P4 Pub/Sub

Practical Publish-Subscribe in the Forwarding Plane
Outline

• Address-oriented routing
• Publish/subscribe
• How to do pub/sub in the network
• Implementation status
• Outlook
Publish/Subscribe

- Publishers send messages tagged with “topics”
- Subscribers subscribe to messages for certain topics
How is Pub/Sub Implemented?

- Uses message brokers like Apache Kafka
- Run in software = slow
If the Network Could Route Pub/Sub…

Fewer Messages = Higher Throughput

No Broker = Lower Latency
Can We Do Pub/Sub with PISA?

Yes! But how?

1. Define a language that characterizes pub/sub semantics

2. Create a model that abstracts routing decision

3. Compile the model to run on a switch
The Camus Language
Queries

• A query is a formula of atomic predicates

• Each atomic predicate checks a field

• Examples:
  
  • topic="hiking" ∨ topic="skiing"

  • price>30 ∧ (stock="IBM" ∨ stock="BFN")
The Compiler

Protocol Specification

```plaintext
fields {
    shares: 32 bits;
    stock: 64 bits;
    price: 32 bits;
}
```

Rules

```plaintext
shares<100: 2;
stock="IBM": 4;
price>10 ∧ shares...
```
The Decision Model
Binary Decision Diagrams (BDD)

• Each node checks an atomic predicate
• Take high branch if true, otherwise take low branch
• Leaf nodes are the set of actions to be applied
Constructing a BDD

• E.g. add formula $x > 1 \land z = 5$ to existing BDD

This eval does not change the formula
Reducing the BDD

- E.g. BDD for “\(x>1 \land y=2\)” and “\(z=5\)”:

Reduction #1: non-unique nodes
Reducing the BDD
Reducing the BDD

Reduction #2: identical children
Reducing the BDD

- $x > 1$
- $y = 2$
- $z = 5$

- [ ]
- [2]
- [1]
- [1,2]
Compiling the Queries
Translating BDD to Tables

1. Do a graph cut, where each subgraph only contains predicates on the same field

- **x fields**
  - $x > 1$
  - $x = 3$

- **z fields**
  - $z = 5$
  - $z = 5$

- **leaves**
  - [ ]
  - [2]
  - [3]
  - [1,3]
  - [1,2,3]
Identify “entry nodes”

2. For each field, find the nodes that have a parent node that does not belong to the same field

- x entry node
- z entry nodes
- Assign an ID

x > 1

x = 3

z = 5

[ ] [2] [1,3] [1,2,3]
Find Paths for Each Field

3. For each entry node, enumerate all the paths until the next field.
4. For each path, get the most conservative match (or * if nothing matched)
5. For each path, the tuple (entry ID, match, exit ID) corresponds to an entry in its field’s table.
Putting the Tables Together

**x table**

<table>
<thead>
<tr>
<th>State Match</th>
<th>Match</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>=3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>&gt;1</td>
<td>6</td>
</tr>
</tbody>
</table>

**z table**

<table>
<thead>
<tr>
<th>State Match</th>
<th>Match</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>*</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>=5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>*</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>=5</td>
<td>8</td>
</tr>
</tbody>
</table>

**Actions**

<table>
<thead>
<tr>
<th>State</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>[]</td>
</tr>
<tr>
<td>5</td>
<td>[2]</td>
</tr>
<tr>
<td>6</td>
<td>[3]</td>
</tr>
<tr>
<td>7</td>
<td>[1,3]</td>
</tr>
<tr>
<td>8</td>
<td>[1,2,3]</td>
</tr>
</tbody>
</table>
Example: Filtering Packets

{x: 0, z=7}
{x: 3, z=5}
{x: 2, z=5}
Does It Fit?

Limited number of stages

Limited memory per stage

Range matches are more expensive
Number of Predicates vs. Forwarding Rules

More predicates $\rightarrow$ more selective queries $\rightarrow$ fewer overlapping queries $\rightarrow$ smaller tables
Applications
Nasdaq ITCH Feed

• Exchange sends a stream of order messages

• End-hosts are only interested in orders for specific ticker symbols or prices

• But they receive all the orders, most of which are discarded
Filtering ITCH Feeds

shares > 100: [1]
(\text{shares} > 100 \land \text{stock} = \text{MSFT}): [2]
(\text{shares} < 60 \land \text{stock} = \text{AAPL}): [3]

\begin{tabular}{|c|c|}
\hline
\text{shares} & \text{next state} \\
\hline
\text{state} = 1 \land \text{shares} < 60 & 2 \\
\text{state} = 1 \land \text{shares} > 100 & 3 \\
\text{state} = 1 \land \text{shares} = \ast & 4 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|c|}
\hline
\text{stock} & \text{match} & \text{next state} \\
\hline
\text{state} = 2 \land \text{stock} = \text{AAPL} & 5 \\
\text{state} = 2 \land \text{stock} = \ast & 4 \\
\text{state} = 3 \land \text{stock} = \text{MSFT} & 7 \\
\text{state} = 3 \land \text{stock} = \ast & 6 \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
\_\text{fwd}\_ & \text{match} & \text{out ports} \\
\hline
\text{state} = 4 & [] \\
\text{state} = 5 & [3] \\
\text{state} = 6 & [1] \\
\text{state} = 7 & [1, 2] \\
\hline
\end{tabular}
Experimental Setup

Server

Feed Sender  25 Gb/s NIC

Receiver  25 Gb/s NIC

Tofino Switch

Filters for “GOOGL” Add Order messages
Experimental Setup

Tofino Switch

Two modes of operation:

- **Forwarding**: all traffic is forwarded normally
- **Filtering**: only Add Order messages for stock “GOOGL” are forwarded
How We Measure Latency

- Timestamp message when sending
- Timestamp message when received (after filtering)

Feed Sender → 25 Gb/s NIC

Receiver → 25 Gb/s NIC

End-To-End Latency
Nasdaq Sample 08302017, 1 msg/pkt

CDF (%) vs. Latency (us) with unfiltered and filtered lines.
1% GOOGL, 1-12 msgs/pkt
Apache Kafka

- Kafka is a high-throughput pub/sub system
- We implemented Kafka-like routing in the switch
- Drop-in replacement for existing kafka systems
Experimental Setup

Server 1
- Publisher
- Subscribers

Server 2
- Kafka Server
Experimental Setup

Server 1
- Publisher
- Subscribers

Server 2
- Kafka Server

Baseline
p4kafka
Kafka Experiment Parameters

- Number of Kafka broker servers: 1
- Number of topics: 2
- Replication: none
- Total messages: 100M
- Payload size: 512 bytes
- Batch size: 8192
Single Producer Throughput

- Baseline: 99MB/s
- p4kafka: 175MB/s
Producer-Consumer Latency

- **Baseline**: 2,080us
- **p4kafka**: 10us

**Diagram:**
- Y-axis: Microseconds
- X-axis: Latency
- Bars for baseline and p4kafka
1 Tofino > 2000 Kafka Servers

- Single Kafka server: 3.2Gb/s (400 MB/s)
- Tofino throughput: 6.5Tb/s
P4 Pub/Sub
Future Directions

• Can the language express other types of routing?
• Can we provide in-network reliability guarantees?
• What if the actions were more expressive?
• Could we build a stream processing system?
  • Yes!
Extra Slides
Related Work on BDDs

- The data structure is based on the Shannon expansion
- BDDs first introduced by Lee [1] in 1959
- Reduction and ordering proposed by Bryant et. al [2]
- Further extended by pruning “paths that correspond to infeasible constraints” [3]
- NetKAT [4] translates rules to BDDs, which are then complied to OpenFlow rules


Number of Queries vs. Forwarding Rules

![Graph showing the relationship between the number of generated table entries and the number of queries. The graph demonstrates a linear increase in table entries as the number of queries increases.]
Filtered Fields vs. Forwarding Rules

![Diagram](image-url)
Sample Spec and Rules

@pragma filter_field_exact(stock)
@pragma filter_field_range(shares)
@pragma filter_field_range(price)

header_type itch_add_order_t {
  fields {
    stock_locate: 16;
    tracking_number: 16;
    timestamp_ns: 48;
    order_ref_number: 64;
    buy_sell_indicator: 8;
    shares: 32;
    stock: 64;
    price: 32;
  }
}

add_order.price > 120
  and add_order.stock = “IBM”: 1;

add_order.shares < 1000
  and add_order.price > 100
  and add_order.stock = “AAPL”: 2;

add_order.price > 1000
  and add_order.shares > 1000: 3;
Hardware Specs

Server:

• 2x 8-core Intel Xeon CPU E5-2620 v4 @ 2.10GHz
• 256GB DDR4-2133 memory
• NIC: 25Gb/s Mellanox ConnectX-4 Lx
• or NIC: 2x 25Gb/s Intel XXV710
Nasdaq Sample 08302017: GOOGL Inter-Arrival Times
## Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Messages per packet</th>
<th>% GOOGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>1-12 (Zipf dist.)</td>
<td>1%</td>
</tr>
<tr>
<td>Nasdaq sample 08302017</td>
<td>Exactly 1</td>
<td>0.1%</td>
</tr>
<tr>
<td>Synthetic (worst-case)</td>
<td>Exactly 12</td>
<td>100%</td>
</tr>
</tbody>
</table>