mOS

NSDI2017

Simple packet processing

example.cc

```
void simple_packet_processing_program(){
  packet* pkts[32];

  // an infinite loop.
  for(;;){
    int count = fetch_packets(pkts, 32);

    // the processing loop.
    for(int i=0; i<count; i++){
      process_packet(pkts[i]);
    }

    // the releasing loop
    for(int i=0; i<count; i++){
      release_packet(pkts[i]);
    }
  }
}
```

Complicated stream processing

```c
static bool
IsFakeRexmit(mctx_t mctx, int sock, int side, event_t event,
struct filter_arg *arg) {
  struct pkt_info pi;
  char buf[MSS];
  struct tcp_ring_fragment frags[MAX_FRAG_NUM];
  int nfrags = MAX_FRAG_NUM;
  int i, size, boff, poff;

  // retrieve the current packet information
  mtcp_getlastpkt(mctx, sock, side, &pi);

  // for full retransmission, compare the entire payload
  if (mtcp_ppeek(mctx, sock, side, buf, pi.payloadlen, pi.offset) ==
  pi.payloadlen)
    return memcmp(buf, pi.payload, pi.payloadlen);

  // for partial retransmission, compare the overlapping region
  // retrieve the data fragments and traverse them
  mtcp_getsockopt(mctx, sock, SOL_MONSOCKET, (side == MOS_SIDE_CLI) ?
  MOS_FRAGINFO_CLIBUF : MOS_FRAGINFO_SVRBUF, frags, &nfrags);

  for (i = 0; i < nfrags; i++) {
    if ((size = CalculateOverlapLen(&pi, &(frags[i]), &boff, &poff)))
    if (memcmp(buf + boff, pi.payload + poff, size))
    return true; // payload mismatch detected
  }
  return false;
}
```

An overview of mOS

- Do not directly access and modify packets.
- Hide packet processing logic from the programmer.
- Expose events (**primarily related to TCP**) to the programmer.
- Program middlebox by writing callback functions for events.

The architecture graph

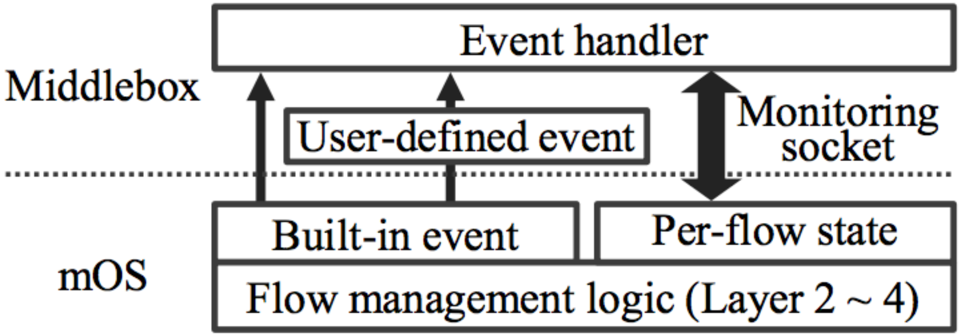
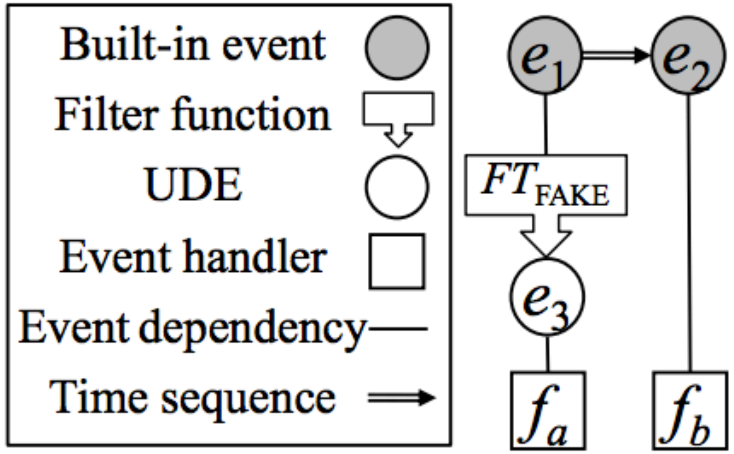


Figure 1: Interaction between mOS and its application

Event action diagram



e_1	MOS_ON_REXMIT
e_2	MOS_ON_CONN_NEW_DATA
e_3	UDE_FAKE_REXMIT
FT_{FAKE}	IsFakeRexmit()
f_a	HandleFakeRexmitPacket()
f_b	AccountDataUsage()

Eight built-in event

Event	Description
MOS_ON_PKT_IN	In-flow TCP packet arrival
MOS_ON_CONN_START	Connection initiation (the first SYN packet)
MOS_ON_REXMIT	TCP packet retransmission
MOS_ON_TCP_STATE_CHANGE	TCP state transition
MOS_ON_CONN_END	Connection termination
MOS_ON_CONN_NEW_DATA	Availability of new flow-reassembled data
MOS_ON_ORPHAN	Out-of-flow (or non-TCP) packet arrival
MOS_ON_ERROR	Error report (e.g., receive buffer full)

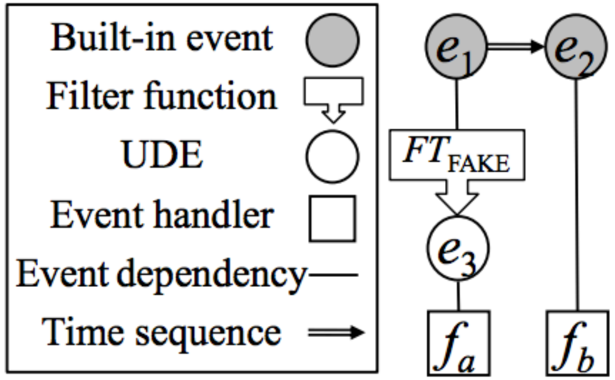
Table 1: mOS built-in events for stream monitoring sockets. Raw monitoring sockets can use only MOS_ON_PKT_IN raised for every incoming packet.

Built-in event triggers filter function

```
static bool
IsFakeRexmit(mctx_t mctx, int sock, int side, event_t event,
struct filter_arg *arg) {
  struct pkt_info pi;
  char buf[MSS];
  struct tcp_ring_fragment frags[MAX_FRAG_NUM];
  int nfrags = MAX_FRAG_NUM;
  int i, size, boff, poff;

  // retrieve the current packet information
  mtcp_getlastpkt(mctx, sock, side, &pi);

  // for full retransmission, compare the entire payload
  if (mtcp_ppeek(mctx, sock, side, buf, pi.payloadlen, pi.offset) ==
  pi.payloadlen)
    return memcmp(buf, pi.payload, pi.payloadlen);

  // for partial retransmission, compare the overlapping region
  // retrieve the data fragments and traverse them
  mtcp_getsockopt(mctx, sock, SOL_MONSOCKET, (side == MOS_SIDE_CLI) ?
  MOS_FRAGINFO_CLIBUF : MOS_FRAGINFO_SVRBUF, frags, &nfrags);

  for (i = 0; i < nfrags; i++) {
    if ((size = CalculateOverlapLen(&pi, &(frags[i]), &boff, &poff)))
    if (memcmp(buf + boff, pi.payload + poff, size))
    return true; // payload mismatch detected
  }
  return false;
}
```

Filter function generates user-defined events

- If the filter function evaluates to true, a user-defined event is generated and the corresponding callback is called.



e_1	MOS_ON_REXMIT
e_2	MOS_ON_CONN_NEW_DATA
e_3	UDE_FAKE_REXMIT
FT_{FAKE}	IsFakeRexmit()
f_a	HandleFakeRexmitPacket()
f_b	AccountDataUsage()

Build event-action diagram.

```
static void
mOSAppInit(mctx_t m)
{
  monitor_filter_t ft = {0};
  int s; event_t hev;

  // creates a passive monitoring socket with its scope
  s = mtcp_socket(m, AF_INET, MOS_SOCK_MONITOR_STREAM, 0);
  ft.stream_syn_filter = "dst net 216.58 and dst port 80";
  mtcp_bind_monitor_filter(m, s, &ft);

  // sets up an event handler for MOS_ON_REXMIT
  mtcp_register_callback(m,s,MOS_ON_REXMIT,MOS_HK_RCV,OnRexmitPkt);

  // defines a user-defined event that detects an HTTP request
  hev = mtcp_define_event(MOS_ON_CONN_NEW_DATA, IsHTTPRequest, NULL);

  // sets up an event handler for hev
  mtcp_register_callback(m, s, hev, MOS_HK_RCV, OnHTTPRequest);
}
```

Figure 2: Initialization code of a typical mOS application. Due to space limit, we omit error handling in this paper.

Per-core, share nothing design

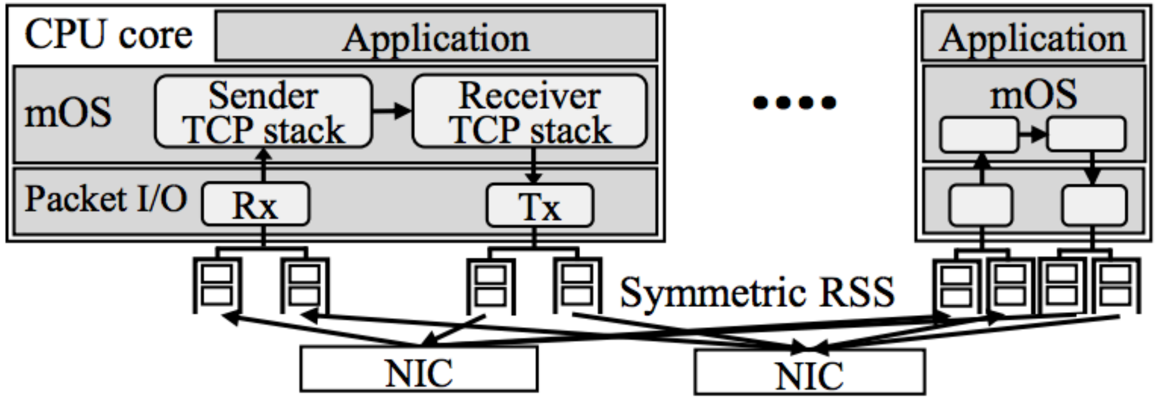


Figure 9: mOS application threading model

Per-flow TCP context

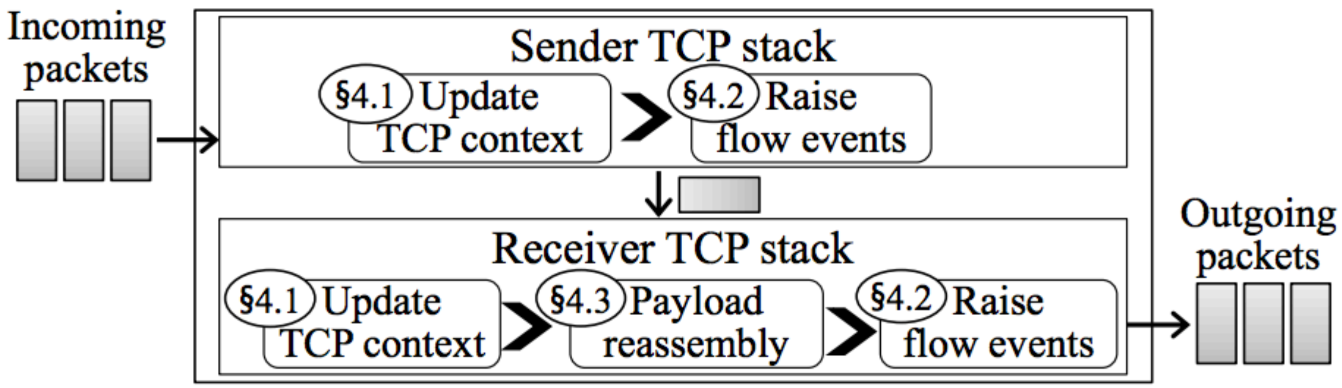


Figure 5: Packet processing steps in mOS

Scalable event management

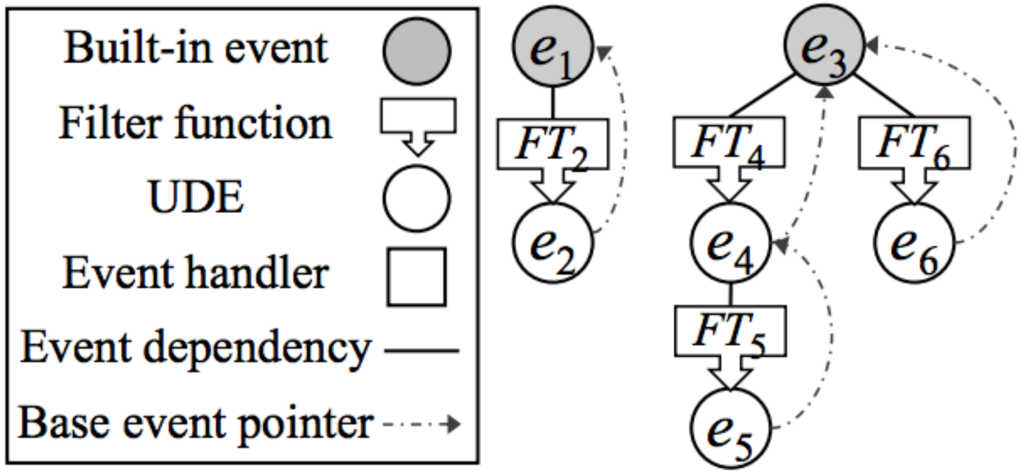
- Problem:

- The flow can dynamically register event by itself.

- Maintaining a dedicated event diagram for each flow introduces huge overheads.

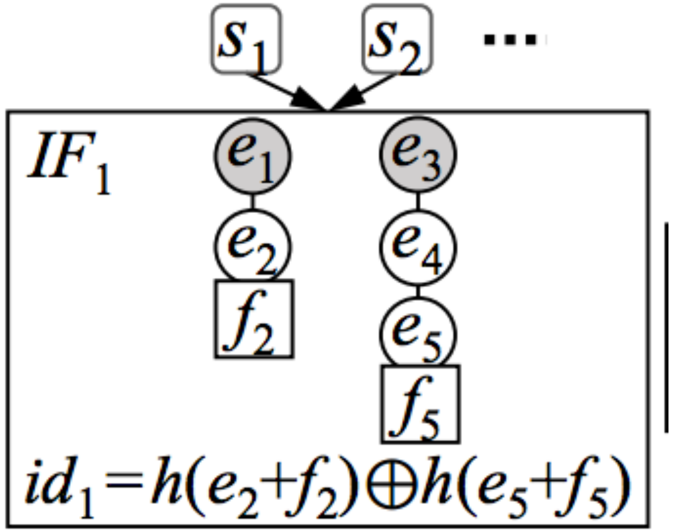
- Solution: Share event diagram, allocate new event diagram when needed.

Global event dependency forest, d-forest

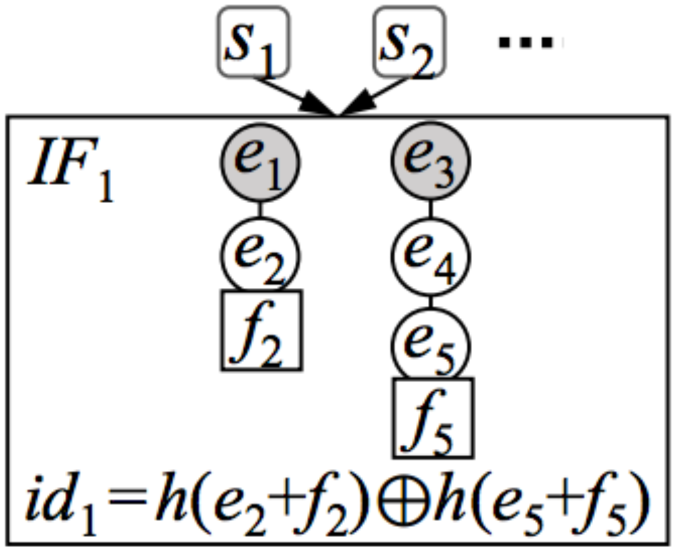


(a) Global event dependency forest (d-forest)

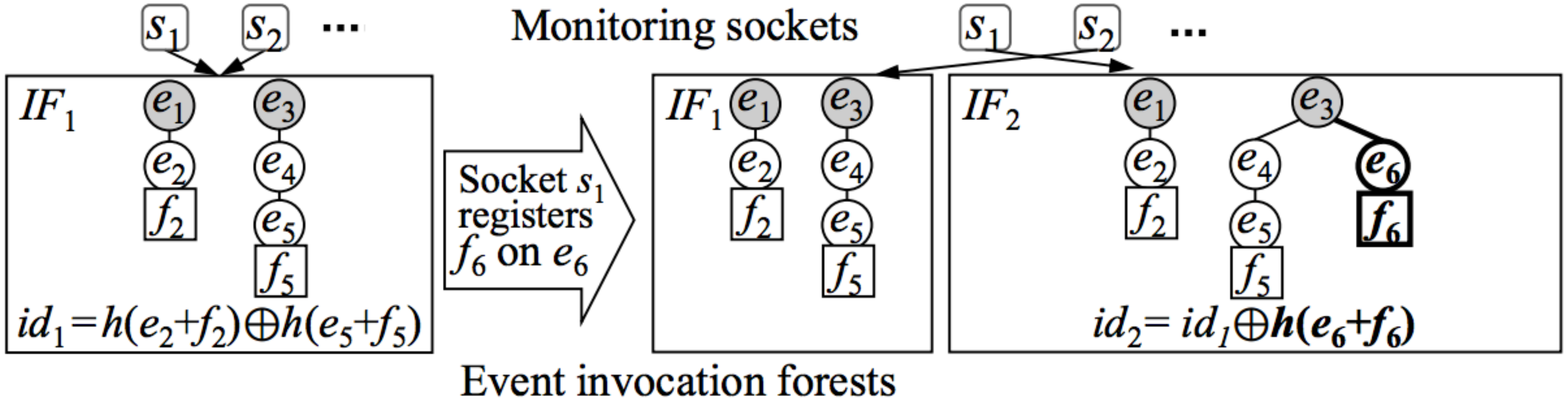
Event invocation forest, i-forest



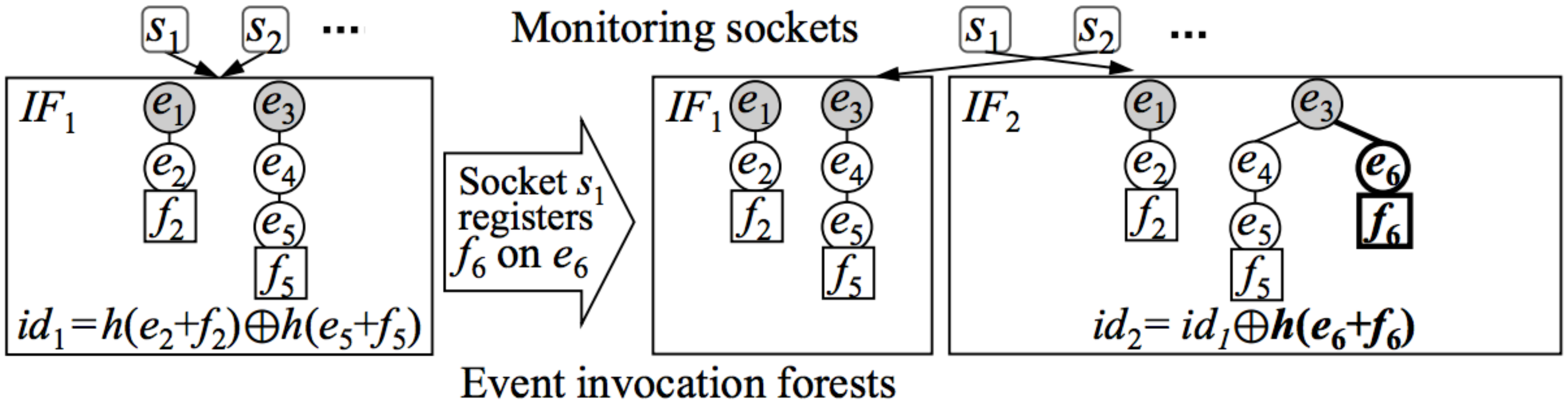
When a new flow context is created



Then the flow may register additional event



Efficient querying of existing i-forest



Port existing middleboxes

- Snort3
- Abacus (re-implement in 400 LoC)
- Halfback
- nDPI.
- PRADs

Performance

Application	original + pcap	original + DPDK	mOS port
Snort-AC	0.51 Gbps	8.43 Gbps	9.85 Gbps
Snort-DFC	0.78 Gbps	10.43 Gbps	12.51 Gbps
nDPIReader	0.66 Gbps	29.42 Gbps	28.34 Gbps
PRADS	0.42 Gbps	2.05 Gbps	2.02 Gbps
Abacus	-	-	28.48 Gbps

Table 3: Performance of original and mOS-ported applications under a real traffic trace. Averaged over five runs.

Micro benchmark

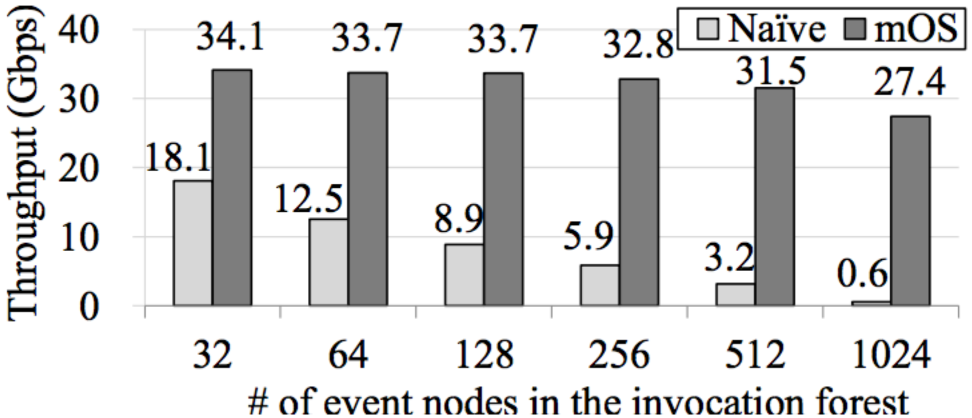


Figure 12: Performance at dynamic event registration