基于dpdk的模拟链路级DDoS攻击实验实验报告

57119101 王晨阳

2021年7月18日

实验概述&实验目的

分布式拒绝服务攻击（英文名称 Distributed Denial of Service，简称 DDoS）是指处于不同位置的多个攻击者同时向一个或数个目标发动攻击，或者一个攻击者控制了位于不同位置的多台机器并利用这些机器对受害者同时实施攻击。由于攻击的发出点是分布在不同地方的，这类攻击称为分布式拒绝服务攻击，其中的攻击者可以有多个。
本实验利用 10Gbps 网卡前端的服务器来模拟僵尸网络所产生的流量。利用 Cisco TRex 进行发包来尝试逼近链路带宽的理论值，以达到模拟 DDos 攻击时网络链路没有多余带宽来为正常用户使用的情况。根据以太网帧的结构分析，帧的大小介于 64bit 和 1518bit，本实验采用这两个极限值作为网络包的大小来进行流量监测。dpdk 是绕过 linux 内核的网络发包环境，实验将采用控制变量法，对不同大小的报文用不同的核数进行流量监测，分析不同核数时的 cpu 利用率以及吞吐量。

实验流程

连接网络

Dell服务器通过千兆电口eth0连接教育网。请确认可以访问外网。

Dell服务器共有四个万兆光纤网口 nic0～nic3。用下面的命令找到闪光的nic0网口。

```
sudo ethtool -p nic0
```

用光纤连接网口nic0和新华三交换机。 ctrl+C 退出。

用下面的命令找到闪光的nic1网口。

```
sudo ethtool -p nic1
```

用光纤连接网口nic1和新华三交换机。

设置网口

`ifconfig -a` 查看网口信息，列出了 Intel X710 光纤网卡的网口信息，包括nic0和nic1和nic2和nic3，默认驱动是 Kernel Driver i40e。

```
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.226.2 netmask 255.255.255.0 broadcast 192.168.226.255
        ether 6c:2b:59:f2:e3:f8 txqueuelen 1000 (以太网)
        RX packets 132053 bytes 117380759 (117.3 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 84238 bytes 11494156 (11.4 MB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
        device interrupt 16 memory 0x90300000-90320000
eth1: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 6c:2b:59:f2:e2:5c txqueuelen 1000 (以太网)
        RX packets 4839 bytes 3024379 (3.0 MB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 3615 bytes 709001 (709.0 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
        device memory 0x90100000-9017ffff
nic0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.100.2 netmask 255.255.255.0 broadcast 192.168.100.255
        ether 3c:fd:fe:a6:6f:f0 txqueuelen 1000 (以太网)
        RX packets 1686 bytes 429116 (429.1 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 2180 bytes 526558 (526.5 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
nic1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.100.6 netmask 255.255.255.0 broadcast 192.168.100.255
        ether 3c:fd:fe:a6:6f:f1 txqueuelen 1000 (以太网)
        RX packets 47 bytes 4851 (4.8 KB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 600 bytes 115530 (115.5 KB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
nic2: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 3c:fd:fe:a6:6f:f2 txqueuelen 1000 (以太网)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
nic3: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
        ether 3c:fd:fe:a6:6f:f3 txqueuelen 1000 (以太网)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

在Dell工作站上 ping 一下H3C新华三万兆光纤交换机保证双向联通。

```
ping 192.168.100.1
```

用下面的命令可以查看哪些网口在dpdk绑定管理中。这里要sudo权限。

```
cd ~/trex/v2.89
sudo ./dpdk_nic_bind.py -s
```

然后，执行 dpdk 端口设置脚本。这里要sudo权限。

```
cd ~/trex/v2.89
sudo ./dpdk_setup_ports.py -i
```

进行 64Byte 最小报文的发包测试

先将nic2和nic3解绑，否则后面nic0和nic1做dpdk驱动绑定的时候，会提示冲突。

```
sudo ./dpdk_nic_bind.py -u 0000:a6:00.2
sudo ./dpdk_nic_bind.py -u 0000:a6:00.3
```

在dell服务器上执行如下命令:

```
sudo ./t-rex-64 -f cap2/imix_64_fast.yaml -m 20 -l 1000
#参数m是发包重放次数，此处是20倍
#l 是网络抖动检测
```

进行 1518Byte 最大报文的发包实验

在dell服务器上执行如下命令：变更 `-c` 的参数，观察cpu利用率以及吞吐率并绘制不同包大小、不同核数时cpu利用率以及吞吐率。

```
sudo ./t-rex-64 -f cap2/imix_1518.yaml -m 120 -l 1000 -c 2
#参数c可以调整服务器上用来收发报文的核数
#cap2/imix_1518.yaml 为包大小1518B的配置文件
```

用下面的命令将nic0、nic1、nic2、nic3重新交还给Linux内核驱动管理。

```
sudo ./dpdk_nic_bind.py -b i40e 0000:a6:00.0 0000:a6:00.1 0000:a6:00.2
0000:a6:00.3
```

进行594Byte中间报文的发包实验

发送594字节的数据包执行下述命令:

```
sudo ./t-rex-64 -f cap2/imix_594.yaml -m 263 -l 1000 -c 2
```

使用iperf3进行Linux内核发包

在浪潮服务器上ping通192.168.100.2 ， 并开启iperf3服务器端:

```
ping 192.168.100.2
iperf3 -s -B 192.168.100.5
```

在dell服务器中， 我们首先使用ping 命令查看是否能够连接到浪潮服务器:

```
ping 192.168.100.5
```

确认连通以后，在dell服务器中开启iperf3客户端命令：

```
iperf3 -c 192.168.100.5 -M 88 -B 192.168.100.2 -b 1000M
```

实验结果&分析

线路吞吐率和包发转速率

在帧长确定的情况下，更多的 CPU 意味着更高的包转发速率和吞吐率，但受到网络总带宽的限制，这个速率是由上限的。

对于相同的 CPU 核数来说，由于不可能每个比特都传输有效数据，导致不同帧长的包转发速率和吞吐率产生差异。通俗地讲，就是发送的内容当中，想要发送的数据占比不同。

> 以太网每个帧之间会有帧间距（IPG），默认帧间距大小为12 字节。每个帧还有7 个字节的前导（Preamble），和 1 个字节的帧首定界符（SFD），共 20 字节。

我们使用的是 10 Gbps 的网络，因此有如下理论计算值：

- 帧长 64 Bytes

  $M = \frac{10\times10^9}{(64+20)\times8} = 14880952\text{pps}$

  $T = M \times 64 \times 8 = 7.619\text{Gbps}$

- 帧长 594 Bytes

  $M = \frac{10\times10^9}{(594+20)\times8} = 2035831\text{pps}$

  $T = M \times 594 \times 8 = 9.674\text{Gbps}$

- 帧长 1518 Bytes

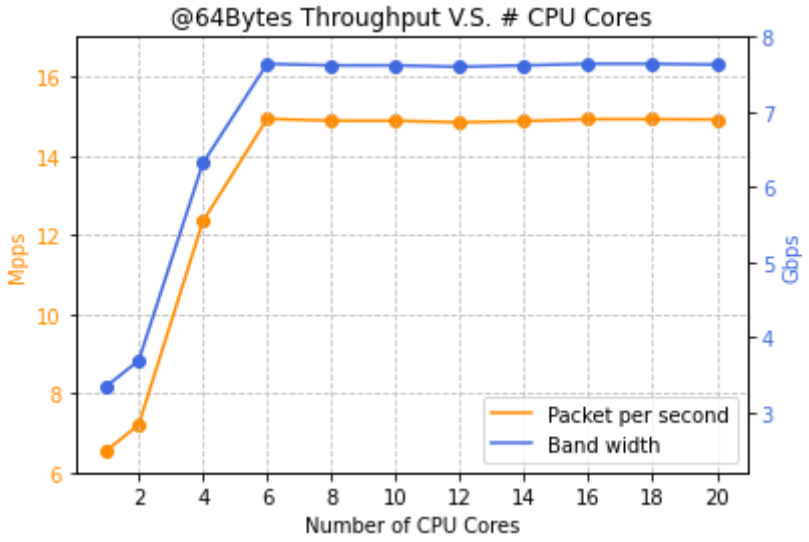
  $M = \frac{10\times10^9}{(1518+20)\times8} = 812743\text{pps}$

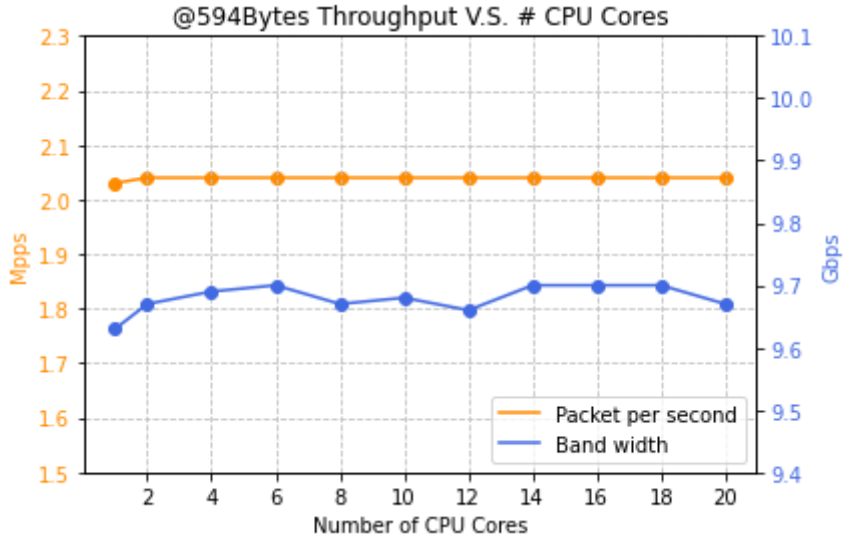
  $T = M \times 1518 \times 8 = 9.869\text{Gbps}$

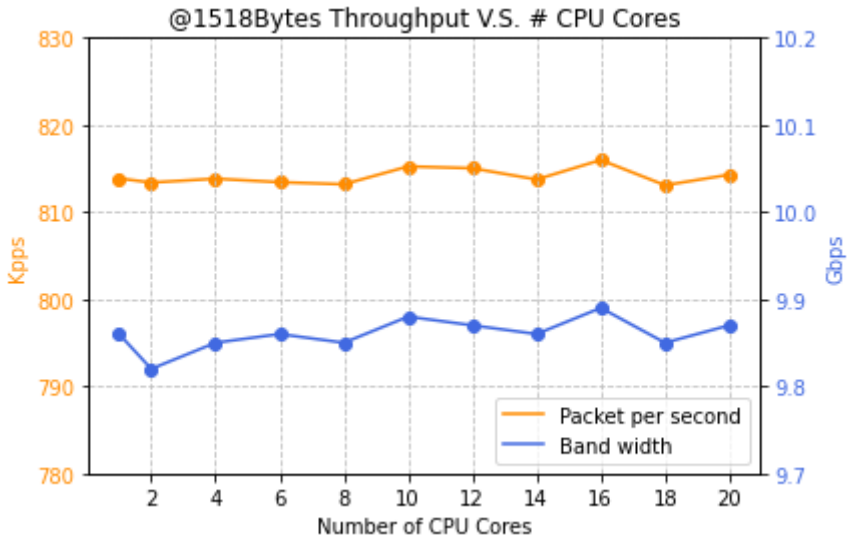
帧长 / Bytes	吞吐率 T / Gbps	包转发速率 M / pps
64	7.619	14880952
594	9.674	2035831
1518	9.869	812743

容易看出，帧长越长，吞吐率越高，包转发速率越低。

以下为实验实际测出来的结果：



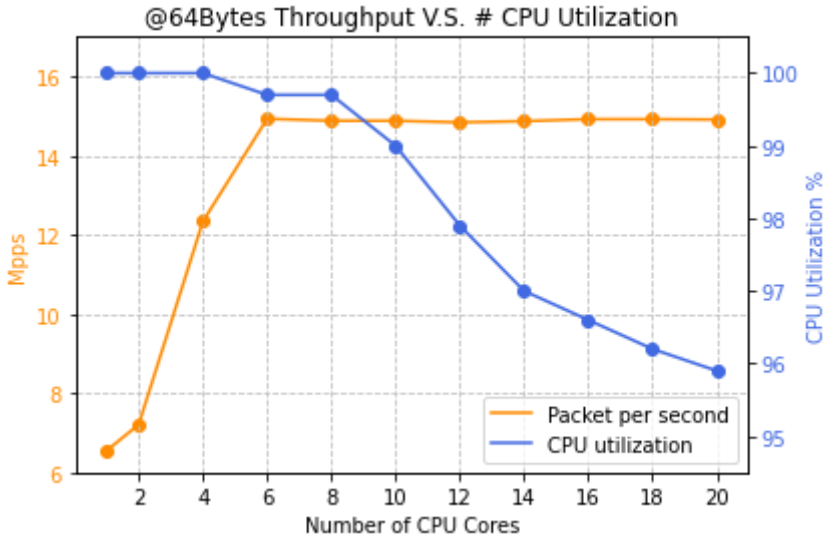


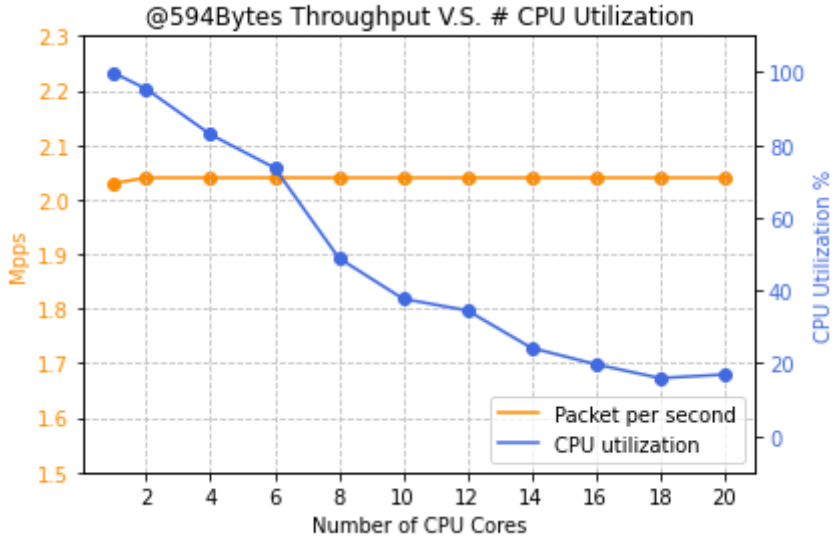


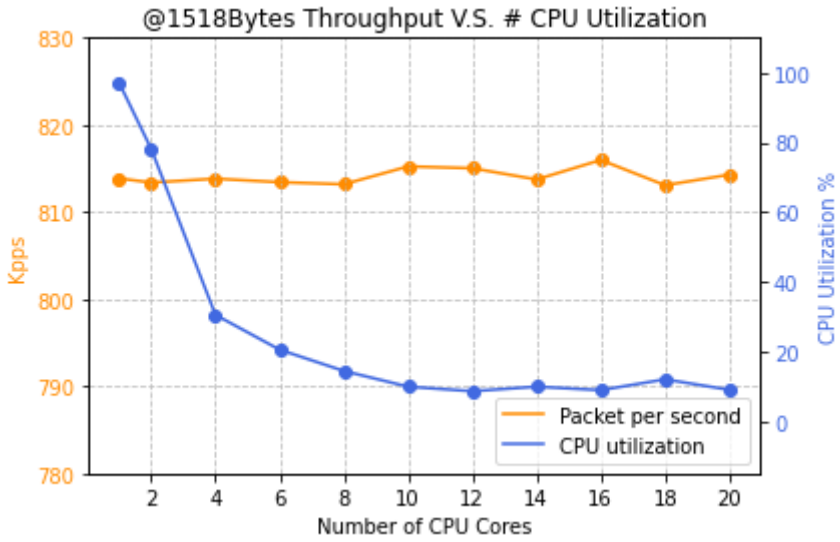
线路吞吐率和 CPU 利用率

CPU 核数越多，单位时间处理的数据增加。但受制于总线宽度，利用率随核数增加而下降。

以下为实验实际测出来的结果：







dpdk 和 Linux 内核 I/O

```
euguest@Dell-Precision-7920-Tower:~/trex/v2.89$ iperf3 -c 192.168.100.5 -M 88 -B 192.168.100.2 -b 1000M
Connecting to host 192.168.100.5, port 5201
[  5] local 192.168.100.2 port 50833 connected to 192.168.100.5 port 5201
[ ID] Interval           Transfer     Bitrate         Retr  Cwnd
[  5]   0.00-1.00   sec  95.2 MBytes   799 Mbits/sec  2475   44.1 KBytes
[  5]   1.00-2.00   sec  86.5 MBytes   726 Mbits/sec  2727   31.5 KBytes
[  5]   2.00-3.00   sec  81.8 MBytes   686 Mbits/sec  2984   28.3 KBytes
[  5]   3.00-4.00   sec  79.6 MBytes   668 Mbits/sec  3856   17.1 KBytes
[  5]   4.00-5.00   sec  86.9 MBytes   729 Mbits/sec  2437   64.8 KBytes
[  5]   5.00-6.00   sec  82.2 MBytes   690 Mbits/sec  1799   33.5 KBytes
[  5]   6.00-7.00   sec  85.0 MBytes   713 Mbits/sec  1926   32.9 KBytes
[  5]   7.00-8.00   sec  90.5 MBytes   759 Mbits/sec  2251   28.1 KBytes
[  5]   8.00-9.00   sec  77.8 MBytes   652 Mbits/sec  2940   29.5 KBytes
[  5]   9.00-10.00  sec  62.8 MBytes   526 Mbits/sec  1619   24.2 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bitrate         Retr
[  5]   0.00-10.00  sec   828 MBytes   695 Mbits/sec  25014            sender
[  5]   0.00-10.04  sec   826 MBytes   690 Mbits/sec                  receiver
```

可以看到，使用 Linux 内核 I/O 速度是明显慢于 dpdk 的。这是因为，前者使用套接字来进行系统调用，其大量时间花费在了上下文切换与数据拷贝上。而 dpdk 直接绕过了 Linux 内核与硬件交互，获得了更快的速度。

实验总结

本实验研究了 DDos 的基本原理，探讨了帧长和 CPU 核数对线路吞吐率、CPU 利用率的影响，并比较了 dpdk 和 Linux 内核 I/O 的区别。通过本实验，增进了对这些内容的理解。

附录

- 绘图代码 - pyhton

```python
### transformed from jupyter notebook ###
### not executable directly by python ###

import matplotlib.pyplot as plt
from matplotlib import style

# Data
numberOfCores       = [1     ,2     ,4     ,6     ,8     ,10    ,
                       12    ,14    ,16    ,18    ,20    ]

tx_Gbps_64          = [3.35  ,3.69  ,6.32  ,7.64  ,7.62  ,7.62  ,
                       7.60  ,7.62  ,7.64  ,7.64  ,7.63  ]
rx_Gbps_64          = [6.70  ,6.14  ,7.60  ,7.64  ,7.62  ,7.62  ,
                       7.60  ,7.62  ,7.64  ,7.64  ,7.63  ]
Mpps_64             = [6.55  ,7.21  ,12.34 ,14.93 ,14.88 ,14.88 ,
                       14.84 ,14.87 ,14.92 ,14.92 ,14.91 ]
cpuUtilization_64   = [100.0 ,100.0 ,100.0 ,99.7  ,99.7  ,99.0  ,
                       97.9  ,97.0  ,96.6  ,96.2  ,95.9  ]

tx_Gbps_594         = [9.63  ,9.67  ,9.69  ,9.70  ,9.67  ,9.68  ,
                       9.66  ,9.70  ,9.70  ,9.70  ,9.67  ]
rx_Gbps_594         = [9.63  ,9.67  ,9.69  ,9.70  ,9.67  ,9.68  ,
                       9.66  ,9.70  ,9.70  ,9.70  ,9.67  ]
Mpps_594            = [2.03  ,2.04  ,2.04  ,2.04  ,2.04  ,2.04  ,
                       2.04  ,2.04  ,2.04  ,2.04  ,2.04  ]
cpuUtilization_594  = [99.8  ,95.4  ,82.9  ,73.7  ,48.9  ,37.7  ,
                       34.5  ,24.2  ,19.7  ,15.9  ,16.9  ]

tx_Gbps_1518        = [9.86  ,9.82  ,9.85  ,9.86  ,9.85  ,9.88  ,
                       9.87  ,9.86  ,9.89  ,9.85  ,9.87  ]
```

```
rx_Gbps_1518        = [9.86  ,9.82  ,9.83  ,9.88  ,9.86  ,9.88  ,
                       9.87  ,9.86  ,9.89  ,9.85  ,9.87  ]
Kpps_1518           = [813.86,813.39,813.83,813.43,813.19,815.24,
                       815.03,813.74,815.98,813.08,814.29]
cpuUtilization_1518 = [97.1  ,78.1  ,30.5  ,20.5  ,14.4  ,9.9   ,
                       8.6   ,9.9   ,8.9   ,12.0  ,9.0   ]


# Preprocess
Gbps_64   = tx_Gbps_64
Gbps_594  = tx_Gbps_594
Gbps_1518 = tx_Gbps_1518

style.use('grayscale')


# frame length=64Byte
## pps & bps
fig=plt.figure()

fig.patch.set_facecolor('white')

ax1=fig.subplots()
ax2=ax1.twinx()

l1=ax1.plot(numberOfCores, Mpps_64, linestyle='-', color='darkorange',
label='Packet per second')
l2=ax2.plot(numberOfCores, Gbps_64, linestyle='-', color='royalblue',
label='Band width')

ax1.scatter(numberOfCores, Mpps_64, marker='o', c='darkorange')
ax2.scatter(numberOfCores, Gbps_64, marker='o', c='royalblue')

ax1.set_xlabel('Number of CPU Cores')
plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])

ax1.set_ylabel('Mpps')
ax2.set_ylabel('Gbps')

ax1.set_ylim(6,17)
ax2.set_ylim(2.2,8)

ax1.yaxis.label.set_color('darkorange')
ax2.yaxis.label.set_color('royalblue')

ax1.yaxis.set_tick_params(labelcolor='darkorange')
ax2.yaxis.set_tick_params(labelcolor='royalblue')

plt.title('@64Bytes Throughput V.S. # CPU Cores')

ax1.grid(linestyle='--', c='silver')

ln = l1+l2
labs = [l.get_label() for l in ln]
ax1.legend(ln, labs, loc=4)

plt.show()

```

```
87
88  ## pps & cpu utilization
89  fig=plt.figure()
90
91  fig.patch.set_facecolor('white')
92
93  ax1=fig.subplots()
94  ax2=ax1.twinx()
95
96  l1=ax1.plot(numberOfCores, Mpps_64, linestyle='-', color='darkorange',
    label='Packet per second')
97  l2=ax2.plot(numberOfCores, cpuUtilization_64, linestyle='-', color='royalblue',
    label='CPU utilization')
98
99  ax1.scatter(numberOfCores, Mpps_64, marker='o', c='darkorange')
100 ax2.scatter(numberOfCores, cpuUtilization_64, marker='o', c='royalblue')
101
102 ax1.set_xlabel('Number of CPU Cores')
103 plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])
104
105 ax1.set_ylabel('Mpps')
106 ax2.set_ylabel('CPU Utilization %')
107
108 ax1.set_ylim(6,17)
109 ax2.set_ylim(94.5,100.5)
110
111 ax1.yaxis.label.set_color('darkorange')
112 ax2.yaxis.label.set_color('royalblue')
113
114 ax1.yaxis.set_tick_params(labelcolor='darkorange')
115 ax2.yaxis.set_tick_params(labelcolor='royalblue')
116
117 plt.title('@64Bytes Throughput V.S. # CPU Utilization')
118
119 ax1.grid(linestyle='--', c='silver')
120
121 ln = l1+l2
122 labs = [l.get_label() for l in ln]
123 ax1.legend(ln, labs, loc=4)
124
125 plt.show()
126
127
128 # frame length=594Byte
129 ## bps & pps
130 fig=plt.figure()
131
132 fig.patch.set_facecolor('white')
133
134 ax1=fig.subplots()
135 ax2=ax1.twinx()
136
137 l1=ax1.plot(numberOfCores, Mpps_594, linestyle='-', color='darkorange',
    label='Packet per second')
138 l2=ax2.plot(numberOfCores, Gbps_594, linestyle='-', color='royalblue',
    label='Band width')
139
140 ax1.scatter(numberOfCores, Mpps_594, marker='o', c='darkorange')
```

```
ax2.scatter(numberOfCores, Gbps_594, marker='o', c='royalblue')

ax1.set_xlabel('Number of CPU Cores')
plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])

ax1.set_ylabel('Mpps')
ax2.set_ylabel('Gbps')

ax1.set_ylim(1.5,2.3)
ax2.set_ylim(9.4,10.1)

ax1.yaxis.label.set_color('darkorange')
ax2.yaxis.label.set_color('royalblue')

ax1.yaxis.set_tick_params(labelcolor='darkorange')
ax2.yaxis.set_tick_params(labelcolor='royalblue')

plt.title('@594Bytes Throughput V.S. # CPU Cores')

ax1.grid(linestyle='--', c='silver')

ln = l1+l2
labs = [l.get_label() for l in ln]
ax1.legend(ln, labs, loc=4)

plt.show()


## pps & cpu utilization
fig=plt.figure()

fig.patch.set_facecolor('white')

ax1=fig.subplots()
ax2=ax1.twinx()

l1=ax1.plot(numberOfCores, Mpps_594, linestyle='-', color='darkorange',
label='Packet per second')
l2=ax2.plot(numberOfCores, cpuUtilization_594, linestyle='-', color='royalblue',
label='CPU utilization')

ax1.scatter(numberOfCores, Mpps_594, marker='o', c='darkorange')
ax2.scatter(numberOfCores, cpuUtilization_594, marker='o', c='royalblue')

ax1.set_xlabel('Number of CPU Cores')
plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])

ax1.set_ylabel('Mpps')
ax2.set_ylabel('CPU Utilization %')

ax1.set_ylim(1.5,2.3)
ax2.set_ylim(-10,110)

ax1.yaxis.label.set_color('darkorange')
ax2.yaxis.label.set_color('royalblue')

ax1.yaxis.set_tick_params(labelcolor='darkorange')
ax2.yaxis.set_tick_params(labelcolor='royalblue')
```

```
197
198 plt.title('@594Bytes Throughput V.S. # CPU Utilization')
199
200 ax1.grid(linestyle='--', c='silver')
201
202 ln = l1+l2
203 labs = [l.get_label() for l in ln]
204 ax1.legend(ln, labs, loc=4)
205
206 plt.show()
207
208
209 # frame length=1518Byte
210 ## pps & bps
211 fig=plt.figure()
212
213 fig.patch.set_facecolor('white')
214
215 ax1=fig.subplots()
216 ax2=ax1.twinx()
217
218 l1=ax1.plot(numberOfCores, Kpps_1518, linestyle='-', color='darkorange',
    label='Packet per second')
219 l2=ax2.plot(numberOfCores, Gbps_1518, linestyle='-', color='royalblue',
    label='Band width')
220
221 ax1.scatter(numberOfCores, Kpps_1518, marker='o', c='darkorange')
222 ax2.scatter(numberOfCores, Gbps_1518, marker='o', c='royalblue')
223
224 ax1.set_xlabel('Number of CPU Cores')
225 plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])
226
227 ax1.set_ylabel('Kpps')
228 ax2.set_ylabel('Gbps')
229
230 ax1.set_ylim(780,830)
231 ax2.set_ylim(9.7,10.2)
232
233 ax1.yaxis.label.set_color('darkorange')
234 ax2.yaxis.label.set_color('royalblue')
235
236 ax1.yaxis.set_tick_params(labelcolor='darkorange')
237 ax2.yaxis.set_tick_params(labelcolor='royalblue')
238
239 plt.title('@1518Bytes Throughput V.S. # CPU Cores')
240
241 ax1.grid(linestyle='--', c='silver')
242
243 ln = l1+l2
244 labs = [l.get_label() for l in ln]
245 ax1.legend(ln, labs, loc=4)
246
247 plt.show()
248
249
250 # pps & cpu utilization
251 fig=plt.figure()
252
```

```
fig.patch.set_facecolor('white')

ax1=fig.subplots()
ax2=ax1.twinx()

l1=ax1.plot(numberOfCores, Kpps_1518, linestyle='-', color='darkorange',
label='Packet per second')
l2=ax2.plot(numberOfCores, cpuUtilization_1518, linestyle='-',
color='royalblue', label='CPU utilization')

ax1.scatter(numberOfCores, Kpps_1518, marker='o', c='darkorange')
ax2.scatter(numberOfCores, cpuUtilization_1518, marker='o', c='royalblue')

ax1.set_xlabel('Number of CPU Cores')
plt.xticks(ticks=[2,4,6,8,10,12,14,16,18,20])

ax1.set_ylabel('Kpps')
ax2.set_ylabel('CPU Utilization %')

ax1.set_ylim(780,830)
ax2.set_ylim(-15,110)

ax1.yaxis.label.set_color('darkorange')
ax2.yaxis.label.set_color('royalblue')

ax1.yaxis.set_tick_params(labelcolor='darkorange')
ax2.yaxis.set_tick_params(labelcolor='royalblue')

plt.title('@1518Bytes Throughput V.S. # CPU Utilization')

ax1.grid(linestyle='--', c='silver')

ln = l1+l2
labs = [l.get_label() for l in ln]
ax1.legend(ln, labs, loc=4)

plt.show()
```