被动网络测量

网络运营商查看不同类型的网络流量数据以了解其网络的属性。 一些网络数据可以在网络设备（例如路由器、交换机）转发实时流量时直接收集。 收集此数据不会影响网络行为，因此称为“被动”（与“主动”测量相对）。

在本次实验中，您将分析两种类型的被动网络测量数据——流量和 BGP 路由。

该实验有几个部分。 每个部分都包含标有您需要完成的 TODO 的部分。

背景

使用 IPFIX 进行流量测量

大多数网络中的路由器使用 IPFIX 协议 收集流量测量结果。 NetFlow 是 Cisco 定义的 IPFIX 专有形式，在网络社区中众所周知，因为 Cisco 为许多大型网络提供路由器。

在实验的这一部分中，您将分析从将普林斯顿校园网络连接到互联网的路由器捕获的 NetFlow 记录跟踪。 该作业将要求您执行与网络运营商执行的类似分析 - 询问有关普林斯顿校园中最流行的流量端点、最流行的 Web 应用程序等的问题。 （正如你可以想象的，当我们开始考虑安全性时，分析这些基线的能力将会派上用场！）

流记录位于文件夹中的“netflow.csv”文件中。 为了简化分析，我们确保普林斯顿校园网的IP地址以128.112开头，并将其低16位匿名化，以保护校园网用户的隐私。 为了进一步简化您的任务，我们已将这些记录解析为 CSV（逗号分隔变量）格式，字段名称列在文件的第一行中。 （在真实网络中，路由器将 IPFIX 记录导出为二进制文件。）

使用 BGP 路由表测量域间路由

为了帮助网络运营商了解 Internet 路由的状态，许多路由器都能够定期将 BGP 路由表“转储”到静态文件中。 这些路由表包含有关每个 IP 前缀、路由器为每个前缀学习的所有 BGP 路由以及路由器最终选择的“最佳”BGP 路由的信息。 分析 BGP 路由表可以提供有关不同 IP 前缀的流量的目的地信息。

在本次实验中，我们为您提供了来自名为 RouteViews 的项目的路由表转储。 您可以访问该站点以了解有关他们收集的路由表的更多信息。 如果您想使用实时路由表视图，“telnet route-views2.routeviews.org”还将在 Routeviews 项目的真实 BGP 路由器上为您提供命令行提示。 http://routeviews.org/bgpdata/ 有定期的二进制路由表转储和来自参与路由器的更新日志。 同样，对于此实验，我们已将二进制路由表转储解析为更易于直接分析的格式。 数据位于“bgp_rib.csv”文件中。

使用map()和reduce()进行功能数据分析

本实验中的几个数据分析步骤使用“MapReduce”编程模型。 MapReduce 起源于函数式编程语言，涉及使用两个函数（称为`map()`和`reduce()`）将函数应用于可迭代数据（如链表、数组等）。

map()

通用的`map()`函数有两个参数：另一个函数（它本身带有一个参数）和一个可迭代对象。 然后，`map()`将参数函数应用（映射）到可迭代对象中的每个项。 更多有关详细信息，请参阅 Python 内置`map()`函数的文档：https://docs.python.org/2/library/functions.html#map。 以下示例使用`map()`将 3 添加到列表的每个元素

```
some_numbers = [1,2,3]
three_more = map(lambda x: x+3, some_numbers)
print three_more
```

```
[4, 5, 6]
```

`map()` 通常与匿名函数（上例中的 `lambda`）一起使用，但也可以像普通函数一样轻松地使用：

```
def add3(i):
    return i+3

some_numbers = [1,2,3]
three_more = map(add3, some_numbers)
print three_more
```

请注意，`map()` 的实际实现允许映射函数采用多个参数或在闭包中包含更多信息，但这对于此分配来说不是必需的。

reduce()

通用的 `reduce()` 函数接受另一个函数（它本身接受*两个*值）、一个可迭代对象和一个可选的初始值设定项值。 参数函数应用于可迭代对象中的前两个元素（或第一个元素和初始值设定项）以获取返回值。 然后将该函数应用于此返回值和可迭代对象中的下一项。 这将继续遍历可迭代对象，直到只剩下一个返回值。 这允许 `reduce()` 计算可迭代对象中所有数据的摘要。 有关更多详细信息，请参阅 Python 内置 `reduce()` 函数的文档：https://docs.python.org/2/library/functions.html#reduce。 以下示例使用 `reduce()` 来计算整数列表中 4 的数量：

```
def count_4s(count, i):
    # The order of the arguments matters.
    #     The first argument is the accumulated value
    #     The secod argument is next value from the iterable
    if i == 4:
        return count + 1
    else:
        return count

some_numbers = [1,4,0,1,4]
num_fours = reduce(count_4s, some_numbers, 0) # 0 is the initializer value
print num_fours
```

同样，`reduce()` 的实际实现允许归约函数接受两个以上的参数或在闭包中包含更多信息，但这对于此分配来说不是必需的。

MapReduce 很受欢迎，因为它允许轻松并行化大型数据集上的分析任务。 尽管有许多开源和专有的 MapReduce 风格的数据处理库（通常使用不同的方式来表达可迭代数据集和在多台计算机上分配任务），但它们都涉及 `map()` 和 `reduce()` 函数，就像您将看到的那样在本次作业中使用。 如果您有兴趣了解有关数据分析的函数式编程的更多信息，您可以学习 COS 326。

Part A: IPFIX 数据

解析 IPFIX 数据

`netflow.csv` 文件包含来自普林斯顿校园网络边界路由器的预处理 netflow 数据。 数据是“未采样的”，即它为经过边界路由器上任何接口的每个数据包编译流统计数据。 我们使用 `nfdump` 工具来处理路由器收集的原始 NetFlow 数据。 `netflow.csv` 文件的每一行（顶部的标头除外）都会记录流的以下信息：

```
Date first seen, Time first seen (m:s), Date last seen, Time last seen (m:s),
Duration (s), Protocol,
Src IP addr, Src port, Dst IP addr, Dst port, Packets, Bytes, Flags, Input
interface, Output interface
```

为了分析这些数据，我们首先需要将其读入Python数据结构。 以下代码使用内置的“csv”库将“netflow.csv”读入字典列表。 如果您有兴趣，可以查看“csv”库文档：https://docs.python.org/2/library/csv.html

```
import csv

with open('netflow.csv', 'r') as netflow_file:
    netflow_reader = csv.DictReader(netflow_file)
    netflow_data = list(netflow_reader)

print "Number of flow records: {}".format(len(netflow_data))
print
print "Sample flow record: {}".format(netflow_data[0])
```

分析 IPFIX 数据

以下各节侧重于使用您准备的网络流数据回答特定问题。 这些问题既是真正的网络运营商感兴趣的，也可能揭示一些关于普林顿社区如何使用互联网的令人惊讶的事实。

普林斯顿网络用户最常访问的 IP 地址是什么？

为了回答这个问题，我们必须决定如何衡量 IP 地址的流行度。 所有流的总流量似乎是一个合理的选择，但无论流量大小，流向 IP 地址的总流量也是如此。 网络运营商实际上使用这两个指标（以及其他指标），我们也将在此处执行此操作。

第一步：根据流量数确定热门IP地址

完成以下代码以生成一个 Python 字典 `ips_by_flows`，其中包含 `netflow_data` 中每个外部（不是 128.112.*.*）IP 地址的流量总数。 字典的键应该是 IP 地址，值应该是整数流计数。

首先完成 `count_by_flows()` 函数，该函数应该采用上述形式的现有字典，并从 `current_flow` 中适当更新它。 如果您对数据类型感到困惑，请使用 print 语句来检查变量。

您可能需要创建一个额外的帮助函数来测试 IP 地址是否以 128.112 开头。 有用的字符串方法记录在此处：https://docs.python.org/2/library/stdtypes.html#string-methods。

然后，您将需要使用 `reduce()` 函数来构建字典结果。 作为提示，`reduce()` 的初始化参数应该是 `defaultdict(lambda: 0)`。 defaultdict() 函数创建一个字典，其默认值是参数函数的输出（在本例中，只需 0）。 这允许您增加特定键的值，而无需首先检查该键是否已在字典中（如果您使用“{}”而不是“defaultdict()”创建字典，这将引发 KeyError）。

提供的代码将打印并绘制最流行的 IP。 `check_ips_by_flows()` 函数会将您的答案的前 15 个最流行 IP 的 md5 哈希值与正确答案的 md5 哈希值进行比较。 这将打印一条消息，让您知道您是否正确或需要继续调试。

```
%matplotlib inline
from collections import defaultdict
from plotting import plot_flows
from testing import check_ips_by_flows

# TODO: complete count_by_flows function
def count_by_flows(counts, current_flow):
    # counts is the current dict result
    # current_flow being processed
    pass


# TODO: use reduce() function to apply count_by_flows to netflow_data and assign
the result to ips_by_flows


# print the top 5 IP addresses by number of flows
sorted_ips_by_flows = sorted(ips_by_flows.items(), reverse=True, key=lambda x:
x[1])
print "Most popular IP addresses by number of flows:
{}\n".format(sorted_ips_by_flows[0:5])

# check the results
check_ips_by_flows(sorted_ips_by_flows[0:15])

# plot the results
plot_flows(sorted_ips_by_flows)
```

步骤2：根据总量确定热门IP地址

完成以下代码以生成一个字典`ips_by_volume`，其中包含每个外部（非普林斯顿）IP 地址的字节总数。 字典的键应该是 IP 地址，值应该是整数字节计数。 请记住，`netflow_data`中的值是字符串。 您需要将它们转换为整数或浮点数才能进行算术运算。

```
%matplotlib inline
from plotting import plot_volumes
from testing import check_ips_by_volume

# TODO: complete count_by_volume function
def count_by_volume(counts, current_flow):
    # counts is the current dict result
    # current_flow being processed
    pass


# TODO: use reduce() function to apply count_by_volume to netflow_data and
assign the result to ips_by_volume


# print the top 5 IP addresses by volume
sorted_ips_by_volume = sorted(ips_by_volume.items(), reverse=True, key=lambda x:
x[1])
print "Most popular IP addresses by volume:
{}\n".format(sorted_ips_by_volume[0:5])

# check the results
check_ips_by_volume(sorted_ips_by_volume[0:15])
```

```
# plot the results
plot_volumes(sorted_ips_by_volume)
```

普林斯顿网络用户最流行的应用程序（按协议）是什么?

您认为普林斯顿网络上最常见的应用程序协议是什么？ 网络流量（HTTP 和 SSL）？ 安全远程连接 (SSH)？ 电子邮件（SMTP、IMAP、POP3）？ 在实践中，网络运营商可能希望识别流行的应用程序来制定供应计划或更改网络配置以不同地处理来自不同应用程序的流量（例如，在不同链路上路由流量）。 这可以防止大容量应用程序（例如视频流）中断关键的小容量应用程序的性能。

您可以通过在网络流数据中查找最流行的流量端口来回答这些问题。 许多应用程序协议对其流量使用众所周知的固定端口。 例如，HTTP 流量发生在端口 80 上，SSL 流量发生在端口 443 上，SSH 流量发生在端口 22 上，SMTP 流量发生在端口 25 上。

我们将再次使用流量数和总流量作为端口“受欢迎程度”的指标。

完成以下代码以根据 netflow 数据创建`ports_by_flows`和`ports_by_volume`字典。 使用与上面创建`ips_by_flows`和`ips_by_volume`相同的策略。

包括所有目标端口和源端口**低于 1024**（低于 1024 的端口是“众所周知的”并且可以轻松映射到应用程序）。

```
%matplotlib inline
from plotting import plot_ports
from testing import check_ports_by_flows, check_ports_by_volume

# TODO: create ports_by_flows and ports_by_volume dicts from netflow_data


# Print the most popular ports and check the results
sorted_ports_by_flows = sorted(ports_by_flows.items(), reverse=True, key=lambda
x: x[1])
sorted_ports_by_volume = sorted(ports_by_volume.items(), reverse=True,
key=lambda x: x[1])
print "Most popular ports by number of flows:
{}".format(sorted_ports_by_flows[0:5])
print "Most popular ports by volume: {}\n".format(sorted_ports_by_volume[0:5])
check_ports_by_flows(sorted_ports_by_flows[0:15])
check_ports_by_volume(sorted_ports_by_volume[0:15])

# plot the results
plot_ports(sorted_ports_by_flows, sorted_ports_by_volume)
```

问题

根据上述分析结果，回答下列问题。

- 问题1：按流量数量和流量计算，5 个最流行的外部（非普林斯顿）IP 地址是什么?
- 问题2：使用 Vagrant 终端中的“whois”命令来了解有关这些 IP 地址的信息（例如`whois 169.54.233.126`）。 从问题 1 的答案中选择 2 个地址，并写下您从中得到的内容，以及您认为它们是最受欢迎的原因。
- 问题3：按流量和流量计算，最受欢迎的 5 个端口是什么？ 哪些应用程序与它们相关？ 这里有一个固定端口的维基百科页面：https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers。 您还可以在线搜索以查找固定端口/应用程序映射。 您对哪些应用程序最受欢迎感到惊讶吗?
- 问题4：您认为为什么 telnet（端口 23）在普林斯顿网络中占据如此多的流量?

- 问题5：所提供的 NetFlow 数据是在大约上午 6:05 至 6:10 的 5 分钟内捕获的。 您认为捕获时间对最终最受欢迎的应用程序有多大影响？ 如果数据是在不同的 5 分钟窗口（您选择的）内捕获的，您希望看到什么变化？

Part B: 分析 BGP 路由表

到目前为止，我们已经研究了 IP 地址和端口级别的最高流量，但网络运营商可能也有兴趣探索哪些其他网络（即自治系统，autonomous systems，AS）负责发送或接收流量到网络。 从我们之前关于对等互连和互联网业务关系的讲座中，应该清楚为什么运营商可能关心了解哪些 AS 正在向其发送流量！此信息也可能有助于探索各种网络攻击（例如，拒绝服务攻击的来源），我们将在下一个实验中探讨这些攻击。

RouteViews项目允许网络运营商从互联网周围几个不同自治系统的角度获取有关全球路由系统的实时信息。 RouteViews服务器充当软件 BGP 路由器，通过 BGP 会话获取其 BGP 路由信息，就像任何其他路由器学习 BGP 路由一样。 RouteViews 服务器与其他使用 BGP 的路由器之间的主要区别在于 RouteViews 服务器不转发任何真实的 Internet 流量。

RouteViews 定期以称为 MRT 的二进制格式记录 BGP 路由表（有时称为路由信息库或“RIB”）。 我们从一台这样的服务器收集数据，并使用 `bgpdump` 工具将数据解析为更易于解析的输出格式。 BGP RIB 表中的条目如下所示：

```
TIME: 03/07/16 02:00:00
TYPE: TABLE_DUMP_V2/IPV4_UNICAST
PREFIX: 0.0.0.0/0
SEQUENCE: 0
FROM: 185.44.116.1 AS47872
ORIGINATED: 03/06/16 20:27:05
ORIGIN: IGP
ASPATH: 47872 3356
NEXT_HOP: 185.44.116.1
COMMUNITY: 3356:2 3356:514 3356:2087 47872:1 47872:3356

TIME: 03/07/16 02:00:00
TYPE: TABLE_DUMP_V2/IPV4_UNICAST
PREFIX: 0.0.0.0/0
SEQUENCE: 0
FROM: 80.241.176.31 AS20771
ORIGINATED: 03/04/16 10:21:21
ORIGIN: IGP
ASPATH: 20771 1299
NEXT_HOP: 80.241.176.31
```

BGP RIB 可能有多个 IP 前缀条目。

对于此分配，我们仅考虑 IP 前缀的单个条目。 我们将此数据转换为 `bgp_rib.csv` 文件。 每个包含以下字段：

```
TIME, ORIGIN, FROM, SEQUENCE, ASPATH, PREFIX, NEXT_HOP
```

以下代码将 `bgp_rib.csv` 文件导入到字典列表中（就像我们上面对 `netflow_data` 所做的那样）。

```
import csv

with open('bgp_rib.csv', 'r') as bgp_file:
    bgp_reader = csv.DictReader(bgp_file,delimiter=";")
    bgp_data = list(bgp_reader)

print "Number of BGP RIBs: {}".format(len(bgp_data))
print
print "Sample BGP RIB: {}".format(bgp_data[0])
```

BGP 路由视图数据中最长的路由（按唯一 AS 的数量）是多少?

要回答这个问题，您需要按“ASPATH”字段中**唯一** AS 编号的数量对“bgp_data”进行排序。

完成下面的代码以找到最长的 ASPATH 并将其分配给“longest_aspath”变量。 您可以使用 Python 的内置 `sorted()` 函数 (https://docs.python.org/2/library/functions.html#sorted) 或您选择的其他方法。

```
from testing import check_longest_aspath

# TODO: Find the longest (by number of *unique* AS numbers) ASPATH and assign it
to the 'longest_aspath' variable


# print and check the longest ASPATH
print "The longest ASPATH is: {}\n".format(longest_aspath)
check_longest_aspath(longest_aspath)
```

问题

请回答以下问题

- 问题6：如果您在线搜索“AS 编号查找”，您将找到多种 AS 搜索服务。 查找这条最长路线中的 AS，以找到其原籍国。 按顺序列出这些国家。
- 问题7：为什么如此长的 BGP 路由会成为网络运营商关注的问题？ 至少给出 2 个理由。