eBPF on the Mainframe
Packet filtering and more

Michael Holzheu <michael.holzheu@de.ibm.com>
LinuxCon Dublin 2015/10/05
Notes:
Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.
IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.
All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending upon individual customer configurations and conditions.
This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.
All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.
Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.
Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

Trademark Disclaimer:
The following are trademarks of the International Business Machines Corporation in the United States and/or other countries.

AIX*, IBM*, PowerVM, System z10, z/OS*
BladeCenter*, IBM eServer, PR/SM, WebSphere*, zSeries*
DataPower*, IBM (logo)*, Smarter Planet, z9*
DB2*, InfiniBand*, System x*, z10 BC
FICON* Parallel Sysplex*, System z*, z10 EC
GDPS* POWER*, System z9*, zEnterprise
HiperSockets POWER7*, PEC12

* Registered trademarks of IBM Corporation
IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

The following are trademarks or registered trademarks of other companies.
Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.
IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.
Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.
Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Windows Server and the Windows logo are trademarks of the Microsoft group of countries.
ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.
Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

* Other product and service names might be trademarks of IBM or other companies.
Notice Regarding Specialty Engines (e.g., zIIPs, zAAPs, and IFLs)

Any information contained in this document regarding Specialty Engines ("SEs") and SE eligible workloads provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g., zIIPs, zAAPs, and IFLs). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the “Authorized Use Table for IBM Machines” provided at


No other workload processing is authorized for execution on an SE.

IBM offers SEs at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.
Agenda

- Berkley Packet Filter (BPF)
- extended Berkley Packet Filter (eBPF)
- eBPF system call
- eBPF and LLVM
BPF - Berkley Packet Filter

- First described in paper 1992 (Steven McCanne and Van Jacobson)
- Generic assembler language and byte code (ISA)
- Used for kernel packet filtering on Free-BSD and Linux
- Interpreter and BPF kernel JIT compiler backends
- Filter program is attached to socket
- Efficient: Packets are dropped already in kernel
BPF - Scenario tcpdump

[client1] # ping server
[client2] # ping server

[server] # tcpdump -i eth0 src client1
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1

sockfd = socket()
BPF - Scenario tcpdump

[server]# tcpdump -i eth0 src client1

BPF program: e370f050...

if (ip.src != client1)
   return 0x0000
else
   return 0xffff

libpcap
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1

```c
setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_FILTER, &bpf, sizeof(bpf));
```

```c
BPF program: e370f050 ...

if (ip.src != client1)
    return 0x0000
else
    return 0xffff
```
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1

read(sockfd, buf, sizeof(buf));

BPF program: e370f050 ...
if (ip.src != client1)
    return 0x0000
else
    return 0xffff
BPF - Scenario tcpdump

```plaintext
[server] # tcpdump -i eth0 src client1

read(sockfd, buf, sizeof(buf));

SKB: src=client2

BPF program: e370f050 ...

if (ip.src != client1)
    return 0x0000
else
    return 0xffff

[client2] # ping server
```
BPF - Scenario tcpdump

[server]  # tcpdump -i eth0 src client1

read(sockfd, buf, sizeof(buf));

BPF program: e370f050 ...
if (ip.src != client1)
    return 0x0000
else
    return 0xffffffff
BPF - Scenario tcpdump

[server]

```bash
[server] # tcpdump -i eth0 src client1
```

```c
read(sockfd, buf, sizeof(buf));
```

kernel

```
BPF program: e370f050 ...
if (ip.src != client1)
    return 0x0000
else
    return 0xffff
```

[client1] [client2]
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1

read(sockfd, buf, sizeof(buf));

BPF program: e370f050 ...

if (ip.src != client1)
    return 0x0000
else
    return 0xffff

SKB: src=client1

[client1] # ping server
BPF - Scenario tcpdump

[server] # tcpdump -i eth0 src client1

read(sockfd, buf, sizeof(buf));

BPF program: e370f050 ...
if (ip.src != client1)
  return 0x0000
else
  return 0xffff

kernel

SKB: src=client1

[client1] [client2]
BPF - Scenario tcpdump

[snapshot:

[server] # tcpdump -i eth0 src client1

IP: src=client1

BPF program: e370f050 ...

if (ip.src != client1)
    return 0x0000
else
    return 0xffff
server: # tcpdump -i eth0 src client1
IP client1 > server: ICMP echo request, id 30146, seq 1, length 64

BPF program: e370f050 ...
if (ip.src != client1)
    return 0x0000
else
    return 0xffffffff

BPF - Scenario tcpdump
BPF - 32 bit machine language (ISA)

- Akkumulator: \( a \)
- Index register: \( x \)
- Immediate: \( k \)
- Jump targets: \( jt \) and \( jf \)
- Temporary memory: \( M[] \)

```c
#include <linux/filter.h>

struct sock_filter {
    __u16   code;
    __u8    jt;
    __u8    jf;
    __u32   k;
};
```
BPF - Example

# tcpdump -i eth0 src client1 -d

(000) ldh [12]  # Read T/L
(001) jeq #0x800  jt 2  jf 4  # IP packet?  ## Read src IP: offset 26
(002) ld [26]  # Read src IP: offset 26
(003) jeq #0x09000001  jt 8  jf 9  # 9.0.0.1 ?
(004) jeq #0x806  jt 6  jf 5  # ARP packet?  ## Read src ARP IP: offset 28
(005) jeq #0x8035  jt 6  jf 9  # RARP packet?  ## Read src ARP IP: offset 28
(006) ld [28]  # Read src ARP IP: offset 28
(007) jeq #0x09000001  jt 8  jf 9  # 9.0.0.1 ?
(008) ret #65535  # client1 -> rc = 0xffff
(009) ret #0  # not client1 -> rc = 0x0000
BPF - sysctl JIT setting

/proc/sys/net/core/bpf_jit_enable:

- 0: Use interpreter
- 1: Use JIT compiler
- 2: Use JIT compiler and print debug output

# echo 2 > /proc/sys/net/core/bpf_jit_enable
# tcpdump -i eth0 src client1
# dmesg

flen=10  proglen=188  pass=4  image=000003ff80016e10
JIT code: 00000000: eb 8f f0 58 00 24 b9 04 00 ...
...  
JIT code: 000000b0: 00 00 00 1a 00 00 00 1c 00 00 ff ff
000003ff80016e10: eb8ff0580024  stmg  %r8,%r15,88(%r15)
000003ff80016e16: b90400ef  lgr  %r14,%r15
...  
000003ff80016ea4: eb8ff0a80004  lmg  %r8,%r15,168(%r15)
000003ff80016ea8: 07fe  bcr  15,%r14
eBPF
ISA History: Register Width
50 years: Register Width

- **Mainframe**
- **x86**
- **ARM**

- **System z**: [https://en.wikipedia.org/wiki/IBM_System_z](https://en.wikipedia.org/wiki/IBM_System_z)
- **ARM64**: [https://de.wikipedia.org/wiki/AMD_Opteron_%28K8%29](https://de.wikipedia.org/wiki/AMD_Opteron_%28K8%29)

### Register Width Over Time

- **8**: 1964
- **16**: 1965
- **32**: 1966
- **64**: 1967
- **Mainframe**
- **x86**
- **ARM**

---

**Legend**

- **ARM64**: [https://de.wikipedia.org/wiki/AMD_Opteron_%28K8%29](https://de.wikipedia.org/wiki/AMD_Opteron_%28K8%29)
- **System z**: [https://en.wikipedia.org/wiki/IBM_System_z](https://en.wikipedia.org/wiki/IBM_System_z)
50 years: Register Width

Mainframe - x86 - ARM - BPF

S/360 8080 8086 80386 ARM

System z AMD64 ARM64 eBPF

eBPF: https://lwn.net/Articles/603983
eBPF - extended Berkley Packet Filter

- Linux 3.15 by Alexei Starovoitov <ast@kernel.org>
- Uses 64-bit registers
- New instructions:
  - 64 bit ALU plus old 32 bit ALU
  - Atomic: bpf_xadd
  - Function call: bpf_call
  - Endianness conversion functions
  - Singed compares
- First used only Linux internally
- BPF programs are converted into eBPF
- Now also externally usable with BPF system call
- LLVM eBPF backend
Eleven 64 bit registers:
- B0: Return value from in-kernel function, and exit value for eBPF program
- B1 - B5: Arguments from eBPF program to in-kernel function
- B6 - B9: Callee saved registers that in-kernel function will preserve
- B10: Read-only frame pointer to access stack
- Offset: off (s16)
- Immediate: imm (s32)

```c
#include <linux/bpf.h>

struct bpf_insn {
    __u8    code;
    __u8    dst_reg:4;
    __u8    src_reg:4;
    __s16   off;
    __s16   off;
    __s32   imm;
};
```

<table>
<thead>
<tr>
<th>op</th>
<th>dst</th>
<th>src</th>
<th>off</th>
<th>imm</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>32</td>
</tr>
</tbody>
</table>
## eBPF - Register mapping on s390x

<table>
<thead>
<tr>
<th>eBPF register</th>
<th>s390x reg</th>
<th>s390x ABI</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0</td>
<td>R0</td>
<td>volatile</td>
<td>Work register (even)</td>
</tr>
<tr>
<td>W1</td>
<td>R1</td>
<td>volatile</td>
<td>Work register (odd)</td>
</tr>
<tr>
<td>B0</td>
<td>R14</td>
<td>volatile</td>
<td>Exit value for eBPF program, Return value from in-kernel function, Return value for BPF_LD. Has to be loaded after call: ldr %b0,r2</td>
</tr>
<tr>
<td>B1-B5</td>
<td>R2-R6</td>
<td>R2-R5 volatile, R6 saved</td>
<td>Parameter registers for kernel function calls, Scratch registers for BPF_LD SKB ops, B1 contains context parameter for BPF program (e.g., SKB pointer).</td>
</tr>
<tr>
<td>B6-B9</td>
<td>R7-R10</td>
<td>saved</td>
<td>Callee saved registers that in-kernel function will preserve</td>
</tr>
<tr>
<td>L</td>
<td>R11</td>
<td>saved</td>
<td>Literal pool</td>
</tr>
<tr>
<td>SKB_DATA</td>
<td>R12</td>
<td>saved</td>
<td>SKB data pointer</td>
</tr>
<tr>
<td>B10 (BFP)</td>
<td>R13</td>
<td>saved</td>
<td>Read-only frame pointer to access stack</td>
</tr>
<tr>
<td></td>
<td>R15</td>
<td>saved</td>
<td>Stack pointer for C functions</td>
</tr>
</tbody>
</table>
eBPF - Converted BPF program

```
# tcpdump -i eth0 eth0 client1

BPF_LD  BPF_ABS BPF_H  src=0 dst=0 off=0 imm=12          # Load protocol type (2048 = IP)
BPF_JMP BPF_JNE BPF_K  src=0 dst=0 off=3 imm=0x800       # if (b0 != 0x800)
BPF_LD  BPF_ABS BPF_W  src=0 dst=0 off=0 imm=26          # b0 = load_word(SKB, 26)
BPF_JMP BPF_JEQ BPF_K  src=0 dst=0 off=5 imm=0x090000001 # if (b0 == 9.0.0.1)
BPF_JMP BPF_JA  BPF_K  src=0 dst=0 off=6 imm=0           # goto [NOT_FOUND]
BPF_ALU BPF_MOV BPF_K  src=0 dst=0 off=0 imm=0xffff      # b0 = 0xffff
BPF_JMP BPF_EXI BPF_K  src=0 dst=0 off=0 imm=0           # exit
```
**eBPF - Converted BPF program**

3ff80006312: eb67f0480024 stmg %r6, %r7, 72(%r15)  # Save registers on stack
3ff80006318: ebbcf0700024 stmg %r11, %r12, 112(%r15)  
3ff8000631e: ebeff0880024 stmg %r14, %r15, 136(%r15)  
3ff80006324: b90400bf lgr %r11, %r15  
3ff80006328: a7fbfda8 aghi %r15, -600  # Get stack space for BPF and function calls
3ff8000632c: e3b0f0980024 stg %r11, 152(%r15)  
3ff80006332: e31020800016 llgf %r1, 128(%r2)  
3ff80006338: 5b102084 s %r1, 132(%r2)  
3ff8000633c: e310f0a80024 stg %r1, 168(%r15)  
3ff80006342: e3c020d80004 lg %r12, 216(%r2)  
3ff80006348: b9040072 lgr %r2, %r14  
3ff8000634c: c01f00134624 llilf %r1, 1263140  # BPF_LD BPF_ABS BPF_H src=0 dst=0 off=0 imm=12
3ff80006352: c03100000800 lgfi %r3, 0  # BPF_JMP BPF_JNE BPF_K src=0 dst=0 off=8 imm=0x800
3ff80006358: 0d61 basr %r1, %r6  
3ff8000635a: a7740043 brc 7, 3ff800063e0  
3ff8000635e: c01100000000 clgr %r14, %r1  
3ff80006362: c01100000806 lgfi %r3, 26  # BPF_JMP BPF_JE BPF_K src=0 dst=0 off=26 imm=26
3ff80006368: a7f40025 brc 15, 3ff800063d6  
3ff8000636c: c0117f000001 clgr %r14, %r1  
3ff80006370: c01100000001 clgr %r3, 0  # BPF_JMP BPF_JE BPF_K src=0 dst=0 off=0 imm=0x00000001
3ff80006372: c0ef0000ffff llilf %r14, 65535  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0xffff
3ff80006378: a7f40009 brc 15, 3ff800063d6  
3ff8000637c: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff80006380: a7f40004 brc 15, 3ff800063d6  
3ff80006384: c01f001345e0 llilf %r1, 1263072  # BPF_JMP BPF_JEQ BPF_K src=0 dst=0 off=1263072 imm=0
3ff80006388: 0d61 basr %r1, %r6  
3ff8000638a: a7740009 brc 8, 3ff800063d6  
3ff8000638e: c0117f000001 clgr %r14, %r1  
3ff80006390: c01100000000 clgr %r3, 0  # BPF_JMP BPF_JE BPF_K src=0 dst=0 off=0 imm=0
3ff80006392: c0310000001a lgfi %r3, 28  # BPF_LD BPF_ABS BPF_W src=0 dst=0 off=0 imm=28
3ff80006398: c01f001345e0 llilf %r1, 1263072  # BPF_JMP BPF_JEQ BPF_K src=0 dst=0 off=1263072 imm=0
3ff8000639c: 0d61 basr %r1, %r6  
3ff800063a0: a774000e brc 8, 3ff800063d6  
3ff800063a4: c0117f000001 clgr %r14, %r1  
3ff800063a8: c01100000000 clgr %r3, 0  # BPF_JMP BPF_JE BPF_K src=0 dst=0 off=0 imm=0
3ff800063ac: c0310000001c lgfi %r3, 28  # BPF_LD BPF_ABS BPF_W src=0 dst=0 off=0 imm=28
3ff800063b0: c01f001345e0 llilf %r1, 1263072  # BPF_JMP BPF_JEQ BPF_K src=0 dst=0 off=1263072 imm=0
3ff800063b4: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063b8: a7f40004 brc 15, 3ff800063d6  
3ff800063bc: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063c0: a7f40009 brc 15, 3ff800063d6  
3ff800063c4: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063c8: a7f40004 brc 15, 3ff800063d6  
3ff800063cc: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063d0: a7f40009 brc 15, 3ff800063d6  
3ff800063d4: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063d8: a7f40004 brc 15, 3ff800063d6  
3ff800063dc: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063e0: a7f40009 brc 15, 3ff800063d6  
3ff800063e4: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063e8: ebf800000002 brc 15, 3ff800063d6  
3ff800063ee: c0ef00000000 llilf %r14, 0  # BPF_ALU BPF_MOV BPF_K src=0 dst=0 off=0 imm=0
3ff800063fa: 07fe bcr 15, %r14  # Return to caller
if (bo != 9.0.0.1) skip 2

lgfi %r1,2415919105 # BPF_JMP BPF_JNE BPF_K dst=0 off=2 imm=0x09000001
cglomer |%r14, %r1       # Note: R14: Mapped to B0
brc 7,3ff800063d6      # R1: Work register
llilf %r14,65535      # BPF_ALU BPF_MOV BPF_K dst=0 imm=0xffff
Performance: One filter run

```
# tcpdump -i lo src localhost
# ping -f localhost
```

Reason: Branch prediction for jumps from low kernel to high module addresses

<table>
<thead>
<tr>
<th>BPF engine</th>
<th>Instructions (executed)</th>
<th>z196</th>
<th>EC12</th>
<th>z13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interpreter</td>
<td>117</td>
<td>40 ns</td>
<td>30 ns</td>
<td>20 ns</td>
</tr>
<tr>
<td>eBPF JIT</td>
<td>55</td>
<td>55 ns</td>
<td>17 ns</td>
<td>8 ns</td>
</tr>
</tbody>
</table>

Notes:
Used STCK before and after filter call to measure time
Test done on development system with (not yet final) JIT.
eBPF system call
eBPF system call

- Integrated in Linux 3.19 (Febr. 2015)
- Useful for creating kernel statistics
- The eBPF system call is a multiplexer with the following functions:
  - BPF map functions: Create, iterate, lookup
  - BPF load function: Load eBPF program
  - Attach maps to eBPF programs
- With a new socket call the eBPF program can be attached to a socket:
  - setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_BPF, &prog_id,...);
Task: Count network packets per protocol

One byte protocol field in IP header
eBPF system call - Scenario

[server] # sockexl_user
eBPF system call - Scenario

```c
[server] # sockex1_user

bpf_create_map()

map:

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0</td>
</tr>
<tr>
<td>UDP</td>
<td>0</td>
</tr>
<tr>
<td>ICMP</td>
<td>0</td>
</tr>
</tbody>
</table>

256 Elements
eBPF system call - Scenario

[server] # sockex1_user

map:
tcp: 0
udp: 0
icmp: 0

eBPF program: e370f050 ...
map[ip[prot]]++
eBPF system call - Scenario

[server] # sockex1_user

prog_id = bpf_prog_load()

eBPF program: e370f050 ...

tcp: 0
udp: 0
icmp: 0

map[ip[prot]]++
eBPF system call - Scenario

[server] # sockex1_user

setsockopt(sockfd, SOL_SOCKET, SO_ATTACH_BPF, &prog_id)

map:
- tcp: 0
- udp: 0
- icmp: 0

map[ip[prot]]++

eBPF program: e370f050 ...
eBPF system call - Scenario

[server] # sockex1_user

kernel

map:
tcp: 0
udp: 0
icmp: 1

ICMP packet 1

eBPF program: e370f050 ...
map[ip[prot]]++

[client] # ping server
eBPF system call - Scenario

[server] # sockex1_user

[client] # ping server

map:
tcp: 0
udp: 0
icmp: 2

ICMP packet 2

eBPF program: e370f050 ...
map[ip[prot]]++
eBPF system call - Scenario

[server] # sockex1_user

bpf_lookup_elem(ICMP) = 2

tcp: 0
udp: 0
icmp: 2

map[ip[prot]]++

eBPF program: e370f050 ...

map: tcp: 0, udp: 0, icmp: 2
eBPF system call - Scenario

```plaintext
[server] # sockex1_user

TCP 0 UDP 0 ICMP 2 packets
```

```
map:
tcp: 0
udp: 0
icmp: 2
```

```
eBPF program: e370f050 ...
map[ip[prot]]++
```
I have a dragon and I'm not afraid to use it!

eBPF & LLVM
LLVM eBPF backend (kernel part)

/* Count packets of different protocols */

#include <uapi/linux/bpf.h>
#include <uapi/linux/if_ether.h>
#include <uapi/linux/ip.h>
#include "bpf_helpers.h"

struct bpf_map_def SEC("maps") my_map = {
   .type = BPF_MAP_TYPE_ARRAY,
   .key_size = sizeof(u32),
   .value_size = sizeof(long),
   .max_entries = 256,
};

SEC("socket1")
int bpf_prog1(struct sk_buff *skb)
{
    int index = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
    long *value;

    value = bpf_map_lookup_elem(&my_map, &index);
    if (value)
        __sync_fetch_and_add(value, 1);

    return 0;
}
char _license[] SEC("license") = "GPL";
LLVM eBPF backend (userspace part)

```c
int main(int ac, char **argv)
{
    FILE *f;
    int i, sock;

    /* Load eBPF code and create map */
    load_bpf_file("sockex1_kern.o");
    sock = open_raw_sock("lo");
    setsockopt(sock, SOL_SOCKET, SO_ATTACH_BPF,
               prog_fd, sizeof(prog_fd[0]));

    for (i = 0; i < 5; i++) {
        long long tcp_cnt, udp_cnt, icmp_cnt;
        int key;

        key = IPPROTO_TCP;
        bpf_lookup_elem(map_fd[0], &key, &tcp_cnt);
        key = IPPROTO_UDP;
        bpf_lookup_elem(map_fd[0], &key, &udp_cnt);
        key = IPPROTO_ICMP;
        bpf_lookup_elem(map_fd[0], &key, &icmp_cnt);

        printf("TCP %lld UDP %lld ICMP %lld packets\n",
                tcp_cnt, udp_cnt, icmp_cnt);
        sleep(1);
    }
    return 0;
}
```

# ./sockex1
TCP 0 UDP 0 ICMP 0 packets
TCP 0 UDP 0 ICMP 4 packets
TCP 0 UDP 0 ICMP 8 packets
TCP 0 UDP 0 ICMP 12 packets
TCP 0 UDP 0 ICMP 16 packets
Other BPF and eBPF exploiters

- **xt_bpf**: Kernel module for iptables
  - Match rules with BPF

- **Traffic control:**
  - **cls_bpf**: Kernel module for classifier for traffic shaping
  - **act_bpf**: BPF based action (since Linux 4.1)

- **seccomp-bpf**: Secure computing (since Linux 3.5)
  - Syscall filter for sandboxes (e.g. chromium browser or docker)

- **Kprobes support** (Linux 4.1)
Availability for Linux on z Systems

- **Kernel: BPF**
  - Upstream: Linux 3.7
  - SLES12.0
  - RHEL7.0

- **Kernel: eBPF**
  - Upstream: Linux 4.1

- **LLVM**
  - Upstream: 3.7
  - git commit: ac73683b1 ("[bpf] add big- and host- endian support")
Thank you!
Image sources

- OK (Public Domain): https://openclipart.org/detail/23156/ok-by-dholler
- Computer (Public Domain): https://openclipart.org/detail/25340/Computer_1-by-And
- Wizard (IBM): https://ibm.biz/BdHd9t
- White Board (Public Domain): https://openclipart.org/detail/32389/white-board
- LLVM logo dragon (Apple Inc.): http://llvm.org/Logo.html
- BPF CPU (Public Domain): https://openclipart.org/detail/28105/processoractive
- Flower (Public Domain): https://openclipart.org/detail/172600/geraldton-wav
- Text Bubble (Public Domain): https://openclipart.org/detail/48421/talk-bubble