Developing Stateful Middleboxes with the mOS API

KYOUNGSOO PARK & YOUNGGYOUN MOON

ASIM JAMSHED, DONGHWI KIM, & DONGSU HAN

SCHOOL OF ELECTRICAL ENGINEERING, KAIST
Network Middlebox

Networking devices that provide extra functionalities
- Switches/routers = L2/L3 devices
- All others are called middleboxes

- NAT
- Firewalls
- Web/SSL proxies
- L7 protocol analyzers

IDS/IPS

mOS networking stack
Middleboxes are Increasingly Popular

Middleboxes are ubiquitous
- Number of middleboxes \(\approx\) number of routers (Enterprise)
- Prevalent in cellular networks (e.g., NAT, firewalls, IDS/IPS)
- Network functions virtualization (NFV)
- SDN controls routing through network functions

Provides key functionalities in modern networks
- Security, caching, load balancing, etc.
- Because original Internet design lacks many features
Most Middleboxes Deal with TCP Traffic

- TCP dominates the Internet
  - 95+% of traffic is TCP [1]
- Flow-processing middleboxes
  - Stateful firewalls
  - Protocol analyzers
  - Cellular data accounting
  - Intrusion detection/prevention systems
  - Network address translation
  - And many others!

TCP state management is complex and error-prone!

Example: Cellular Data Accounting System

Custom middlebox application
No open-source projects
Develop Cellular Data Accounting System

For every IP packet, p
sub = FindSubscriber(p.srcIP, p.destIP);
sub.usage += p.length;

For every IP packet, p
if (p is not retransmitted){
    sub = FindSubscriber(p.srcIP, p.destIP);
    sub.usage += p.length;
}

For every IP packet, p
if (p is not retransmitted){
    sub = FindSubscriber(p.srcIP, p.destIP);
    sub.usage += p.length;
} else { // if p is retransmitted
    if (p’s payload != original payload) {
        report abuse by the subscriber;
    }
}

Charge for retransmission?
TCP tunneling attack? [NDSS’14]
Logically, simple process!

mOS networking stack
Cellular Data Accounting Middlebox

Core logic
- Determine if a packet is retransmitted
- Remember the original payload (e.g., by sampling)
- Key: TCP flow management

How to implement?
- Borrow code from open-source IDS (e.g., Snort/Suricata)
- Problem: 50~100K code lines tightly coupled with their IDS logic

Another option?
- Borrow code from open-source kernel (e.g., Linux/FreeBSD)
- Problem: kernel is for one end, so it lacks middlebox semantics

What is the common practice? state-of-the-art?
- Implement your own flow management
- Problem: repeat it for every custom middlebox
Programming TCP End-Host Application

Typical TCP end-host applications

- Berkeley socket API
  - Nice abstraction that separates flow management from application
  - Write better code if you know TCP internals
  - Never requires you to write TCP stack itself

Typical TCP middleboxes?

- Middlebox logic
- Packet processing
- Flow state tracking
- Flow reassembly
- Spaghetti code?

No clear separation!

Berkeley socket API

- Middlebox logic
- Packet processing
- Flow state tracking
- Flow reassembly
- Spaghetti code?

No clear separation!
mOS Networking Stack

Reusable networking stack for middleboxes
- Programming abstraction and APIs to developers

Key concepts
- Separation of flow management from custom logic
- Event-based middlebox development (event/action)
- Per-flow flexible resource consumption

Benefits
- Clean, modular development of stateful middleboxes
- Developers focus on core logic rather than flow management
- High performance flow management on mTCP stack
Key Abstraction: mOS Monitoring Socket

Represents the middlebox viewpoint on network traffic
- Monitors both TCP connections and IP packets
- Provides similar API to the Berkeley socket API

Separation of flow management from custom middlebox logic!
Key Abstraction: mOS Event

Notable condition that merits middlebox processing
- Different from TCP socket events

Built-in event (BE)
- Events that happen naturally in TCP processing
- e.g., packet arrival, TCP connection start/teardown, retransmission, etc.

User-defined event (UDE)
- User can define their own event
- UDE = base event + boolean filter function
  - Raised when base event triggers and filter evaluates to TRUE
  - Nested event: base event can be either BE or UDE
  - e.g., HTTP request, 3 duplicate ACKs, malicious retransmission

Middlebox logic = a set of $<$event, event handler$>$ tuples
Sets up a traffic filter in Berkeley packet filter (BPF) syntax
Defines a user-defined event that detects an HTTP request
Uses a built-in event that monitors each TCP connection start event
UDE Filter Function

Called whenever the base event is triggered
If it returns TURE, UDE callback function is called

```c
static bool chk_http_request(mctx_t m, int sock, int side, event_t event)
{
    struct httpbuf *p;
    u_char* temp; int r;

    if (side != MOS_SIDE_SVR) // monitor only server-side buffer
        return false;
    if ((p = mtcp_get_uctx(m, sock)) == NULL) {
        p = calloc(1, sizeof(struct httpbuf));   // user-level structure
        mtcp_set_uctx(m, sock, p);
    }
    r = mtcp_peek(m, sock, side, p->buf + p->len, REQMAX - p->len - 1);
    p->len += r; p->buf[p->len] = 0;
    if ((temp = strstr(p->buf, "\n\n")) || (temp = strstr(p->buf, "\r\n\r\n"))) {
        p->reqlen = temp - p->buf;
        return true;
    }
    return false;
}
```
Current mOS stack API

Socket creation and traffic filter

```c
int mtcp_socket(mctx_t mctx, int domain, int type, int protocol);
int mtcp_close(mctx_t mctx, int sock);
int mtcp_bind_monitor_filter(mctx_t mctx, int sock, monitor_filter_t ft);
```

User-defined event management

```c
event_t mtcp_define_event(event_t ev, FILTER filt);
int mtcp_register_callback(mctx_t mctx, int sock, event_t ev, int hook, CALLBACK cb);
```

Per-flow user-level context management

```c
void * mtcp_get_uctx(mctx_t mctx, int sock);
void mtcp_set_uctx(mctx_t mctx, int sock, void *uctx);
```

Flow data reading

```c
ssize_t mtcp_peek(mctx_t mctx, int sock, int side, char *buf, size_t len);
ssize_t mtcp_ppeek(mctx_t mctx, int sock, int side, char *buf, size_t count, off_t seq_off);
```
Current mOS stack API

Packet information retrieval and modification

```c
int mtcp_getlastpkt(mctx_t mctx, int sock, int side, struct pkt_info *pinfo);

int mtcp_setlastpkt(mctx_t mctx, int sock, int side, off_t offset, byte *data, uint16_t datalen, int option);
```

Flow information retrieval and flow attribute modification

```c
int mtcp_getsockopt(mctx_t mctx, int sock, int l, int name, void *val, socklen_t *len);

int mtcp_setsockopt(mctx_t mctx, int sock, int l, int name, void *val, socklen_t len);
```

Retrieve end-node IP addresses

```c
int mtcp_getpeername(mctx_t mctx, int sock, struct sockaddr *addr, socklen_t *addrlen);
```

Per-thread context management

```c
mctx_t mtcp_create_context(int cpu);

int mtcp_destroy_context(mctx_t mctx);
```

Initialization

```c
int mtcp_init(const char *mos_conf_fname);
```
mOS Stack Internals

• mOS networking stack internals
  • Shared-nothing parallel architecture
  • Dual-stack fine-grained flow management
  • Fine-grained resource management
  • Event generation and processing
  • Scalable event management

More details in our NSDI 2017 paper:

“mOS: A Reusable Networking Stack for Flow Monitoring Middleboxes”
Challenges & Lessons Learned

• Key challenge - ambitious goal
  • Seek for abstraction that applies to ALL kinds of complex middleboxes
  • Original idea includes tight L4-L7 integration (proxy socket, extended-epoll, etc.)
  • Took us 4 years, ~30K lines of code, lots of trial and errors, etc.

• Solution 1 – well-defined set of API is the key
  • Experience with the well-established API – mTCP [NSDI14]
  • Focus on intra-L4 abstraction – state tracking, flow reassembly, flexible events

• Solution 2 – learn from real-world applications
  • Convince ourselves with application to real middleboxes
  • Wrote 7-8 real applications (Snort, cellular accounting system, NAT, firewalls, …)

• Solution 3 – feedback from industry
  • Talks at DPDK summit – precious feedback from daily developers
  • Actively respond to queries
mOS Applications Demo
Goal

Demonstrate benefits of mOS API in real-world applications

- L4 proxy for fast packet loss recovery (mHalfback)
- L7 protocol analyzer (mPRADS)
- L4 load balancer (mOS L4-LB)
mHalfback

L4 proxy for fast packet loss recovery
Halfback \textit{[CoNEXT ’15]}

A transport-layer scheme for optimizing flow completion time (FCT)

**Key idea**
- Skips TCP slow start phase to pace up transmission rate at start
- Performs proactive retransmission for fast packet loss recovery
mHalfback

A middlebox that transparently reduces FCT w/o modifying end hosts

Core logic

◦ 1) For each TCP data packet arrival, hold a copy of the packet
◦ 2) When an ACK packet comes from the receiver, retransmit a data packet

Sender

mHalfback

Receiver

packet loss recovered
Implementing mHalfback using mOS

Core logic

For every IP packet, \( p \)

Where is \( p \) from?

- \( p \)'s payload size > 0?
  - yes
    - Enqueue \( p \) \( \rightarrow \) \( d \)
    - Retransmit \( d \) to client

- \( p \) a ACK packet?
  - yes
    - mHalfback requires only \( \sim 120 \) LoCs using mOS API

mOS code

```c
msock = mtcp_socket(mctx, AF_INET, MOS.SOCK_MONITOR_STREAM, 0);

/* enqueue TCP data packets sent from server side */
svr_ev = mtcp_define_event(MOS_ON_PKT_IN, IsFromServer, NULL);
data_ev = mtcp_define_event(svr_ev, HasPayload, NULL);
mtcp_register_callback(mctx, s, data_ev, MOS_HK_SND, EnqueueTCPDataPacket);

/* for each ACK, retransmit a TCP data packet to client */
cli_ev = mtcp_define_event(MOS_ON_PKT_IN, IsFromClient, NULL);
ack_ev = mtcp_define_event(cli_ev, IsACKPacket, NULL);
mtcp_register_callback(mctx, s, ack_ev, MOS_HK_RCV, ProactiveRetransmit);
```
mHalfback Evaluation

Environment

- Server runs a *nginx* server (v1.4.6)
- Client runs *Apache benchmark* (v2.3) to download a 200KB file
  - Placed a packet dropper in front of client to simulate a lossy link
  - Run the experiment with/without mHalfback in between the server and client
mHalfback Demo

- Direct connection (without mHalfback)
mHalfback Demo

- Connection via mHalfback

```
server@tree1:~$ qdisc netem 1: root refcnt 65 limit 1000 delay 60.0ms
server@tree1:~$ 

mhalfback@tree2:/home/ygmoon/trunk/mos/mos_server

# application to run (abacus)
application {
    type = monitor
    run = halfback
}

mhalfback@tree2:/home/ygmoon/trunk/mos/mos_server$ emasc ^C
mhalfback@tree2:/home/ygmoon/trunk/mos/mos_server$ sudo emacs apps/halfback/halfback.c

packet_dropper@tree3:/home/ygmoon/trunk/mos/mos_server/apps/micro/packet_dropper
[ ALL ] dpdk0, RX: 1(pps) (err: 0), 0.00(Gbps), TX: 0(pps), 0.00(Gbps)
[ ALL ] flows: 0
[ PEAK ] RX: 0.00(Gbps), TX: 0.00(Gbps)
[ RECENT AVG ] RX: 0.00(Gbps), TX: 0.00(Gbps)

client@tree4:~$ 
```
mHalfback Performance

20% to 41% FCT reduction under 5% packet loss
- Without any modification on the end hosts
mPRADS
Application-layer protocol analyzer
Passive Real-time Asset Detection System (PRADS)

A passive fingerprinting tool for gathering host and service information.

It performs PCRE pattern matching on TCP packets to detect:
- Type of OSes
- Server/client applications (nginx, Apache, Mozilla, Chronium, …)
- Web application type (WordPress, Drupal, …)

Example output:

```
10.0.0.6, [client: Wget/1.15 (linux (gnu)):80:6], [distance:0]
10.0.0.1, [service: nginx 1.4.6:80:6], [distance:0]
-- Total TCP service assets detected : 1
-- Total TCP client assets detected : 1
```
Limitation in PRADS (Demo)

Its PCRE module cannot detect a pattern that spans over multiple packets.

```
Packet 1
“... ...\r\nServer: ng”
```

```
Packet 2
“inx/1.4.6 (Ubuntu)\r\n...”
```

```
tcpdump

Server: ng
length 558
E..b..@.@.$.
...
........U\..........@........
~...4.A.inx/1.4.6 (Ubuntu)
Connection: Keep-Alive
```

```
PRADS

[*] Sniffing...
10.0.0.6,[syn:S20:64:1:60:M1460,S,T,N,W7:.],,[unknown:unknown],[link:ethernet/modem],[uptime:2
10.0.0.6,[client:Wget/1.15 (linux (gnu)):8080:6],[distance:0]
```

```
wget

Downloaded: 1 files, 512 in 0s (58.7 MB/s)
ygmoon@tree6:~$```

```
mPRADS Demo

We port PRADS to mOS to verify the correctness of mOS-based apps.

→ mPRADS detects the pattern over flow-reassembled data correctly.
Benefits of mOS Porting

1. mOS API hides the details of TCP flow management
   - mPRADS doesn’t have to care the complex payload reassembly internals

```c
g_ev_peek = mtcp_define_event(MOS_ON_CONN_NEW_DATA, ft_peek, NULL);

bool ft_peek(mctx_t mctx, int sock, int side, event_t ev, filter_arg_t *arg) {
    int ret;
    if ((ret = mtcp.peek(mctx, sock, side, buf + len, MAX_BUFLEN - len)) > 0)
        len += ret;
    return (len >= MIN_THRESH);
}
```

2. mOS encourages code reuse of common L4-L7 processing
   - A well-designed set of event definitions can be shared across different apps

   Applications: mPRADS, mSnort-IDS

   UDE library: UDE_ON_HTTP_HDR

   mOS

   shared

mOS L4-LB

Highly-scalable L4 load balancer
Implementing L4 LB with mOS

- mOS provides monitoring/manipulation APIs for packet-level apps
  - mOS L4 LB with 5 balancing algorithms = ~ 200 lines of code
  ```c
  if (mtcp_register_callback(mctx, lsock, MOS_ON_PKT_IN, MOS_HK_SND, translate_addr))
      exit(EXIT_FAILURE);

  mtcp_setlastpkt(mctx, sock, 0, off_port,
                  (uint8_t *)&port, sizeof(in_port_t),
                  MOS_TCP_HDR | MOS_OVERWRITE | MOS_UPDATE_TCP_CHKSUM);
  ```

- mOS adopts shared-nothing threading model for core scalability
  - L4-LB runs symmetric RSS to pre-compute ports available to each core
  ```c
  if (mtcp_get_rss_core(mctx, &ph, &binfo->addr) == cpu)
      *b = binfo->addr;
  ```

- Flow reassembly buffer can be disabled if the mOS app doesn’t need it
  ```c
  int opt = 0;
  mtcp_setsockopt(mctx, sock, SOL_MONSOCKET, MOS_CLIBUF, &opt, sizeof(opt));
  ```
**mOS L4-LB Evaluation**

**Environment**
- mOS L4-LB runs round-robin LB algorithm
  - Intel Xeon E5-2697v3 (14 cores @ 2.60GHz) x2, 35 MB L3 cache size
  - 128 GB RAM, 4 x 10 Gbps NICs
- Four pairs of clients and servers: 40 Gbps max
  - Intel E3-1220 v3 (4 cores, 3.1 GHz), 8 MB L3 cache size
  - 16 GB RAM, 1 x 10 Gbps NIC per machine
  - Each runs a mTCP-based web server/client with (4K x 4 = total 16K concurrent flows)
mOS L4-LB Demo

- 16,000 concurrent flows in total each downloading 4KB files

<table>
<thead>
<tr>
<th>mOS L4-LB</th>
<th>Clients</th>
<th>Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>run-cores: 16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nb_mem_channels: 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max_concurrency: 100000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>neko-size: 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>window_size: 8192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcp_twInterval: 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tcp_twTimeout: 30000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>multiprocess: false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mos_log: logs/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sflash_print: dpdk0 dpdk1 dpdk2 dpdk3 forward</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

--- Netdev configuration (4 entries) ---

dpdk0 (id: 0, HADDR: 00:18:21:84:73:78) maps to CPU 0x0000000000000000

dpdk1 (id: 1, HADDR: 00:18:21:84:73:78) maps to CPU 0x0000000000000000

dpdk2 (id: 2, HADDR: 00:18:21:84:73:78) maps to CPU 0x0000000000000000

dpdk3 (id: 3, HADDR: 00:18:21:84:73:78) maps to CPU 0x0000000000000000

--- Static ARP table configuration (10 entries) ---

- 0x00A0000000, NETMASK: 0xFFFFFFF, HADDR: 0x00:00:00:00:00:00:00
- 0x00A0000000, NETMASK: 0xFFFFFFF, HADDR: 0x00:00:00:00:00:00:00
- 0x00A0000000, NETMASK: 0xFFFFFFF, HADDR: 0x00:00:00:00:00:00:00
- 0x00A0000000, NETMASK: 0xFFFFFFF, HADDR: 0x00:00:00:00:00:00:00

--- Routing table configuration (6 entries) ---

- 0x00A0000000, INTERFACE: eth0 (id: 0)
- 0x00B0000000, INTERFACE: eth0 (id: 0)
- 0x00C0000000, INTERFACE: dpdk (id: 0)
- 0x00D0000000, INTERFACE: dpdk (id: 1)
- 0x00E0000000, INTERFACE: dpdk (id: 2)
- 0x00F0000000, INTERFACE: dpdk (id: 3)

--- NIC Forwarding table configuration (0 entries) ---

| NIC Forwarding Index Table: |
| 0: 0 |
| 1: 0 |
| 2: 0 |
| 3: 0 |
| 4: 0 |
| 5: 0 |
| 6: 0 |
| 7: 0 |
| 8: 0 |
| 9: 0 |
| 10: 0 |
| 11: 0 |
| 12: 0 |
| 13: 0 |
| 14: 0 |
| 15: 0 |

configuration updated by mtpc_setconf()
mOS L4-LB Demo

- 16,000 concurrent flows in total each downloading 4KB files

### 4 backend servers

<table>
<thead>
<tr>
<th>Flow Rate</th>
<th>RX</th>
<th>TX</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.50(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.42(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1524175(pps) (err: 0), 1.29(Gbps), TX: 1528183(pps), 8.51(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3301</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.51(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.46(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1513531(pps) (err: 0), 1.28(Gbps), TX: 1518395(pps), 8.46(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3317</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.51(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.48(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1523579(pps) (err: 0), 1.29(Gbps), TX: 1527150(pps), 8.50(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3315</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.51(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.29(Gbps), TX: 8.47(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1517552(pps) (err: 0), 1.29(Gbps), TX: 1524040(pps), 8.47(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3314</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.51(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.40(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1514835(pps) (err: 0), 1.28(Gbps), TX: 1517122(pps), 8.45(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3479</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.47(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.42(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1514038(pps) (err: 0), 1.28(Gbps), TX: 1513636(pps), 8.43(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.47(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.42(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1519864(pps) (err: 0), 1.29(Gbps), TX: 1522003(pps), 8.48(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 3288</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.48(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.45(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1511796(pps) (err: 0), 1.28(Gbps), TX: 1513610(pps), 8.43(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>flows: 2947</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEAK</td>
<td>1.29(Gbps), TX: 8.48(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RECENT AVG</td>
<td>1.28(Gbps), TX: 8.44(Gbps)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>dpdk0, RX: 1511796(pps) (err: 0), 1.28(Gbps), TX: 1513610(pps), 8.43(Gbps)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
mOS L4-LB Demo

- 16,000 concurrent flows in total each downloading 4KB files

mOS L4-LB

```
IP: 0x0A0000200, NETMASK: 0xFFFFFFFF00, INTERFACE: dpdk2(idx: 2)
IP: 0x0A0000300, NETMASK: 0xFFFFFFFF00, INTERFACE: dpdk3(idx: 3)

+-----+-----------------------------+
|     | NIC Forwarding table configuration (0 entries)     |
|     | NIC Forwarding Index Table: | |
| 0   | --+-- | 1 |
| 1   | --+-- | 1 |
| 2   | --+-- | 1 |
| 3   | --+-- | 1 |
| 4   | --+-- | 1 |
| 5   | --+-- | 1 |
| 6   | --+-- | 1 |
| 7   | --+-- | 1 |
| 8   | --+-- | 1 |
| 9   | --+-- | 1 |
| 10  | --+-- | 1 |
| 11  | --+-- | 1 |
| 12  | --+-- | 1 |
| 13  | --+-- | 1 |
| 14  | --+-- | 1 |
| 15  | --+-- | 1 |
+-----+-----------------------------+

Configuration updated by mtcp_setconf().
[LoadConfigData: 206] scanning config/lb.yaml..
[LoadConfigData: 331] finished parsing config/lb.yaml (found 1 proxies)
```
mOS L4-LB Performance

Core scalability (file size: 4KB)
- Performs 6~7x better than haproxy

Varying file size (using 16 CPU cores)
- Performs 6~10x better than haproxy
Wrap-up: mOS Applications Demo

Developing or extending L4-L7 stateful middleboxes was difficult

- Due to lack of reusable networking stack for middleboxes

We demonstrated that mOS eases development of diverse apps

- mHalfback → intuitive flow-level abstractions for middleboxes
- mPRADS → robust payload reassembly, code reusability
- mOS L4-LB → performance scalability
Thank you!

mOS code and programming guide are available!

https://mos.kaist.edu/