Lab #6: eBPF

Kotaiba Alachkar 8-)

Preparation

```
     sudo apt install vagrant virtualbox wget
     sudo modprobe vboxdrv vboxnetadp vboxnetflt vboxpci
     wget http://amiens.studlab.os3.nl/an2018/lab-ebpf/an-ebpf.tgz
     tar xf an-ebpf.tgz
     cd an-ebpf
     vagrant up
     vagrant ssh
     sudo su
     cd /vagrant
     cd src
     make
```

The output command should state **’SUCCESS’** in the last line:

```
+ tc qdisc del dev eth1 clsact
+ set +x
+ tc qdisc add dev eth1 clsact
+ tc filter add dev eth1 ingress prio 1 handle 1 bpf da obj tcbpf1_kern.o
sec classifier
+ set +x
SUCCESS
```

Task 1: Compiling and usage of sample eBPF program (2 points)

```
/*
 * AN2018 lab6 ebpf
 *
 * This is a modified verison of linux/samples/bpf/tcbpf1_kern.c
 *
 * it includes bpf_debug.h to output debug information that can be
 * read using tools/bpf-trace.
 *
 * ifindex can be obtained using: ip link show | cut -c 1
 *
 */

#define KBUILD_MODNAME "foo"
#include <uapi/linux/bpf.h>
```

```c
#include <uapi/linux/if_ether.h>
#include <uapi/linux/if_packet.h>
#include <uapi/linux/ip.h>
#include <uapi/linux/in.h>
#include <uapi/linux/tcp.h>
#include <uapi/linux/udp.h>
#include <uapi/linux/filter.h>
#include <uapi/linux/pkt_cls.h>
#include "bpf_helpers.h"
#include "bpf_debug.h"


/* compiler workaround */
#define _htonl __builtin_bswap32

static inline void set_dst_mac(struct __sk_buff *skb, char *mac)
{
    //sets destination mac
    bpf_skb_store_bytes(skb, 0, mac, ETH_ALEN, 1);
}

#define IP_CSUM_OFF (ETH_HLEN + offsetof(struct iphdr, check))
#define TOS_OFF (ETH_HLEN + offsetof(struct iphdr, tos))

static inline void set_ip_tos(struct __sk_buff *skb, __u8 new_tos)
{
    //sets tos, and recalculate checksum
    __u8 old_tos = load_byte(skb, TOS_OFF);

    bpf_l3_csum_replace(skb, IP_CSUM_OFF, htons(old_tos), htons(new_tos),
2);
    bpf_skb_store_bytes(skb, TOS_OFF, &new_tos, sizeof(new_tos), 0);
}

#define IP_SRC_OFF (ETH_HLEN + offsetof(struct iphdr, saddr))

#define TCP_CSUM_OFF (ETH_HLEN + sizeof(struct iphdr) + offsetof(struct
tcphdr, check))
#define UDP_CSUM_OFF (ETH_HLEN + sizeof(struct iphdr) + offsetof(struct
udphdr, check))

#define IS_PSEUDO 0x10

static inline void set_tcp_ip_src(struct __sk_buff *skb, __u32 new_ip) {
    //set source address for tcp and recalculates checksum
    __u32 old_ip = _htonl(load_word(skb, IP_SRC_OFF));

    bpf_l4_csum_replace(skb, TCP_CSUM_OFF, old_ip, new_ip, IS_PSEUDO |
sizeof(new_ip));
    bpf_l3_csum_replace(skb, IP_CSUM_OFF, old_ip, new_ip, sizeof(new_ip));
    bpf_skb_store_bytes(skb, IP_SRC_OFF, &new_ip, sizeof(new_ip), 0);
```

```
}

#define TCP_DPORT_OFF (ETH_HLEN + sizeof(struct iphdr) + offsetof(struct
tcphdr, dest))
static inline void set_tcp_dest_port(struct __sk_buff *skb, __u16 new_port)
{
    //set destination port for tcp and recalculates checksum
    __u16 old_port = htons(load_half(skb, TCP_DPORT_OFF));

    bpf_l4_csum_replace(skb, TCP_CSUM_OFF, old_port, new_port,
sizeof(new_port));
    bpf_skb_store_bytes(skb, TCP_DPORT_OFF, &new_port, sizeof(new_port), 0);
}

#define UDP_DPORT_OFF (ETH_HLEN + sizeof(struct iphdr) + offsetof(struct
udphdr, dest))
static inline void set_udp_dest_port(struct __sk_buff *skb, __u16 new_port)
{
    //set destinationport for udp and recalculates checksum
    __u16 old_port = htons(load_half(skb, UDP_DPORT_OFF));

    bpf_l4_csum_replace(skb, UDP_CSUM_OFF, old_port, new_port,
sizeof(new_port));
    bpf_skb_store_bytes(skb, UDP_DPORT_OFF, &new_port, sizeof(new_port), 0);
}

static inline void set_udp_ip_src(struct __sk_buff *skb, __u32 new_ip) {
    //set source address for udp and recalculats checksum
    __u32 old_ip = _htonl(load_word(skb, IP_SRC_OFF));

    bpf_l4_csum_replace(skb, UDP_CSUM_OFF, old_ip, new_ip, IS_PSEUDO |
sizeof(new_ip));
    bpf_l3_csum_replace(skb, IP_CSUM_OFF, old_ip, new_ip, sizeof(new_ip));
    bpf_skb_store_bytes(skb, IP_SRC_OFF, &new_ip, sizeof(new_ip), 0);
}

static inline __u32 ip(__u8 o1, __u8 o2, __u8 o3, __u8 o4){
    /* takes the four octets of a ip address and calculates the hex version
     * in nework byte order*/
    __u32 result = 0;
    result += o4 * 256*256*256;
    result += o3 * 256*256;
    result += o2 * 256;
    result += o1;
    return result;
}

static inline void print_ip( __u32 ip){
   /* prints 'human readable' ipv4 address using bpf_debug
    * the function writes two output lines since kprint is limited
    * to three arguments */
```

```c
    __u8 o1 = (__u8) ip;
    __u8 o2 = (__u8) (ip >> 8);
    __u8 o3 = (__u8) (ip >> 16);
    __u8 o4 = (__u8) (ip >> 24);
    bpf_debug("ip(1) %d.%d.x.x\n", o1, o2);
    bpf_debug("ip(2) x.x.%d.%d\n", o3, o4);
}

SEC("rewrite_tcp")
int _rewrite_tcp(struct __sk_buff *skb)
{
    __u8 proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr,
protocol));
    __u8 ifindex = 255;

    if (proto == IPPROTO_TCP) {
        __u8 old_tos = load_byte(skb, TOS_OFF);
        __u16 old_port = load_half(skb, TCP_DPORT_OFF);
        __u32 old_ip = _htonl(load_word(skb, IP_SRC_OFF));

        //set_udp_ip_src(skb, 0xA010101);  //1.1.1.10
        //set_udp_ip_src(skb, 0xFE02A8C0); //192.168.2.254
        //set_udp_ip_src(skb, 0xA0A0A0A0); //10.10.10.10

        set_udp_ip_src(skb, ip(10,10,10,10)); //10.10.10.10
        set_ip_tos(skb, 8);
        set_tcp_dest_port(skb, htons(8000));

        __u8 new_tos = load_byte(skb, TOS_OFF);
        __u16 new_port = load_half(skb, TCP_DPORT_OFF);
        __u32 new_ip = _htonl(load_word(skb, IP_SRC_OFF));

        bpf_debug("rewrote tos %d -> %d\n", old_tos, new_tos);
        bpf_debug("rewrote src_ip %x -> %x\n", old_ip, new_ip);
        bpf_debug("<old_dest_ip>\n");
        print_ip(old_ip);
        bpf_debug("</old_dest_ip>\n");
        bpf_debug("<new_dest_ip>\n");
        print_ip(new_ip);
        bpf_debug("</new_dest_ip>\n");
        bpf_debug("rewrote dst_port %d -> %d\n", old_port, new_port);

        if (ifindex != 255) {
            return bpf_clone_redirect(skb, ifindex, 0);
        }
    }
    return BPF_OK;
}

SEC("rewrite_udp")
int _rewrite_udp(struct __sk_buff *skb) {
```

```c
    __u8 proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr,
protocol));
    __u8 ifindex = 255;

    if (proto == IPPROTO_UDP) {
        __u8 old_tos = load_byte(skb, TOS_OFF);
        __u16 old_port = load_half(skb, UDP_DPORT_OFF);
        __u32 old_ip = _htonl(load_word(skb, IP_SRC_OFF));

        // rewrite packet
        //set_udp_ip_src(skb, 0xA010101);  //1.1.1.10
        //set_udp_ip_src(skb, 0xFE02A8C0); //192.168.2.254
        //set_udp_ip_src(skb, 0xA0A0A0A0); //10.10.10.10

        set_udp_ip_src(skb, ip(192,168,2,254)); //192.168.2.254
        set_ip_tos(skb, 8);
        set_udp_dest_port(skb, htons(8000));

        __u8 new_tos = load_byte(skb, TOS_OFF);
        __u16 new_port = load_half(skb, UDP_DPORT_OFF);
        __u32 new_ip = _htonl(load_word(skb, IP_SRC_OFF));

        bpf_debug("rewrote tos %d -> %d\n", old_tos, new_tos);
        bpf_debug("rewrote src_ip %x -> %x\n", old_ip, new_ip);
        bpf_debug("<old_dest_ip>\n");
        print_ip(old_ip);
        bpf_debug("</old_dest_ip>\n");
        bpf_debug("<new_dest_ip>\n");
        print_ip(new_ip);
        bpf_debug("</new_dest_ip>\n");
        bpf_debug("rewrote dst_port %d -> %d\n", old_port, new_port);

        if (ifindex != 255) {
            return bpf_clone_redirect(skb, ifindex, 0);
        }
    }
    return BPF_OK;
}

SEC("task4")
int _task4(struct __sk_buff *skb) {
    /* Modify this function to filter certain packets.
     * Use the code above and  linux-4.15/include/uapi/linux/bpf.h
     * as a reference. */

    return BPF_OK;
}

char _license[] SEC("license") = "GPL";
```

Q1.1 Look at the code section SEC("rewrite_tcp") and explain the code within. Be specific, explain the role of used function parameters and the purpose of returned values.

SEC(“rewrite_tc”) code snippest:

```
SEC("rewrite_tcp")
int _rewrite_tcp(struct __sk_buff *skb)
{
    __u8 proto = load_byte(skb, ETH_HLEN + offsetof(struct iphdr,
protocol));
    __u8 ifindex = 255;

    if (proto == IPPROTO_TCP) {
        __u8 old_tos = load_byte(skb, TOS_OFF);
        __u16 old_port = load_half(skb, TCP_DPORT_OFF);
        __u32 old_ip = _htonl(load_word(skb, IP_SRC_OFF));

        //set_udp_ip_src(skb, 0xA010101);  //1.1.1.10
        //set_udp_ip_src(skb, 0xFE02A8C0); //192.168.2.254
        //set_udp_ip_src(skb, 0xA0A0A0A0); //10.10.10.10

        set_udp_ip_src(skb, ip(10,10,10,10)); //10.10.10.10
        set_ip_tos(skb, 8);
        set_tcp_dest_port(skb, htons(8000));

        __u8 new_tos = load_byte(skb, TOS_OFF);
        __u16 new_port = load_half(skb, TCP_DPORT_OFF);
        __u32 new_ip = _htonl(load_word(skb, IP_SRC_OFF));

        bpf_debug("rewrote tos %d -> %d\n", old_tos, new_tos);
        bpf_debug("rewrote src_ip %x -> %x\n", old_ip, new_ip);
        bpf_debug("<old_dest_ip>\n");
        print_ip(old_ip);
        bpf_debug("</old_dest_ip>\n");
        bpf_debug("<new_dest_ip>\n");
        print_ip(new_ip);
        bpf_debug("</new_dest_ip>\n");
        bpf_debug("rewrote dst_port %d -> %d\n", old_port, new_port);

        if (ifindex != 255) {
            return bpf_clone_redirect(skb, ifindex, 0);
        }
    }
    return BPF_OK;
}
```

The SEC(“rewrite_tcp”) is changing the SourceIP, DstPort and TOS parameters within the packet using a predefined function if the protocol value equals to the one given in the IPPROTO_TCP constant value. In addition to that if the ifindex is not the default value “255”, the code will redirect the packet to a new interface, specified as ifindex.

The input of the function is buffer skb. load_byte loads one byte on a specific offset from buffer skb. load_half loads two bytes on a specific offset from buffer skb. load_word loads four bytes on a specific offset from buffer skb.

The function returns an integer. It contains multiple definitions of unsigned integers:

- u8: Unsigned 8-bit integer * u16: Unsigned 16-bit integer
- u32: Unsigned 32-bit integer ==== Q2.2 - Compile the program using tools/bpf-compile wrapper: tools/bpf-compile <filename>. Apply the program to eth1 interface using: ./tools/bpf-tc eth1 bpf_sample.o rewrite_tcp Verify that the object file is loaded and provide the output of the above command. Hint: use tc.==== Compile the program using tools/bpf-compile wrapper: <code> root@archlinux:/vagrant/src# ../tools/bpf-compile bpf_sample.c ++ sed -e 's/\.c$/\.o/' +++ basename bpf_sample.c ++ echo bpf_sample.c + OBJ_FILENAME=bpf_sample.o + LLC=llc + CLANG=clang + ARCH=x86 + ARCH_FULL=x86_64-linux-gnu ++ uname -r + HEADERS=/usr/src/linux-headers-4.15.3-1-ARCH + KERNEL_MASTER=/vagrant/linux-4.15 + llc -march=bpf -filetype=obj -o bpf_sample.o + exec clang -nostdinc -isystem /usr/lib/gcc/x86_64-linux-gnu/6/include -I/vagrant/linux-4.15/arch/x86/include -I/vagrant/linux-4.15/arch/x86/include/generated -I/vagrant/linux-4.15/include -I/usr/lib/clang/5.0.1/include -I/vagrant/linux-4.15/arch/x86/include/uapi -I/vagrant/linux-4.15/arch/x86/include/generated/uapi -I/vagrant/linux-4.15/include/uapi -I/vagrant/linux-4.15/include/generated/uapi -include/vagrant/linux-4.15/include/linux/kconfig.h -I/vagrant/linux-4.15/samples/bpf -I/vagrant/linux-4.15/tools/testing/selftests/bpf/ -I/vagrant/include -DKERNEL -DASM_SYSREG_H -Wno-unused-value -Wno-pointer-sign -fno-stack-protector -Wno-compare-distinct-pointer-types -Wno-gnu-variable-sized-type-not-at-end -Wno-address-of-packed-member -Wno-tautological-compare -Wno-unknown-warning-option -O2 -emit-llvm -c bpf_sample.c -o -

</code>

Apply the program to eth1 interface:

```
root@archlinux:/vagrant/src# ../tools/bpf-tc eth1 bpf_sample.o rewrite_tcp
+ tc qdisc del dev eth1 clsact
+ set +x
+ tc qdisc add dev eth1 clsact
+ tc filter add dev eth1 ingress prio 1 handle 1 bpf da obj bpf_sample.o sec
rewrite_tcp
+ set +x
```

Verify the object file:

```
root@archlinux:/vagrant/src# tc filter show dev eth1 ingress
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1 bpf_sample.o:[rewrite_tcp]
direct-action not_in_hw id 15 tag e7983f733cf41b3f jited
```

Q3.3 - Illustrate the functionality realized by the attached program. Use tools

such as ping or nc to generate sample packets and tcpdump for your packet traces.

Create a dummy net interface:

```
root@archlinux:/vagrant/src# ip link add dummy0 type dummy
root@archlinux:/vagrant/src# ip link set dev dummy0 up

root@archlinux:/vagrant/src# ip link show dummy0
4: dummy0: <BROADCAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN mode DEFAULT group default qlen 1000
    link/ether 72:7b:b7:82:94:4f brd ff:ff:ff:ff:ff:ff
```

I changed the interface index in bpf_sample.c to 4 since dummy0 is in index 4:

```
__u8 ifindex = 4;
```

Now, recompile again:

```
root@archlinux:/vagrant/src# ../tools/bpf-compile bpf_sample.c
++ sed -e 's/\.c$/\.o/'
+++ basename bpf_sample.c
++ echo bpf_sample.c
+ OBJ_FILENAME=bpf_sample.o
+ LLC=llc
+ CLANG=clang
+ ARCH=x86
+ ARCH_FULL=x86_64-linux-gnu
++ uname -r
+ HEADERS=/usr/src/linux-headers-4.15.3-1-ARCH
+ KERNEL_MASTER=/vagrant/linux-4.15
+ exec clang -nostdinc -isystem /usr/lib/gcc/x86_64-linux-gnu/6/include -
I/vagrant/linux-4.15/arch/x86/include -
I/vagrant/linux-4.15/arch/x86/include/generated -
I/vagrant/linux-4.15/include -I/usr/lib/clang/5.0.1/include -
I/vagrant/linux-4.15/arch/x86/include/uapi -
I/vagrant/linux-4.15/arch/x86/include/generated/uapi -
I/vagrant/linux-4.15/include/uapi -
I/vagrant/linux-4.15/include/generated/uapi -
include/vagrant/linux-4.15/include/linux/kconfig.h -
I/vagrant/linux-4.15/samples/bpf -
I/vagrant/linux-4.15/tools/testing/selftests/bpf/ -I/vagrant/include -
D__KERNEL__ -D__ASM_SYSREG_H -Wno-unused-value -Wno-pointer-sign -fno-stack-
protector -Wno-compare-distinct-pointer-types -Wno-gnu-variable-sized-type-
not-at-end -Wno-address-of-packed-member -Wno-tautological-compare -Wno-
unknown-warning-option -O2 -emit-llvm -c bpf_sample.c -o -
+ llc -march=bpf -filetype=obj -o bpf_sample.o

root@archlinux:/vagrant/src# ../tools/bpf-tc eth1 bpf_sample.o rewrite_tcp
+ tc qdisc del dev eth1 clsact
```

```
+ set +x
+ tc qdisc add dev eth1 clsact
+ tc filter add dev eth1 ingress prio 1 handle 1 bpf da obj bpf_sample.o sec
rewrite_tcp
+ set +x
```

Setup a tcpdum on dummy0 and use nc to the vagrant box using “nc 192.168.2.100 80”:

On my server:

```
root@bristol:~# nc 192.168.2.100 80
```

On vagrant:

```
tcpdump: listening on dummy0, link-type EN10MB (Ethernet), capture size
262144 bytes
07:44:06.145936 IP (tos 0x8, ttl 64, id 40173, offset 0, flags [DF], proto
TCP (6), length 60)
    10.10.10.10.37748 > 192.168.2.100.8000: Flags [S], cksum 0x3ef9
(correct), seq 734858136, win 29200, options [mss 1460,sackOK,TS val
332251631 ecr 0,nop,wscale 7], length 0
```

The ToS value was changed to 8 and the source IP-address to “10.10.10.10”.

Task 2: Writing eBPF program: traffic firewalling (3 points)

Q2.1 Implement your filtering program in the sample by creating your own, it can be as simple as accepting only specific protocols (e.g. TCP+IPv4 only) or verifying port numbers and IP addresses. Use the section SEC("task4"). Use tcpdump to verify that it works.

Only capture a non-80 traffic:

```
SEC("task4")
int _task4(struct __sk_buff *skb) {
    /* Modify this function to filter certain packets.
     * Use the code above and  linux-4.15/include/uapi/linux/bpf.h
     * as a reference. */

if (oldPort == 80) {
    return BPF_DROP;
} else {
    return bpf_clone_redirect(skb, ifindex, 0);
}
}
```

Task 3: Filtering performance measurements (3 points)

Compiling:

```
root@archlinux:/vagrant# tools/bpf-compile src/ip_filter_w_map.c
++ sed -e 's/\.c$/\.o/'
+++ basename src/ip_filter_w_map.c
++ echo ip_filter_w_map.c
+ OBJ_FILENAME=ip_filter_w_map.o
+ LLC=llc
+ CLANG=clang
+ ARCH=x86
+ ARCH_FULL=x86_64-linux-gnu
++ uname -r
+ HEADERS=/usr/src/linux-headers-4.15.3-1-ARCH
+ KERNEL_MASTER=/vagrant/linux-4.15
+ exec clang -nostdinc -isystem /usr/lib/gcc/x86_64-linux-gnu/6/include -
I/vagrant/linux-4.15/arch/x86/include -
I/vagrant/linux-4.15/arch/x86/include/generated -
I/vagrant/linux-4.15/include -I/usr/lib/clang/5.0.1/include -
I/vagrant/linux-4.15/arch/x86/include/uapi -
I/vagrant/linux-4.15/arch/x86/include/generated/uapi -
I/vagrant/linux-4.15/include/uapi -
I/vagrant/linux-4.15/include/generated/uapi -
include/vagrant/linux-4.15/include/linux/kconfig.h -
I/vagrant/linux-4.15/samples/bpf -
I/vagrant/linux-4.15/tools/testing/selftests/bpf/ -I/vagrant/include -
D__KERNEL__ -D__ASM_SYSREG_H -Wno-unused-value -Wno-pointer-sign -fno-stack-
protector -Wno-compare-distinct-pointer-types -Wno-gnu-variable-sized-type-
not-at-end -Wno-address-of-packed-member -Wno-tautological-compare -Wno-
unknown-warning-option -O2 -emit-llvm -c src/ip_filter_w_map.c -o -
+ llc -march=bpf -filetype=obj -o ip_filter_w_map.o

root@archlinux:/vagrant# tools/bpf-tc eth1 ip_filter_w_map.o classifier
+ tc qdisc del dev eth1 clsact
+ set +x
+ tc qdisc add dev eth1 clsact
+ tc filter add dev eth1 ingress prio 1 handle 1 bpf da obj
ip_filter_w_map.o sec classifier
Note: 8 bytes struct bpf_elf_map fixup performed due to size mismatch!
+ set +x
```

Verification of bpf-map:

```
root@archlinux:/vagrant# bpf-map info /sys/fs/bpf/tc/globals/ddos
Type:       Hash
Key size:   4
Value size: 8
Max entries:    11000
```

```
Flags:      0x0
```

Remove eBPF object file and check again:

```
root@archlinux:/vagrant# tc filter show ingress dev eth1
filter protocol all pref 1 bpf chain 0
filter protocol all pref 1 bpf chain 0 handle 0x1
ip_filter_w_map.o:[classifier] direct-action not_in_hw id 17 tag
33fbcfbd5e591bb2 jited
root@archlinux:/vagrant# tc qdisc del dev eth1 clsact
root@archlinux:/vagrant# tc filter show ingress dev eth1
```

Q3.1 - Execute iptables -t raw -i eth1 -A PREROUTING -j NOTRACK on the vagrant VM to prevent ’conntrack table’ space exhaustion. Start an iperf3 server on the VM and run iperf3 -c 192.168.2.100 -t 70 -O 10 on the host machine to test the bandwidth be- tween the host and VM. First, run it as is to see the raw performance. Also on the host system, run hping3 --rand-source 192.168.2.100 --faster command to start generat- ing DDoS traffic; leave hping3 running it the background. If hping3 kills your connection, change --faster to -i u1000 and change the number to increase or decrease the sending rate of hping3. The goal is to see slight degradation of iperf speeds (caused by hping3) without any rules applied. Show what you did, and that a hping3 is properly tuned.

First, Execute:

```
root@archlinux:/vagrant# iptables -t raw -i eth1 -A PREROUTING -j NOTRACK
root@archlinux:/vagrant# iperf3 -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
```

On my server:

```
root@bristol:/home/kotaiba# iperf3 -c 192.168.2.100 -t 70 -O 10
Connecting to host 192.168.2.100, port 5201
[  4] local 192.168.2.1 port 49758 connected to 192.168.2.100 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   228 MBytes  1.91 Gbits/sec   96    249 KBytes
(omitted)
[  4]   1.00-2.00   sec   231 MBytes  1.93 Gbits/sec  161    387 KBytes
(omitted)
[  4]   2.00-3.00   sec   227 MBytes  1.91 Gbits/sec   24    437 KBytes
(omitted)
[  4]   3.00-4.00   sec   230 MBytes  1.93 Gbits/sec   25    450 KBytes
(omitted)
[  4]   4.00-5.00   sec   240 MBytes  2.01 Gbits/sec   62    317 KBytes
```

```
(omitted)
[  4]   5.00-6.00   sec   247 MBytes  2.07 Gbits/sec   43    411 KBytes
(omitted)
[  4]   6.00-7.00   sec   249 MBytes  2.09 Gbits/sec   53    273 KBytes
(omitted)
[  4]   7.00-8.00   sec   245 MBytes  2.06 Gbits/sec   94    379 KBytes
(omitted)
[  4]   8.00-9.00   sec   247 MBytes  2.07 Gbits/sec  125    363 KBytes
(omitted)
[  4]   9.00-10.00  sec   219 MBytes  1.84 Gbits/sec   60    424 KBytes
(omitted)
[  4]   0.00-1.00   sec   241 MBytes  2.02 Gbits/sec   49    468 KBytes
[  4]   1.00-2.00   sec   221 MBytes  1.86 Gbits/sec   24    452 KBytes
[  4]   2.00-3.00   sec   225 MBytes  1.88 Gbits/sec   52    455 KBytes
[  4]   3.00-4.00   sec   225 MBytes  1.89 Gbits/sec   12    464 KBytes
[  4]   4.00-5.00   sec   227 MBytes  1.91 Gbits/sec   57    406 KBytes
[  4]   5.00-6.00   sec   221 MBytes  1.85 Gbits/sec   11    396 KBytes
[  4]   6.00-7.00   sec   229 MBytes  1.92 Gbits/sec  181    354 KBytes
[  4]   7.00-8.00   sec   223 MBytes  1.87 Gbits/sec  108    404 KBytes
[  4]   8.00-9.00   sec   219 MBytes  1.84 Gbits/sec  143    427 KBytes
[  4]   9.00-10.00  sec   224 MBytes  1.88 Gbits/sec  108    368 KBytes
[  4]  10.00-11.00  sec   228 MBytes  1.91 Gbits/sec  118    392 KBytes
[  4]  11.00-12.00  sec   223 MBytes  1.87 Gbits/sec   12    417 KBytes
[  4]  12.00-13.00  sec   224 MBytes  1.88 Gbits/sec  123    444 KBytes
[  4]  13.00-14.00  sec   244 MBytes  2.05 Gbits/sec   29    433 KBytes
[  4]  14.00-15.00  sec   246 MBytes  2.06 Gbits/sec  109    499 KBytes
[  4]  15.00-16.00  sec   240 MBytes  2.02 Gbits/sec   90    427 KBytes
[  4]  16.00-17.00  sec   227 MBytes  1.91 Gbits/sec   55    423 KBytes
[  4]  17.00-18.00  sec   239 MBytes  2.00 Gbits/sec  107    378 KBytes
[  4]  18.00-19.00  sec   226 MBytes  1.89 Gbits/sec   17    399 KBytes
[  4]  19.00-20.00  sec   220 MBytes  1.85 Gbits/sec   54    329 KBytes
[  4]  20.00-21.00  sec   226 MBytes  1.89 Gbits/sec   32    402 KBytes
[  4]  21.00-22.00  sec   221 MBytes  1.85 Gbits/sec   75    406 KBytes
[  4]  22.00-23.00  sec   228 MBytes  1.92 Gbits/sec   17    433 KBytes
[  4]  23.00-24.00  sec   226 MBytes  1.89 Gbits/sec  149    362 KBytes
[  4]  24.00-25.00  sec   233 MBytes  1.95 Gbits/sec   88    314 KBytes
[  4]  25.00-26.00  sec   234 MBytes  1.96 Gbits/sec  168    352 KBytes
[  4]  26.00-27.00  sec   232 MBytes  1.95 Gbits/sec   31    352 KBytes
[  4]  27.00-28.00  sec   224 MBytes  1.88 Gbits/sec   15    467 KBytes
[  4]  28.00-29.00  sec   224 MBytes  1.88 Gbits/sec   11    471 KBytes
[  4]  29.00-30.00  sec   221 MBytes  1.85 Gbits/sec   54    477 KBytes
[  4]  30.00-31.00  sec   229 MBytes  1.92 Gbits/sec  103    436 KBytes
[  4]  31.00-32.00  sec   230 MBytes  1.93 Gbits/sec   66    396 KBytes
[  4]  32.00-33.00  sec   235 MBytes  1.97 Gbits/sec   61    284 KBytes
[  4]  33.00-34.00  sec   242 MBytes  2.03 Gbits/sec   63    313 KBytes
[  4]  34.00-35.00  sec   246 MBytes  2.06 Gbits/sec  157    370 KBytes
[  4]  35.00-36.00  sec   229 MBytes  1.92 Gbits/sec   61    444 KBytes
[  4]  36.00-37.00  sec   235 MBytes  1.97 Gbits/sec   47    395 KBytes
[  4]  37.00-38.00  sec   228 MBytes  1.91 Gbits/sec  185    304 KBytes
[  4]  38.00-39.00  sec   225 MBytes  1.89 Gbits/sec   45    335 KBytes
[  4]  39.00-40.00  sec   219 MBytes  1.84 Gbits/sec   59    332 KBytes
```

```
[  4]  40.00-41.00  sec   229 MBytes  1.92 Gbits/sec   41    342 KBytes
[  4]  41.00-42.00  sec   229 MBytes  1.92 Gbits/sec   19    354 KBytes
[  4]  42.00-43.00  sec   228 MBytes  1.91 Gbits/sec  108    400 KBytes
[  4]  43.00-44.00  sec   222 MBytes  1.86 Gbits/sec   61    441 KBytes
[  4]  44.00-45.00  sec   228 MBytes  1.91 Gbits/sec   42    329 KBytes
[  4]  45.00-46.00  sec   227 MBytes  1.91 Gbits/sec   24    304 KBytes
[  4]  46.00-47.00  sec   224 MBytes  1.88 Gbits/sec  114    409 KBytes
[  4]  47.00-48.00  sec   235 MBytes  1.97 Gbits/sec  197    389 KBytes
[  4]  48.00-49.00  sec   236 MBytes  1.98 Gbits/sec   87    430 KBytes
[  4]  49.00-50.00  sec   225 MBytes  1.89 Gbits/sec   60    423 KBytes
[  4]  50.00-51.00  sec   222 MBytes  1.86 Gbits/sec   59    443 KBytes
[  4]  51.00-52.00  sec   222 MBytes  1.86 Gbits/sec   93    368 KBytes
[  4]  52.00-53.00  sec   236 MBytes  1.98 Gbits/sec   68    373 KBytes
[  4]  53.00-54.00  sec   237 MBytes  1.99 Gbits/sec   83    392 KBytes
[  4]  54.00-55.00  sec   229 MBytes  1.92 Gbits/sec   30    420 KBytes
[  4]  55.00-56.00  sec   229 MBytes  1.92 Gbits/sec  166    383 KBytes
[  4]  56.00-57.00  sec   219 MBytes  1.84 Gbits/sec   40    460 KBytes
[  4]  57.00-58.00  sec   222 MBytes  1.86 Gbits/sec   45    458 KBytes
[  4]  58.00-59.00  sec   230 MBytes  1.93 Gbits/sec  210    303 KBytes
[  4]  59.00-60.00  sec   230 MBytes  1.93 Gbits/sec   57    378 KBytes
[  4]  60.00-61.00  sec   219 MBytes  1.84 Gbits/sec   57    389 KBytes
[  4]  61.00-62.00  sec   240 MBytes  2.01 Gbits/sec   44    291 KBytes
[  4]  62.00-63.00  sec   223 MBytes  1.87 Gbits/sec   13    434 KBytes
[  4]  63.00-64.00  sec   222 MBytes  1.86 Gbits/sec   29    457 KBytes
[  4]  64.00-65.00  sec   218 MBytes  1.83 Gbits/sec   29    355 KBytes
[  4]  65.00-66.00  sec   231 MBytes  1.94 Gbits/sec   57    455 KBytes
[  4]  66.00-67.00  sec   228 MBytes  1.91 Gbits/sec   81    395 KBytes
^C[  4]  67.00-67.48  sec   107 MBytes  1.89 Gbits/sec    9    385 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-67.48  sec  15.0 GBytes  1.92 Gbits/sec  4869
sender
[  4]   0.00-67.48  sec  0.00 Bytes  0.00 bits/sec                  receiver
iperf3: interrupt - the client has terminated
```

With hping in the backgroud, the following output are generated:

```
root@bristol:/home/kotaiba# iperf3 -c 192.168.2.100 -t 70 -O 10
Connecting to host 192.168.2.100, port 5201
[  4] local 192.168.2.1 port 49762 connected to 192.168.2.100 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   168 MBytes  1.41 Gbits/sec  371    328 KBytes
(omitted)
[  4]   1.00-2.00   sec   164 MBytes  1.37 Gbits/sec  162    317 KBytes
(omitted)
[  4]   2.00-3.00   sec   174 MBytes  1.46 Gbits/sec   56    297 KBytes
(omitted)
[  4]   3.00-4.00   sec   167 MBytes  1.41 Gbits/sec  111    314 KBytes
(omitted)
[  4]   4.00-5.00   sec   162 MBytes  1.36 Gbits/sec  136    291 KBytes
(omitted)
```

```
[  4]   5.00-6.00   sec   166 MBytes  1.39 Gbits/sec   56    310 KBytes
(omitted)
[  4]   6.00-7.00   sec   171 MBytes  1.44 Gbits/sec   89    301 KBytes
(omitted)
[  4]   7.00-8.00   sec   162 MBytes  1.36 Gbits/sec   78    252 KBytes
(omitted)
[  4]   8.00-9.00   sec   163 MBytes  1.37 Gbits/sec   82    376 KBytes
(omitted)
[  4]   9.00-10.00  sec   175 MBytes  1.47 Gbits/sec  140    267 KBytes
(omitted)
[  4]   0.00-1.00   sec   164 MBytes  1.38 Gbits/sec  133    313 KBytes
[  4]   1.00-2.00   sec   167 MBytes  1.40 Gbits/sec  142    287 KBytes
[  4]   2.00-3.00   sec   167 MBytes  1.40 Gbits/sec  118    293 KBytes
[  4]   3.00-4.00   sec   165 MBytes  1.39 Gbits/sec   84    313 KBytes
[  4]   4.00-5.00   sec   166 MBytes  1.39 Gbits/sec  125    304 KBytes
[  4]   5.00-6.00   sec   170 MBytes  1.42 Gbits/sec   58    356 KBytes
[  4]   6.00-7.00   sec   157 MBytes  1.32 Gbits/sec   83    297 KBytes
[  4]   7.00-8.00   sec   169 MBytes  1.42 Gbits/sec   97    240 KBytes
[  4]   8.00-9.00   sec   164 MBytes  1.37 Gbits/sec  136    238 KBytes
[  4]   9.00-10.00  sec   165 MBytes  1.38 Gbits/sec  115    242 KBytes
[  4]  10.00-11.00  sec   164 MBytes  1.38 Gbits/sec  101    235 KBytes
[  4]  11.00-12.00  sec   168 MBytes  1.41 Gbits/sec   72    308 KBytes
[  4]  12.00-13.00  sec   174 MBytes  1.46 Gbits/sec   97    250 KBytes
[  4]  13.00-14.00  sec   178 MBytes  1.49 Gbits/sec   38    337 KBytes
[  4]  14.00-15.00  sec   161 MBytes  1.35 Gbits/sec   82    262 KBytes
[  4]  15.00-16.00  sec   171 MBytes  1.44 Gbits/sec   53    327 KBytes
[  4]  16.00-17.00  sec   169 MBytes  1.42 Gbits/sec  109    317 KBytes
[  4]  17.00-18.00  sec   157 MBytes  1.32 Gbits/sec   17    366 KBytes
[  4]  18.00-19.00  sec   166 MBytes  1.39 Gbits/sec   79    318 KBytes
[  4]  19.00-20.00  sec   162 MBytes  1.36 Gbits/sec  154    296 KBytes
[  4]  20.00-21.00  sec   172 MBytes  1.44 Gbits/sec   48    297 KBytes
[  4]  21.00-22.00  sec   166 MBytes  1.39 Gbits/sec  122    283 KBytes
[  4]  22.00-23.00  sec   174 MBytes  1.46 Gbits/sec   98    296 KBytes
[  4]  23.00-24.00  sec   172 MBytes  1.44 Gbits/sec   75    346 KBytes
[  4]  24.00-25.00  sec   167 MBytes  1.40 Gbits/sec  110    329 KBytes
[  4]  25.00-26.00  sec   175 MBytes  1.46 Gbits/sec   61    321 KBytes
[  4]  26.00-27.00  sec   172 MBytes  1.44 Gbits/sec  115    304 KBytes
[  4]  27.00-28.00  sec   161 MBytes  1.36 Gbits/sec   83    249 KBytes
[  4]  28.00-29.00  sec   171 MBytes  1.44 Gbits/sec   68    345 KBytes
[  4]  29.00-30.00  sec   167 MBytes  1.41 Gbits/sec   55    356 KBytes
[  4]  30.00-31.00  sec   162 MBytes  1.36 Gbits/sec   55    399 KBytes
[  4]  31.00-32.00  sec   169 MBytes  1.42 Gbits/sec  103    304 KBytes
[  4]  32.00-33.00  sec   161 MBytes  1.35 Gbits/sec  124    329 KBytes
[  4]  33.00-34.00  sec   171 MBytes  1.43 Gbits/sec  125    322 KBytes
[  4]  34.00-35.00  sec   163 MBytes  1.37 Gbits/sec  106    249 KBytes
[  4]  35.00-36.00  sec   166 MBytes  1.39 Gbits/sec  115    288 KBytes
[  4]  36.00-37.00  sec   161 MBytes  1.35 Gbits/sec   46    410 KBytes
[  4]  37.00-38.00  sec   170 MBytes  1.43 Gbits/sec   80    283 KBytes
[  4]  38.00-39.00  sec   173 MBytes  1.45 Gbits/sec   77    308 KBytes
[  4]  39.00-40.00  sec   168 MBytes  1.41 Gbits/sec   79    321 KBytes
[  4]  40.00-41.00  sec   166 MBytes  1.39 Gbits/sec  100    324 KBytes
```

```
[  4]  41.00-42.00  sec   168 MBytes  1.41 Gbits/sec   84    331 KBytes
[  4]  42.00-43.00  sec   155 MBytes  1.30 Gbits/sec  118    358 KBytes
[  4]  43.00-44.00  sec   172 MBytes  1.44 Gbits/sec   95    300 KBytes
[  4]  44.00-45.00  sec   170 MBytes  1.43 Gbits/sec  103    370 KBytes
[  4]  45.00-46.00  sec   174 MBytes  1.46 Gbits/sec   83    320 KBytes
[  4]  46.00-47.00  sec   163 MBytes  1.36 Gbits/sec   25    301 KBytes
[  4]  47.00-48.00  sec   181 MBytes  1.51 Gbits/sec  111    242 KBytes
[  4]  48.00-49.00  sec   170 MBytes  1.43 Gbits/sec   47    344 KBytes
[  4]  49.00-50.00  sec   167 MBytes  1.40 Gbits/sec  122    342 KBytes
[  4]  50.00-51.00  sec   165 MBytes  1.39 Gbits/sec  119    346 KBytes
[  4]  51.00-52.00  sec   166 MBytes  1.40 Gbits/sec  129    318 KBytes
[  4]  52.00-53.00  sec   159 MBytes  1.34 Gbits/sec  151    342 KBytes
[  4]  53.00-54.00  sec   170 MBytes  1.42 Gbits/sec  140    331 KBytes
[  4]  54.00-55.00  sec   172 MBytes  1.44 Gbits/sec   67    253 KBytes
^C[  4]  55.00-55.20  sec  32.8 MBytes  1.37 Gbits/sec    0    355 KBytes
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-55.20  sec  9.02 GBytes  1.40 Gbits/sec  5132
sender
[  4]   0.00-55.20  sec  0.00 Bytes  0.00 bits/sec                  receiver
```

Q3.2 - On the VM, use tools/load_rules script to load filtering rules for both iptables an eBPF program. First, run it with 'ipt' argument, it creates a new (unreferenced) chain named ddos and fills it with sample IPv4 addresses stored in 10k_random_ip.txt file. Measure the performance when all the incoming traffic goes through 'ddos' chain i.e. iptables -i eth1 -I INPUT 1 -j ddos. Show that the chain is applied and is receiving traffic, also include the output of your measurements. When done, don't forget to remove the rule.

load rule:

```
root@archlinux:/vagrant/tools# ../tools/load_rules ipt
9999
```

New chain called "ddos":

```
Chain ddos (0 references)
target     prot opt source               destination
DROP       tcp  --  38.76.22.47          anywhere             tcp
DROP       tcp  --  19.221.12.198        anywhere             tcp
DROP       tcp  --  122.43.213.70        anywhere             tcp
DROP       tcp  --  81-224-221-121-no89.tbcn.telia.com  anywhere
tcp
DROP       tcp  --  125.149.75.15        anywhere             tcp
DROP       tcp  --  136.31.215.92        anywhere             tcp
DROP       tcp  --  103.249.107.248      anywhere             tcp
DROP       tcp  --  167.123.94.251       anywhere             tcp
.
```

```
.
.
```

Now link the INPUT chain to the DDOS chain:

```
root@archlinux:/vagrant/tools# iptables -I INPUT 1 -j ddos

root@archlinux:/vagrant/tools# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ddos       all  --  anywhere             anywhere
```

Now all INPUT will be forwarded to DDOS chain:

```

root@archlinux:/vagrant/tools# iptables -L -v -n
Chain INPUT (policy ACCEPT 438 packets, 41898 bytes)
 pkts bytes target     prot opt in     out     source
destination

  418 40810 ddos       all  --  *      *       0.0.0.0/0
0.0.0.0/0
Chain ddos (1 references)
```

Test and check:

```

on server:
root@bristol:/home/kotaiba# sudo hping3 --rand-source 192.168.2.100 --faster

on VM:
root@archlinux:/vagrant/tools# iptables -L INPUT -v -n
Chain INPUT (policy ACCEPT 1340 packets, 78670 bytes)
 pkts bytes target     prot opt in     out     source
destination
 1340 78670 ddos       all  --  *      *       0.0.0.0/0
0.0.0.0/0
```

Now, redo the measurment:

```
root@bristol:/home/kotaiba# sudo hping3 --rand-source 192.168.2.100 --faster
&
[1] 9286
root@bristol:/home/kotaiba# HPING 192.168.2.100 (vboxnet0 192.168.2.100): NO
FLAGS are set, 40 headers + 0 data bytes

root@bristol:/home/kotaiba# iperf3 -c 192.168.2.100 -t 70 -O 10
Connecting to host 192.168.2.100, port 5201
[  4] local 192.168.2.1 port 49772 connected to 192.168.2.100 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  4.82 MBytes  40.5 Mbits/sec   78   28.3 KBytes
```

```
(omitted)
[  4]   1.00-2.00   sec  2.34 MBytes  19.6 Mbits/sec    0   66.5 KBytes
(omitted)
[  4]   2.00-3.00   sec  2.51 MBytes  21.1 Mbits/sec   12   26.9 KBytes
(omitted)
[  4]   3.00-4.00   sec  4.07 MBytes  34.2 Mbits/sec    5   82.0 KBytes
(omitted)
[  4]   4.00-5.00   sec  3.84 MBytes  32.2 Mbits/sec    7   84.8 KBytes
(omitted)
[  4]   5.00-6.00   sec  3.82 MBytes  32.1 Mbits/sec    3   96.2 KBytes
(omitted)
[  4]   6.00-7.00   sec  5.32 MBytes  44.6 Mbits/sec    8   94.7 KBytes
(omitted)
[  4]   7.00-8.00   sec  5.30 MBytes  44.5 Mbits/sec    4   91.9 KBytes
(omitted)
[  4]   8.00-9.00   sec  4.51 MBytes  37.8 Mbits/sec   47   24.0 KBytes
(omitted)
[  4]   9.00-10.00  sec  2.72 MBytes  22.9 Mbits/sec   21   33.9 KBytes
(omitted)
[  4]   0.00-1.00   sec  2.43 MBytes  20.4 Mbits/sec    0   69.3 KBytes
[  4]   1.00-2.00   sec  2.14 MBytes  18.0 Mbits/sec   15   60.8 KBytes
[  4]   2.00-3.00   sec  1.48 MBytes  12.4 Mbits/sec   11   43.8 KBytes
[  4]   3.00-4.00   sec  2.80 MBytes  23.5 Mbits/sec    3   72.1 KBytes
[  4]   4.00-5.00   sec  1.81 MBytes  15.2 Mbits/sec   15   41.0 KBytes
[  4]   5.00-6.00   sec  3.83 MBytes  32.1 Mbits/sec    7   65.0 KBytes
[  4]   6.00-7.00   sec  2.23 MBytes  18.7 Mbits/sec   13   59.4 KBytes
[  4]   7.00-8.00   sec  6.69 MBytes  56.1 Mbits/sec    0    117 KBytes
[  4]   8.00-9.00   sec  6.09 MBytes  51.1 Mbits/sec    4    112 KBytes
[  4]   9.00-10.00  sec  2.30 MBytes  19.3 Mbits/sec   31   39.6 KBytes
^C[  4]  10.00-10.62  sec   754 KBytes  9.96 Mbits/sec    3   39.6 KBytes
```

As we see a huge decrease in the bandwidth compared with previous output the speed decreased in around 1Gbit.

Now, flush and list:

```
root@archlinux:/vagrant/tools# iptables -F
root@archlinux:/vagrant/tools# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain ddos (0 references)
target     prot opt source               destination
```

Q3.3 - Attach the eBPF object file, and fill the eBPF map using the tools/load_rules script. This time, use the 'ebpf' argument, to fill the 'ddos' bpf-map with the hex versions of the previously loaded addresses. Show that the map is applied and is receiving traffic. Repeat the previous measurement and include the output and results. Did you notice a performance difference?

```
root@archlinux:/vagrant/tools# bpf-map info /sys/fs/bpf/tc/globals/ddos
Type:        Hash
Key size:    4
Value size: 8
Max entries:    11000
Flags:       0x0
```

Execute the same performance test as in the previous section. The results are the following:

```
root@bristol:/home/kotaiba# iperf3 -c 192.168.2.100 -t 70 -O 10
Connecting to host 192.168.2.100, port 5201
[  4] local 192.168.2.1 port 54056 connected to 192.168.2.100 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec   216 MBytes  1.81 Gbits/sec  213    297 KBytes
(omitted)
[  4]   1.00-2.00   sec   195 MBytes  1.63 Gbits/sec  126    274 KBytes
(omitted)
[  4]   2.00-3.00   sec   212 MBytes  1.78 Gbits/sec  134    339 KBytes
(omitted)
[  4]   3.00-4.00   sec   209 MBytes  1.75 Gbits/sec  230    307 KBytes
(omitted)
[  4]   4.00-5.00   sec   219 MBytes  1.84 Gbits/sec  333    301 KBytes
(omitted)
[  4]   5.00-6.00   sec   205 MBytes  1.72 Gbits/sec  137    215 KBytes
(omitted)
[  4]   6.00-7.00   sec   223 MBytes  1.87 Gbits/sec  304    250 KBytes
(omitted)
[  4]   7.00-8.00   sec   215 MBytes  1.80 Gbits/sec  260    247 KBytes
(omitted)
[  4]   8.00-9.00   sec   223 MBytes  1.87 Gbits/sec  266    225 KBytes
(omitted)
[  4]   9.00-10.00  sec   214 MBytes  1.79 Gbits/sec  139    342 KBytes
(omitted)
[  4]   0.00-1.00   sec   215 MBytes  1.80 Gbits/sec  257    352 KBytes
[  4]   1.00-2.00   sec   221 MBytes  1.85 Gbits/sec  296    334 KBytes
[  4]   2.00-3.00   sec   204 MBytes  1.71 Gbits/sec  151    321 KBytes
[  4]   3.00-4.00   sec   192 MBytes  1.61 Gbits/sec  138    317 KBytes
[  4]   4.00-5.00   sec   199 MBytes  1.67 Gbits/sec  228    305 KBytes
[  4]   5.00-6.00   sec   221 MBytes  1.85 Gbits/sec  331    218 KBytes
[  4]   6.00-7.00   sec   215 MBytes  1.80 Gbits/sec  167    215 KBytes
[  4]   7.00-8.00   sec   209 MBytes  1.75 Gbits/sec  229    331 KBytes
[  4]   8.00-9.00   sec   207 MBytes  1.74 Gbits/sec  101    291 KBytes
[  4]   9.00-10.00  sec   193 MBytes  1.62 Gbits/sec  173    286 KBytes
```

```
[  4]  10.00-11.00  sec   211 MBytes  1.77 Gbits/sec  333    325 KBytes
[  4]  11.00-12.00  sec   216 MBytes  1.81 Gbits/sec  147    277 KBytes
[  4]  12.00-13.00  sec   194 MBytes  1.63 Gbits/sec  169    229 KBytes
[  4]  13.00-14.00  sec   210 MBytes  1.76 Gbits/sec  127    315 KBytes
[  4]  14.00-15.00  sec   212 MBytes  1.78 Gbits/sec  231    317 KBytes
[  4]  15.00-16.00  sec   212 MBytes  1.78 Gbits/sec  156    260 KBytes
[  4]  16.00-17.00  sec   224 MBytes  1.88 Gbits/sec  229    335 KBytes
[  4]  17.00-18.00  sec   210 MBytes  1.76 Gbits/sec  184    317 KBytes
[  4]  18.00-19.00  sec   209 MBytes  1.75 Gbits/sec  171    212 KBytes
[  4]  19.00-20.00  sec   216 MBytes  1.81 Gbits/sec  130    297 KBytes
[  4]  20.00-21.00  sec   234 MBytes  1.96 Gbits/sec  275    240 KBytes
[  4]  21.00-22.00  sec   209 MBytes  1.76 Gbits/sec  144    304 KBytes
[  4]  22.00-23.00  sec   223 MBytes  1.87 Gbits/sec  177    320 KBytes
[  4]  23.00-24.00  sec   212 MBytes  1.78 Gbits/sec  198    280 KBytes
[  4]  24.00-25.00  sec  71.3 MBytes   640 Mbits/sec   21    205 KBytes
[  4]  25.00-26.00  sec  63.9 MBytes   570 Mbits/sec   69    266 KBytes
[  4]  26.00-27.00  sec  66.0 MBytes   554 Mbits/sec   42    208 KBytes
[  4]  27.00-28.00  sec  65.9 MBytes   561 Mbits/sec   23    243 KBytes
[  4]  28.00-29.00  sec  68.4 MBytes   565 Mbits/sec   18    201 KBytes
[  4]  29.00-30.00  sec  52.1 MBytes   479 Mbits/sec   38    235 KBytes
[  4]  30.00-31.00  sec  61.6 MBytes   550 Mbits/sec   60    266 KBytes
[  4]  31.00-32.00  sec  57.7 MBytes   484 Mbits/sec   18    204 KBytes
[  4]  32.00-33.00  sec   231 MBytes  1.75 Gbits/sec  231    290 KBytes
[  4]  33.00-34.00  sec   201 MBytes  1.75 Gbits/sec  201    290 KBytes
[  4]  34.00-35.00  sec   208 MBytes  1.75 Gbits/sec  219    290 KBytes
```

As we see above I used the hping again and we can notice the effect of eBPF on bandwidth is much smaller than the iptables. In addition to eBPF shows speeds very similar to the raw speeds, while Iptables has lower speeds.

Task 4: Understanding eBPF architecture (2 points)

The answers for this task is based on bpf man page as stated in the source.

Q4.1 - What is the difference between (c)BPF and eBPF?

Extended BPF (or eBPF) is similar to the original (“classic”) BPF (cBPF) used to filter network packets.eBPF extends cBPF in multiple ways, including the ability to call a fixed set of in-kernel helper functions (via the BPF_CALL opcode extension provided by eBPF) and access shared data structures such as eBPF maps.

Q4.2 - What is the purpose of eBPF maps?

eBPF maps are a generic data structure for storage of different data types. Data types are generally treated as binary blobs, so a user just specifies the size of the key and the size of the value at map-creation time. In other words, a key/value for a given map can have an arbitrary structure.

Q4.3 - How does an eBPF program gets passed to the kernel? Which user-level tools are used for this?

First, the kernel statically analyzes the programs before loading them, in order to ensure that they cannot harm the running system. Them, the BPF programs are loaded into the kernel by the bpf() system call. The user tc-bpf tools used to remains alive so that the tc-system knows which module specifically to load.

Q4.4 - What kind of operations can be performed on a network packet inside eBPF code?

A possible kind of operations that can be performed are network packets can be forwarded, dropped and changed. We can adjust and modify every byte in a network packet in a way that the network packet will change to something totally different based on changes that are pre-programmed conditions. In general, BPF itself decides whether to drop or forward packets to which destination.

Source:

http://man7.org/linux/man-pages/man2/bpf.2.html