether 172.24.0.1/12 is a valid network configuration.

The hard way to solve these problems is to convert each address to binary and compare to the appropriate subnet mask. This process is not one you want to be doing on a regular basis. Instead, let's look at how we can apply numeric shortcuts to identify valid prefixes along with their upper and lower bounds. We're including the table from the previous section to aid you in the process.

PREFIX LEN	1	2	3	4	5	6	7	8
2^N	128	64	32	16	8	4	2	1
MASK (DEC)	128	192	224	240	248	252	254	255
MASK (HEX)	80	C0	E0	F0	F8	FC	FE	FF

Whether you are trying to determine the prefix for an address given a prefix length or comparing two address for routing decisions, it's helpful to consider the address in terms of fully-masked octets versus partially-masked octets and unmasked octets. Given a fully masked octet in an address, we can copy the octet directly into the prefix. If we're comparing two addresses, we can infer that they must match exactly to communicate over IP in the same Layer 2 network segment. Unmasked octets are also easy, since they don't contribute at all to the prefix.

For partially-masked octets, we do have to make some simple calculations. Begin by identifying the power of two corresponding to the final masked bit. Every valid network prefix uses a multiple of that value in this octet. In an octet with four masked bits, the prefix must be a multiple of 16. In an octet with one masked bit, we know that can identify a pair of valid subnets. Since 2^7 is 128, our candidates are 0 and 128 for the partially-masked octet.

Let's demonstrate this within context by dividing the 172.0.0.0 network into /12 allocations. Since we mask four bits in the second octet, we know that the prefix at that octet should always be a multiple of 16. In other words, we could use 172.0.0.0/12, 172.16.0.0/12, or 172.240.0.0/12 as legal network prefixes. 172.24.0.0/12 is invalid.

How do we use this to compare addresses? Knowing that each range includes all addresses between its network ID and the next prefix, we find that 172.48.0.0/12 would include everything from 172.48.0.0 and 172.63.255.255.[4] If we can determine the range for one prefix, we can quickly identify whether other addresses are located on the same subnet.

WRAPPING UP

A network device can't send traffic to other hosts (local or remote), unless it understands the relationship between its address and network prefix. Whether we automate address assignment or do it manually, we need to configure both values. Some systems will break the requirement into three parts: host address, network address, and subnet mask. Some will compute the mask directly from a /length suffix on the network address. Others will derive everything from the host address given a /length. It's important to be comfortable working with the data in any form it is given to you.

Though we've focused on the structure of IP addresses, it's also worth taking an opportunity to review the rules for assigning addresses on a network. Devices that live on the same layer 2 network are configured with addresses in the same prefix. The definition of the layer 2 network can be hard to pin down, but it typically refers to devices connecting to the same port of a router,[5] regardless of the layers of switches and access points that are in that path.

Likewise, devices that are configured with addresses in the same prefix should be placed in the same layer 2 network. IP routing protocols depend on each network having a unique prefix, which is why LANs using private addressing rely on network address translation (NAT) for communication beyond the LAN.

[4] The first and last address in the range are reserved for special purposes

[5] Conveniently ignoring the impact of virtual segmentation, i.e., VLAN tagging, for simplicity