Converting Ranges into Prefixes for Routers

Scot Anderson, Ph.D.

October 11, 2021

1 Introduction

In networking, students are often taught to calculate the set of longest-matching prefixes found in a routing table given a set of ranges or subnets with corresponding ports. This may seem trivial, but it is complicated by the fact that we may give a range of addresses instead of subnets. The basic steps in the algorithm, when given ranges, are as follows:

1. Break each range up into legal subnets
2. Join adjacent subnets that can be combined into a single subnet when they match the same link interface until no more combining can be done.
3. If a link interface has more than one subnet, check if adjacent interface subnets would combine with it to create a single subnet encompassing both sets of subnets, call this A. If this is possible, replace the multiple subnets case (the one with the largest number of subnets) with A. Repeat until no more combinations can be made. Note once these two have been "combined" this way, they cannot participate in another combination.
4. List the prefixes, based on the subnets, in the list with their link interface as the final answer.

2 Algorithm Step 1: Breaking down the ranges into subnets

Given a range $[a, b]$, where $a, b \in \{x | x \textit{ matches } /\hat{}[01]\{32\}\$/\}$. Notice that the regular expression, /ˆ[01]{32}$/, matches a 32-bit string.

Step 1

The base case condition is when a and b share a common prefix p and $a = p||0*$[1] and $b = p||1*$. That is after the prefix, a ends in all zeros and b ends in all ones. Here p is the one and only prefix needed to represent this range. Output p and return.

Case 2

If Case 1 does not hold, then the following must hold $a = p||0||a_{end}*$ and $b = p||1||b_{end}*$ where a_{end}, b_{end} are the strings of binary digits after the 0 and 1 respectively. Break this range up as follows:

Range 1: $[a_1, b_1] = [a, p||0||1*]$

Range 2: $[a_2, b_2] = [p||1||0*, b]$

These ranges are contiguous and together represent the original range.

Run this algorithm on each range separately.

Example 2.1. Given the table of ranges below and the links that they should exit, translate this into a longest-prefix-match routing table.

Destination Address Range	Link Interface
1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 1111 1111	0
1100 0000 1010 1000 0000 1011 0000 0000 1100 0000 1010 1000 0000 1011 0000 1111	1
1100 0000 1010 1000 0000 1011 0001 0000 1100 0000 1010 1000 0000 1011 0001 1111	2
Otherwise	3

[1]The symbol || is commonly used for concatenation of strings. $0*$ or $[01]*$ means that the value 0 or (o or 1) can be repeated many times.

Consider the base case where we have some prefix in common between the start and end of the range, denoted $[start, end]$ If start matches "prefix0*" and end matches "prefix1*", then the prefix for this range is "prefix". Ranges for Link interfaces 1 and 2 satisfy this condition, and we have a partially converted tables below.

Destination Address Range	Link Interface
1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 1111 1111	0
1100 0000 1010 1000 0000 1011 0000	1
1100 0000 1010 1000 0000 1011 0001	2
Otherwise	3

We are going to split this range into two ranges at the first difference. Note: The first difference will always have a 0 in start and a 1 in end (highlighted below in yellow). The first range will always start with 0 and the second range will always start with 1. This makes two contiguous ranges and covers all of the original.

Original	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 1111 1111
First Range	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0111 1111
Second Range	1100 0000 1010 1000 0000 1010 1000 0000 1100 0000 1010 1000 0000 1010 1111 1111

In both cases we have extended our matching prefix by 1 bit (0 in the first range and 1 in the second range). The first range will need to be divided again, but the second range matches our base case and we can put out a prefix of: **1100 0000 1010 1000 0000 1010 1**

Dividing up the first range will give us:

Original	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0111 1111
Second Range	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0011 1111
Second Range	1100 0000 1010 1000 0000 1010 0100 0000 1100 0000 1010 1000 0000 1010 0111 1111

Again the second range matches our base condition and we get a prefix of: **1100 0000 1010 1000 0000 1010 01**. Breaking up the first one again gives us:

Original	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0011 1111
First Range	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0001 1111
First Range	1100 0000 1010 1000 0000 1010 0010 0000 1100 0000 1010 1000 0000 1010 0011 1111

Again the second range matches our base condition and we get a prefix of: **1100 0000 1010 1000 0000 1010 001**. Breaking up the first one again gives us:

Original	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0001 1111
First Range	1100 0000 1010 1000 0000 1010 0000 1100 1100 0000 1010 1000 0000 1010 0000 1111
Original	1100 0000 1010 1000 0000 1010 0001 0000 1100 0000 1010 1000 0000 1010 0001 1111

Our first range finally resolved to the base case condition. So the prefix is: **1100 0000 1010 1000 0000 1010 0000 11**
Again the second range also matches our base case condition and we get a prefix of: **1100 0000 1010 1000 0000 1010 0001**
Our set of prefixes looks like this:

Destination Address Range	Link Interface
1100 0000 1010 1000 0000 1010 1	0
1100 0000 1010 1000 0000 1010 01	
1100 0000 1010 1000 0000 1010 001	
1100 0000 1010 1000 0000 1010 0001	
1100 0000 1010 1000 0000 1010 0000 11	
1100 0000 1010 1000 0000 1011 0000	1
1100 0000 1010 1000 0000 1011 0001	2
Otherwise	3

3 Algorithm Step 2: Joining adjacent subnets ... or Reverse subnetting

When we learned subnetting, we broke a range into two halves. When we are aggregating routes we do the reverse and combine two adjacent prefixes that can be combined into a subnet with a shorter mask. In this step we only aggregate routes that are using the same port.

If there was only one range assigned to a port, you should satisfy yourself that there can be no route aggregation for the port. I.e. if you could combine them here, we would not have split them in the previous step.

If there were multiple ranges assigned to a port (very possible if we are routing to another organization), then we may be able to aggregate routes similar to how we broke them up in the first step.

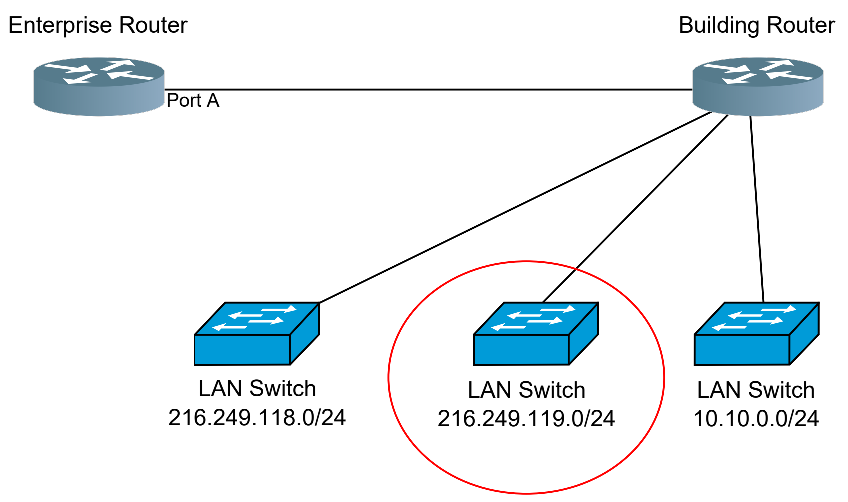

Figure 1: Enterprise Routes

Example 3.1. Suppose at some time in the past we allocated 216.249.118.0/24 and 10.10.0.0/24 to a building router from our enterprise router. The building router then routes those prefixes to switches (Local Area Networks). Figure 1 shows an added subnet to the Building Router shown in the red circle. At the enterprise router level, we now have two route prefixes both going to the Building router:

Prefixes	Link Interface
1101 1000 1111 1001 0111 0110	A
1101 1000 1111 1001 0111 0111	A

These prefixes are identical until the last bit, hence they are contiguous ranges, the same size and can be combined to have the same prefix. Namely:

Prefixes	Link Interface
1101 1000 1111 1001 0111 011	A

4 Algorithm Step 3: Subnet Aggregation to achieve minimal prefixes using Longest Prefix

In some cases we may not immediately see a way to join two prefixes because they point to different ports. Consider Example 2.1. Every prefix starts with 192.168.10 and is broken somewhere in the last byte. Consider the tree of these shown in Figure 2. Everything but the two blue subnets are assigned to Port 0. It would be awfully nice if we could somehow reduce the number of prefixes going to Port 0... and we can!

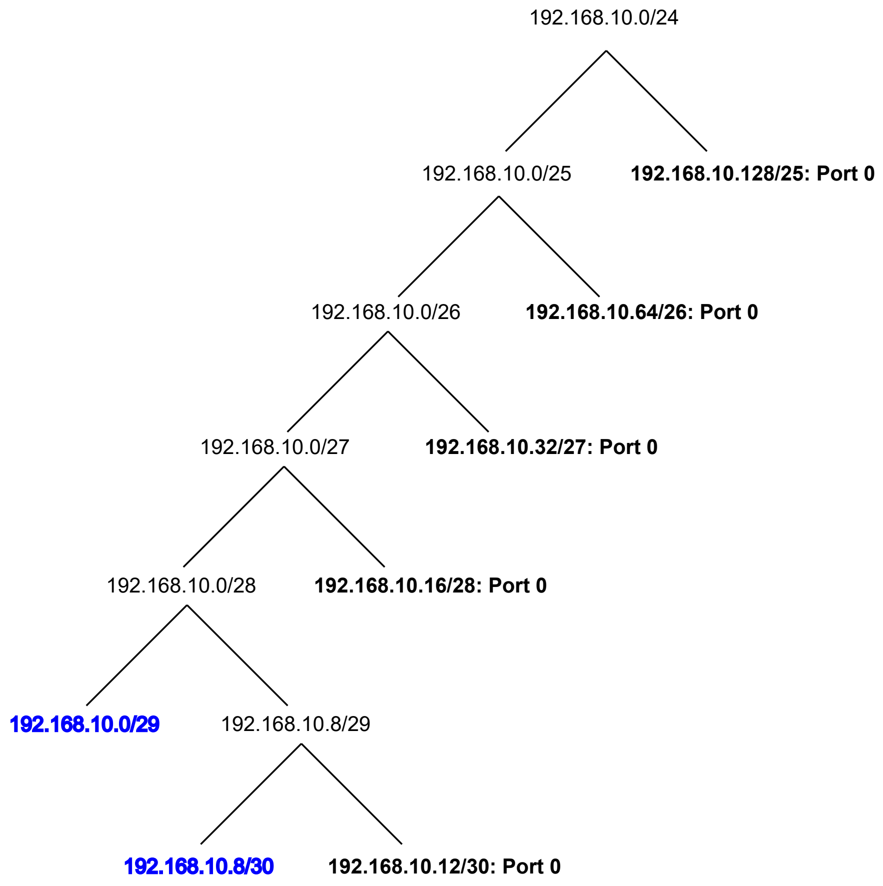

Figure 2: Prefixes as subnet tree.

Logic dictates that there are only two possibilities. (1) The ISP who assigned me IPs has retained these subnets or, (2) they belong to our network and we have not assigned them. In the first case, it would make sense to route these to our ISP out port 3. If I am holding them for some network that will soon be assigned, I could just route these subnets to /dev/null (in Linux parlance). Suppose that these two subnets are assigned to a new Link interface called port 4. We could use the longest matching prefix aggregate our subnets as follows:

Destination Address Range	Link Interface
1100 0000 1010 1000 0000 1010	0
1100 0000 1010 1000 0000 1011 0000	1
1100 0000 1010 1000 0000 1011 0001	2
Otherwise	3
1100 0000 1010 1000 0000 1011 0000 0 1100 0000 1010 1000 0000 1011 0000 10	4

5 Summary

We have shown an algorithm that is capable of creating minimal, longest prefix matching routing tables. The first step has been programmed by your professor and is available on github.