Cache-Friendly IP Reassembly Network Function

Hyunseok Chang, Fang Hao, T.V. Lakshman, Sarit Mukherjee, Limin Wang

NOKIA Bell Labs
Emergence of SD-WAN in Enterprise Networking

**SD-WAN controller** sets up virtual WAN based on various interconnection policies

**SD-WAN gateway** performs various NFs (traffic slicing, steering, enc/dec, filtering)

Traffic encapsulated for network slicing and security

**SD-WAN** traffic is expected to grow to 30% of all WAN traffic by 2022

SD-WAN controller

Software-defined Virtual WAN

Branch Locations

Internet

Public Cloud

Private DC
Challenges Faced by SD-WAN Gateway

- Need to perform various types of NFs within resource-constrained commodity hardware (12-24 cores)

- IP reassembly NF:
  - Must deal with fragmented IP packets due to heavy encapsulation (e.g. VxLAN/NVGRE/STT/IPsec)
  - Existing solutions to prevent IP fragmentation are insufficient (e.g. path MTU discovery, MSS clamping, TSO/GSO, pre-fragmentation)
  - Require efficient stateful packet processing
  - *Is the current design of IP reassembly NF optimal?*
NF Execution Model in Kernel Bypass Paradigm

• **Per-packet run-to-completion**
  • Retrieve input packets through RX API
  • Process each received packet one at a time, up to its forwarding
  • Send pending output packets through TX API

• **Is the model still viable for any complex NF?**

• **Our observation**
  • Run-to-completion model may not be an ideal choice for a complex NF due to *instruction cache pressure*
Impact of Instruction Cache Pressure on NF Performance

- Two separate implementations for the same function

```
function scrambler() {
    while (true) {
        n_rx = recv_pkts(pkts[], MAX)
        for (i = 0; i < n_rx; i++) {
            rewrite_src_1(pkts[i])
            rewrite_src_2(pkts[i])
            rewrite_src_3(pkts[i])
            rewrite_src_4(pkts[i])
        }
        xmit_pkts(pkts[], n_rx)
    }
}

function rewrite_src_i (pkt) {
    pkt→srcIP = hash_i(pkt→srcIP)
}
```

```
function scrambler2() {
    while (true) {
        n_rx = recv_pkts(pkts[], MAX)
        for (i = 0; i < n_rx; i++) {
            rewrite_src_1(pkts[i])
            rewrite_src_2(pkts[i])
            rewrite_src_3(pkts[i])
            rewrite_src_4(pkts[i])
        }
        xmit_pkts(pkts[], n_rx)
    }
}
```

Single Loop

Split Loops
(Loop fission)
• Due to instruction cache pressure, **per-stage batch processing** can outperform **per-packet run-to-completion** for a complex NF.
State-of-the-Art IP Reassembly NF

- Per-packet processing composed of complex nested function calls with side effect → Not instruction-cache friendly
- Can we re-architect it as multi-stage packet processing?

```c
while (true) {
    API_RX (pkts[32])
    for each pkt in pkts[] {
        If (API_fragmented (pkt))
            pkt = API_reassemble (pkt)
        API_lookup_lpm (pkt)
        API_TX (pkt)
    }
    API_free()
}

API_reassemble (pkt) {
    entry = frag_find (pkt)
    p = frag_process (pkt, entry)
    return p
}

frag_find (pkt) {
    entry = frag_lookup (pkt)
    cleanup (entry) if expired
    return entry
}

frag_process (pkt, entry) {
    update_tbl (pkt, entry)
    if (full (entry) {
        p = reassemble (entry)
        return p
    })
```
Multi-Stage IP Reassembly

- Burst RX retrieves a large batch of packets
- Each stage performs more granular processing logic against the large batch
- Reassembly processing and frag-table cleanup logic completed decoupled
- In-order delivery achieved by reassembling packets in pkts[] in place
Prefetch-friendly Frag-Table Lookup

- Existing frag-table lookup is not data cache efficient

<table>
<thead>
<tr>
<th>Frag-key1</th>
<th>Frag-data1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frag-key2</td>
<td>Frag-data2</td>
</tr>
<tr>
<td>Frag-key3</td>
<td>Frag-data3</td>
</tr>
<tr>
<td>Frag-key4</td>
<td>Frag-data4</td>
</tr>
</tbody>
</table>

75% of cache line is wasted during iterative frag-key lookup

- Linear Probing

64B cache line

Four frag-keys can be loaded into a cache line at once
Performance Evaluation

- Line-rate processing for up to 20% fragmented traffic
- +30% packet processing performance improvement
- Multi-stage processing is the main performance boosting factor (+20%)
Conclusion

- Per-packet run-to-completion model may not be desirable for a complex NF due to instruction cache pressure
- Cache-friendly IP reassembly NF enabled by multi-stage processing, loop unrolling, and prefetch-friendly data structure
- Future works
  - Profile per-packet execution paths and cache performance of other types of NFs
  - Generalized tool that can guide code optimization
Thank you!