Parallelizing Match Action Tables With P4

Ryan Massicci | Dr. Minseok Kwon

P4 Programming Language

The P4 programming language is intended for programming data plane applications for packet forwarding in a software defined network. The developers of P4 had three key goals in mind when designing the language: Target independence, protocol independence, and reconfigurability. A fundamental concept in the P4 language is the pipeline. The P4 pipeline consists of the parser, metadata bus, match-action tables, and deparser [1].

P4 Algorithms

Three separate P4 programs, with increasing complexity, have been analyzed and compiled. IPv4 forwarding, multi-hop route inspection, and address resolution/internet control message protocol messaging algorithms have been implemented. Each algorithm has been run in a single core and dual core environment. Their sequential and parallel runtimes of each program has been obtained, and the speedup calculated. The factors that prevented or contributed to the parallel speedup is discussed.

IPv4 Forwarding

IPv4 Forwarding is a simple program that forwards a single packet from one host to another. The program updates the source and destination addresses in the packet header, updates the output port, and decrements the TTL.

```
ipv4_lpm {
  key = { hdr.ipv4.dstAddr; }
  actions = { ipv4_forward; drop; NoAction; }
}
```

Multi-Hop Route Inspection

Multi-Hop Route Inspection builds upon IPv4 forwarding by adding route tracing functionality. Each hop throughout the network adds an identifier to a list. At its destination, the packet has an ordered list of identifiers.

```
ipv4_lpm {
  key = { hdr.ipv4.dstAddr; }
  actions = { ipv4 Forward; }
}

swid {
  actions = { add_swid; }
}
```

ARP/ICMP Messaging

ARP/ICMP builds on the IPv4 forwarding by adding ping functionality. A host machine can send a request to another host connected to the same switch, and the switch will respond to the ping request.

```
ipv4_lpm {
  key = { ipv4_lpm; }
  actions = { set_classes; }
}

table forward {
  key = { arp.isValid; icmp.isValid; }
  actions = { ipv4 forward; send_arp_reply; send_icmp_reply };
}
```

Timing Analysis

The runtime of each program was analyzed on two environments. In the first environment, each program was run on a system with a single core. The second environment enabled a second core on the host machine. Each program forwarded 10,000 packets between the virtual hosts with the data plane rules determined by the P4 program. The transfer speed was determined by taking the average speed of the 10,000 packets reported by Wireshark.

```
<table>
<thead>
<tr>
<th></th>
<th>Synchronous</th>
<th>Parallel</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPv4 Forward</td>
<td>.54 ms</td>
<td>.53 ms</td>
<td>2%</td>
</tr>
<tr>
<td>MHR</td>
<td>.63 ms</td>
<td>.55ms</td>
<td>12%</td>
</tr>
<tr>
<td>ARP/ICMP</td>
<td>.76ms</td>
<td>.67ms</td>
<td>12%</td>
</tr>
</tbody>
</table>
```

Recommendations

When possible, the programmer should avoid any table dependencies. It may be possible to move specific actions out of one table and into another in order to avoid table dependencies. If a table dependency is unavoidable, the programmer should prefer an action dependency over a match dependency. The programmer should also be mindful of including too many actions in a single table.

Resources