IPV4 地址空间测绘

刘艺璇[1)] 许晨晔[2)] 徐东亿[3)]

北京市清华大学计算机科学与技术系

摘 要 使用了 ping IP 地址的方式，获取了中国部分省市 IP 地址段的存活情况，计算了地址块的活跃度，并以此为依据绘制了易于观测的热力图和希尔伯特曲线，从中发现了活跃 IP 的分布规律以及活跃 IP 的分布比例。

关键词 IPV4；地址块；活跃度；热力图；希尔伯特曲线

IPV4 Address Space Mapping

Yixuan Liu[1)] Chenye Xu[2)] Dongyi Xu[3)]

Department of Computer Science and Technology, Tsinghua University, Beijing

Abstract Using the way of ping IP address, we obtained the survival status of some IP addresses in some provinces and cities in China, calculated the activity of some address blocks and plotted thermodynamic diagrams and Hilbert curves which are easy to observe.

Key words IPv4; Address block; Activity; Thermodynamic diagram; Hibert Curve

1 引言

在地理空间当中，高速公路连接着一座座城市，道路交通安全法约束着我们在高速公路当中的行车规范，而各种精度和覆盖度的地图作为重要的地理位置信息资源被广泛的应用在方方面面。

同样，在网络空间当中，互联网也被称之为信息高速公路，网络安全法约束着我们在信息高速公路当中的行为规范，但却长期缺少一份能够在不同级别、不同颗粒度刻画网络空间的地图这一具有战略意义的产品存在。因此，我们进行了全国范围内的 ipv4 地址空间的测绘，希望能对网络空间地图能得到更直观更深入的了解。

1.1 背景

IPv4，是互联网协议（Internet Protocol，IP）的第四版，也是第一个被广泛使用，构成现今互联网技术的基础的协议。

在如今 IPV4 地址空间饱和的情况下，我们可以就 IPV4 地址的空间分布进行调查分析，并对其存活情况、地址块活跃度进行统计。

我们将 IP 地址的存活情况做了可视化，其中热力图可以很清晰的反应存活 IP 地址和地理位置的关系，同时可以对比各地区活跃 IP 的数目；希尔伯特曲线一种能填充满一个平面正方形的分形曲线（空间填充曲线），它可以将连续的一维地址映射到二维空间，从而更清晰地展示出一些连续 ip 地址块的活跃程度。

1.2 解决思路

使用了 ping IP 地址的方式，获取了中国部分省市 IP 地址段的存活情况，ping 通（并得到往返时间）表示该地址存活，ping 不通（即超时）表示不存活。在此基础上我们定义地址块的活跃度为“./24 地址块的活跃地址数量/256”，并计算出一些地址块的活跃度。根据得到的这些数据，我们进行了两个方面的可视化：一方面我们绘制了中国地区部分省市的热力图；另一方面我们得到数学上的希尔伯特曲线，从而可视化地表现连续地址块的活跃情况。

2 具体实现

2.1 获取 ip 地址的活跃度

活跃度的定义为“./24 地址块的活跃地址数量/256”，我们通过 ping IP 地址的方式获取 IP 地

址段的存活情况，进而获取到地址块的活跃度。下面以天津为例，介绍我们的具体步骤。

2.1.1 获取天津所有的 ip 地址段，共 250 万个左右。

117.8.0.0	117.15.255.255	60.24.0.0	60.27.255.255
221.196.0.0	221.197.255.255	180.212.0.0	180.213.255.255
42.80.0.0	42.81.255.255	182.238.0.0	182.238.255.255
221.198.0.0	221.198.255.255	61.181.0.0	61.181.255.255
42.122.0.0	42.122.255.255	218.67.128.0	218.67.255.255
202.113.128.0	202.113.191.255	61.136.0.0	61.136.63.255
202.113.192.0	202.113.223.255	202.99.64.0	202.99.95.255
219.150.96.0	219.150.111.255	202.99.112.0	202.99.127.255
202.113.16.0	202.113.31.255	202.113.0.0	202.113.15.255
103.22.120.0	103.22.123.255	103.3.96.0	103.3.99.255

125.36.0.0	125.39.255.255	123.150.0.0	123.151.255.255
60.28.0.0	60.29.255.255	218.68.0.0	218.69.255.255
219.226.0.0	219.226.255.255	221.129.0.0	221.129.255.255
221.238.0.0	221.238.255.255	60.30.0.0	60.30.255.255
221.239.0.0	221.239.127.255	125.35.128.0	125.35.255.255
202.113.64.0	202.113.127.255	219.150.64.0	219.150.95.255
202.113.32.0	202.113.63.255	219.150.32.0	219.150.63.255
202.164.0.0	202.164.15.255	202.113.224.0	202.113.239.255
202.99.96.0	202.99.103.255	202.99.104.0	202.99.111.255
103.1.20.0	103.1.23.255		

2.1.2 对天津所有的 ip 地址执行 ping 命令，获取地址相关信息。

我们主要通过 ping 的命令来获取每一个 IP 地址是否能 ping 通，如果 ping 不通则认为地址不存活，如果 ping 的通的话，则认为地址存活，并记录相关的一些信息，主要是记录返回的时间以及 ttl。然而部分地址由于内部原因无法被 ping 通，所以存在一定误差。

在 ping IP 时，最开始的时候采用的是单线程 ping，但是速度很慢，主要由于天津 IP 很多但是无法 ping 通的也很多，大部分都是超时返回，导致 ping 一次的时间很长，所以我们采用了多线程，同时开了 150 个线程，并降低了 ping 的超时时间，而且由于我们更看重的是 IP 能否 ping 通，而不是 ping 通的时间，所以我们也减少了发送包的数目，最后达到的处理速度为 2s 处理一个/24 地址块。

```
class ThreadUrl(threading.Thread):
    def __init__(self,q,output_name):
        threading.Thread.__init__(self)
        self.q=q
        self.f_name = output_name


    def run(self):
        while True:
            host=self.q.get()
            ret=subprocess.call('ping -c 2 -w 2 '+host,shell=True,stdout=open(self.f_name,'a'))
            self.q.task_done()
```

2.1.3 将数据存入数据库

为了方便查询以及之后的数据处理，我们使用了 MySQL 来存储数据，使用 python 的 pymysql 库来操作数据库，数据库的每一列依次为 IP 地址（string），能否 ping 通（bool），TTL（int）以及 ping 的平均时间（double），同时为了加快存储速度，我们采用了多线程来进行数据库的操作。

```
class Run_sp(threading.Thread):
    def __init__(self,thread_num,ip_list):
        threading.Thread.__init__(self)
        self.thread_num=thread_num
        self.ip_list=ip_list

    def run(self):
        db = MySQLdb.connect("localhost", "root", "123456", "ip_db")
        cursor = db.cursor()
        num=0
        for host in self.ip_list:
            num += 1
            if host[1] == 0:
                sql = "INSERT INTO ip_tianjin(ip_addreess, is_conn) VALUES('%s', '%d')" % (host[0], host[1])
            else:
                sql = "INSERT INTO ip_tianjin VALUES('%s', '%d', '%d', '%d')" % (host[0], host[1], host[2], host[3])
            cursor.execute(sql)
            db.commit()

            while 1:
                flag= db_cursor.nextset()
                if flag!=1:
                    break
                db_cursor.fetchall()


            if num%1000==0:
                    print "Thread %d execute %d hosts"%(self.thread_num,num)
        print "thread %d end at [%s]"%(self.thread_num,time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time())))
        cursor.close()
        db.close()
```

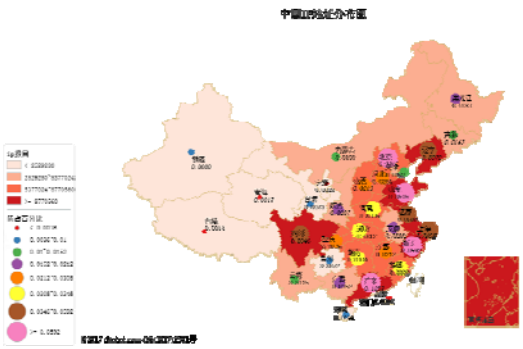
2.1.4 计算活跃度

若需计算某/24 地址块的活跃度，则只需要在数据库中查询该地址块 IP 的信息，计算出可以 ping 通的数目，再除以 256 即可。

2.2 热力图

2.2.1 获取中国 ip 地址分布

最开始我们的目标是绘制全中国的所有 ip 的活跃度。我们从网上收集资料获取了中国各省市的 ip 地址分布，各省市 ip 地址数量分布如下图所示。



由于时间有限，我们最后仅获取了天津，新疆，西藏，青海，宁夏区域的 ip 地址的活跃度。

2.2.2 从数据库中读取 ip

从数据库中读取活跃 ip（即 ping 的通的 ip），进行后续操作。

2.2.3 将 ip 转换为经纬度

在这里采取百度的 api，传入 ak（百度账户密钥）和 ip 到特定网址会返回一个形如下图所示的 json 串，从中可以获取到该 ip 的经纬度信息。

```
address:  "CN|西藏|None|None|CHINANET|0|0"
content:
  address:  "西藏自治区"
  address_detail:
    city:  ""
    city_code:  13
    district:  ""
    province:  "西藏自治区"
    street:  ""
    street_number:  ""
  point:
    x:  "89.13798168"
    y:  "31.36731540"
status:  0
```


该段算法的代码如下所示。由于一次请求百度 api1 个 ip 的经纬度所需时间较长，在这里我使用 grequests 库，可以多线程并发的进行 request 请求，在这里我设置了 page=20，即每次同时进行 20 次请求，这样使得程序运行速度大大加快。

```
page = 20
l = len(ips)
for i in range(0, l, page):
        urls = []
        for j in range(page):
            urls.append("http://api.map.baidu.com/location/ip?ak=" +
ak + "&ip=" + ips[i+j] + "&coor=bd09ll")
        rs = (grequests.get(u) for u in urls)
        address_res = grequests.map(rs, size=page)
```

2.2.4 获取热力图

在对所有测得的活跃 ip 得到的经纬度信息及其出现次数记录完毕之后，调用百度地图的 api，获得最终的热力图。

2.3 希尔伯特曲线图

使用希尔伯特可以将一维的连续 ip 地址转换成二维的表现形式，其转换方式如下：

```
//Hilbert代码到XY坐标转换
void d2xy(int n, int d, int *x, int *y) {
    int rx, ry, s, t = d;
    *x = *y = 0;
    for (s = 1; s < n; s *= 2) {
        rx = 1 & (t / 2);
        ry = 1 & (t ^ rx);
        rot(s, x, y, rx, ry);
        *x += s * rx;
        *y += s * ry;
        t /= 4;
    }
}
```

```
//XY坐标到Hilbert代码转换
int xy2d (int n, int x, int y) {
    int rx, ry, s, d = 0;
    for (s = n/2; s > 0; s /= 2) {
        rx = (x & s) > 0;
        ry = (y & s) > 0;
        d += s * s * ((3 * rx) ^ ry);
        rot (s, &x, &y, rx, ry);
    }
    return d;
}

void rot(int n, int *x, int *y, int rx, int ry) {
    if (ry == 0) {
        if (rx == 1) {
            *x = n - 1 - *x;
            *y = n - 1 - *y;
        }
        int t = *x;
        *x = *y;
        *y = t;
    }
}
```

在得到了一些地址块的活跃度之后，我们将连续的 256 个连续的地址块整合在一起来绘制 16*16 的希尔伯特曲线图，选取蓝色系颜色（RGB=(x, 255, 255)）来填充单位块，将活跃度映射成颜色值，颜色越深（x 越小）表示活跃度越高。

为了得到一个更多活跃度信息数量的可视化表现形式，我们绘制了“二维”希尔伯特曲线，选取了 16*256 个地址块，即一共需要 ping16*256*256 个地址。整体曲线由 4*4 个“一维”希尔伯特曲线构成，即每一个块都是一个独立的 256 个地址块的希尔伯特曲线图。这样的一个“二维”曲线图可以表示更大范围、更多数量的 ip 地址块的活跃度情况。

3 主要成果

3.1 热力图

最终得到的热力图如下所示。

从图中可以得到 3 个结论：

（1）一般而言，一个省的大多数 ip 都分布在其省会城市和省会之外的主要城市。图中这点表现的非常突出，除了因为实际情况如此之外，还有一点原因是百度根据 ip 获取经纬度的精度比较粗糙，一般只会定位到较大的城市。

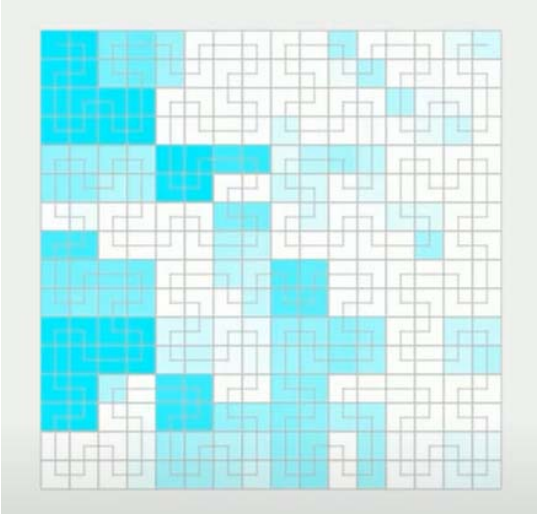
（2）尽管天津的 ip 数量非常多（约为西藏整个省的 7 倍），但实际的结果而言天津的活跃 ip 和西藏的活跃 ip 数量却差不多。这个现象的原因不明，推测可能是天津位于工业区，很多 ip 段都设置了防火墙禁止 ping，使得活跃 ip 占总数的比例非常低。

（3）在图中不太明显，但是在结果中有部分 ip 被定位到了云南、北京等地，尽管我们并没有 ping 这些地区的 ip。推测这一现象出现的原因有两点：1 是百度的根据 ip 获取经纬度的 api 有误，返回了错误的结果；2 是我们找到的各省份的 ip 段本身就有错误，由此导致了错误的结果。

3.2 希尔伯特曲线

3.2.1 希尔伯特曲线

从数据库中获取所需 ip 段的地址块的活跃度，按 2.3 中方法绘制希尔伯特曲线，填充颜色以及相邻块连线后的某个希尔伯特曲线图如下：

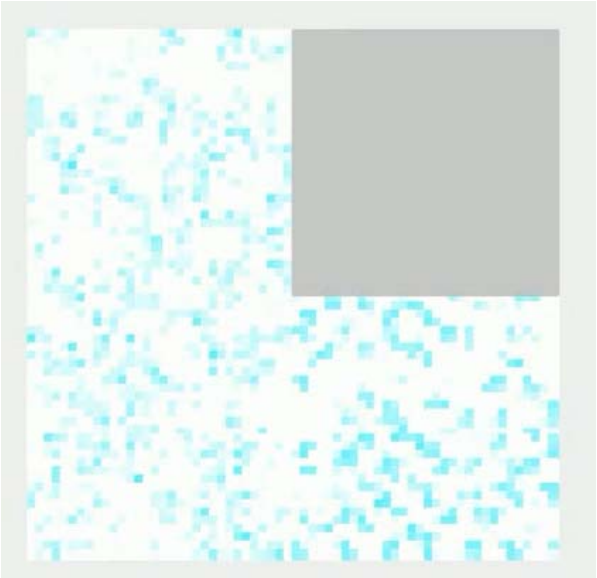


希尔伯特曲线中，一维上相邻的地址块在曲线中也是相邻的，相邻的地址块大体上聚集在一起，从希尔伯特曲线图中可以看出活跃/不活跃的地址块相对聚集在一起，在一维上大体也是连续的。

在比较了几个不同省市所选取的地址块的希尔伯特曲线之后，我们发现天津的 ip 地址块整体活跃度其实低于西藏、新疆，有可能天津由于各种原因可 ping 通的 ip 较少。

3.2.2 “二维”希尔伯特曲线

在实际实现过程中，由于时间原因完成 ping 工作的 ip 地址块数量有限，达不到 16*256 个，只有 12*256 个地址块因此最后绘制出的曲线并不完整，灰色为无数据区域。



4 结语

通过此次的课题研究，我们了解并掌握了一些 ip 地址空间测绘的方法和途径，并基于此完成了一套以 ip 地址块活跃度为核心的测绘方案。

经历了数周的艰苦奋斗，我们最终完成了本次实验。尽管过程艰辛，但我们也从中受益颇多。我们在这次实验中我们学到了以下几点：

(1) 实验要趁早。尽管我们很早就确立了思路，但真正开始动手的时间却比较晚，并且在实验过程中还遇到了很多预想不到的困难，花费了大量的时间才解决，最终导致没能收集到完整的实验数据，没能 ping 完全国所有的 ip，只得到了部分省市的 ip 活跃度分布。如果能尽早开始实验，就能获取更多的实验数据，也有更充足的时间进行数据分析，本次实验就能做的更好。

(2) 在处理这种大型的可并发的任务时，多线程技术非常重要。在实验初期，我们单线程的 ping 所有 ip，速度非常慢，极大的拖累了实验进度。在我们采用了多线程技术后，无论是 ping ip 还是根据百度 api 将 ip 转换为经纬度的速度都比原来要快了十几倍，让我们在有限的时间里得到了一定的成果。

(3) 数据分析要充分。在看了其他组的实验展示之后，我们才意识到，尽管我们获取到了大量的数据，但对这些数据的研究和分析并不够充分。获取数据固然重要，但分析数据也非常重要。本次实验由于时间有限未能将这点做的很好， 但在未来的学习和工作中我们会吸取这次的教训做的更好。

最后感谢王继龙老师深入浅出的理论指导，各位助教对我们问题的耐心解答和帮助!