Subnet Mask Conversion

Slash Notation to Dot-Decimal Notation

By Jason Holloway

This exercise may be largely unnecessary, but I thought it was good fun. Realistically, using a hash table of conversions between slash notation to dot-decimal notation or converting to binary as an intermediary step would be more efficient. Nevertheless, if you want a mathematical approach, keep reading.

Subnet masks are typically expressed in dot-decimal notation (e.g., 255.255.255.252 and 255.255.240.0). They also can be represented in slash notation (e.g., /30 and /20). How can we convert the /30 slash notation to the 255.255.255.252 dot decimal notation?

Subnet masks consist of 4 octets, each representing a byte. They are ordered largest to smallest from left to right. We can find the number of 255 value octets by dividing the slash notation routing prefix by 8.

/30 -> 30 / 8 = 3 and 6 remainder.

/20 -> 20 / 8 = 2 and 4 remainder.

/30 divided 8 will give 3x 255 octets (e.g., **255.255.255**.*y*) and /20 divided by 8 will give 2x 255 octets (e.g., **255.255.240**.*y*).

To calculate the immediate octet (represented as *y* in the above calculations), two formulas can be used.

$$\sum_{\substack{n=1 \\ n=n\cdot 2}}^{2^{s \bmod 8 - 1}} \left(\frac{128}{n}\right)$$

$$\int_0^{s \bmod 8} (\ln 2 \cdot 2^{8-x})\, dx$$

The variable *s* represents the routing prefix. For example, /30 would be *s=30* and /20 would be *s=20*. Using either formula should produce 252 for /30 and 240 for /20. Once the immediate octet has been calculated, any remaining octets can be set to 0. Putting everything together will create 255.255.255.252 for /30 and 255.255.240.0 for /20.

The following python code completes the conversion.

```
def __slash_equation__(x:float)->float:
    return math.log(2) * (2 ** (8 - (x % 8)))

def get_dotted_notation(slash_notation: str)->str:
    try:
        v = int(slash_notation[1:])
        if v < 0 or v > 32: return None
        q = int(v / 8)
        lst = ["255" for i in range(q)]
        if len(lst) == 4:
            dotted = '.'.join(lst)
            return dotted
        else:
            # formula 1
            # res = 0
            # d = 1
            # while(d < 2 ** r):
            #     res += 128 // d
            #     d *= 2

            # formula 2
            res, err = integrate.quad(__slash_equation__, 0, v % 8)
            lst.append(str(round(res)))
            dotted = '.'.join(lst)
            dotted += (4 - len(lst)) * '.0'
            return dotted
    except: return None
```