Programmable Multi-Dimensional Table Filters for Line Rate Network Functions

Vishal Shrivastav
Purdue University
Evolution of Programmable Data Plane Hardware*

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

2013
RMT [SIGCOMM’13]
Match-Action Tables

2016
Domino [SIGCOMM’16]
Stateful Atoms

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

2013

RMT
[SIGCOMM’13]
Match-Action Tables

2016

Domino
[SIGCOMM’16]
Stateful Atoms

PIFO
[SIGCOMM’16]
Push-In-First-Out Queue

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

2013
- RMT [SIGCOMM’13]
  Match-Action Tables

2016
- Domino [SIGCOMM’16]
  Stateful Atoms
- PIFO [SIGCOMM’16]
  Push-In-First-Out Queue

2019
- FlowBlaze [NSDI’19]
  Finite State Machine

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

- **2013**: RMT [SIGCOMM'13] Match-Action Tables
- **2016**: Domino [SIGCOMM'16] Stateful Atoms, PIFO [SIGCOMM'16] Push-In-First-Out Queue

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

2013
- RMT [SIGCOMM'13]
  - Match-Action Tables

2016
- Domino [SIGCOMM'16]
  - Stateful Atoms
- PIFO [SIGCOMM'16]
  - Push-In-First-Out Queue

2019
- FlowBlaze [NSDI'19]
  - Finite State Machine
- PIEO [SIGCOMM'19]
  - Push-In-Extract-Out Queue

2022
- Taurus [ASPLOS'22]
  - Map-Reduce

* Not an exhaustive list
Evolution of Programmable Data Plane Hardware*

2013
- RMT [SIGCOMM’13]
  - Match-Action Tables

2016
- Domino [SIGCOMM’16]
  - Stateful Atoms
- PIFO [SIGCOMM’16]
  - Push-In-First-Out Queue

2019
- FlowBlaze [NSDI’19]
  - Finite State Machine
- PIEO [SIGCOMM’19]
  - Push-In-Extract-Out Queue

2022
- Taurus [ASPLOS’22]
  - Map-Reduce
- Thanos [SIGCOMM’22]
  - Multi-Dimensional Table Filtering

* Not an exhaustive list
Data Plane Filtering is Prevalent

Performance-aware Routing
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set

Multi-dimensional Filtering
Data Plane Filtering is Prevalent

<table>
<thead>
<tr>
<th>Paths</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Congestion</td>
</tr>
</tbody>
</table>

**Performance-aware Routing**
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set.

**Congestion-aware Load Balancing**
Filter path with minimum congestion.

Multi-dimensional Filtering
Data Plane Filtering is Prevalent

Performance-aware Routing
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set.

Congestion-aware Load Balancing
Filter path with minimum congestion or Filter d random egress ports. Choose the least queued port from those d ports.

Multi-dimensional Filtering
Data Plane Filtering is Prevalent

Performance-aware Routing
Filter paths with delay $< d$ and utilization $< u$. Choose a random path from the filtered set.

Congestion-aware Load Balancing
Filter path with minimum congestion or
Filter $d$ random egress ports. Choose the least queued port from those $d$ ports.

Resource-aware L4 Load Balancing
Filter servers with $\text{avail mem} > m$ and $\text{avail bw} > b$. From the filtered set, choose server with least cpu utilization.

Multi-dimensional Filtering

<table>
<thead>
<tr>
<th>Servers</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mem</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Plane Filtering is Prevalent

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Performance-aware Routing
Filter paths with delay $< d$ and utilization $< u$.
Choose a random path from the filtered set.

Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter $d$ random egress ports. Choose the least queued port from those $d$ ports.

Resource-aware L4 Load Balancing
Filter servers with avail mem $> m$ and avail bw $> b$.
From the filtered set, choose server with least cpu utilization.

Data Plane Diagnosis
Filter switch ports with packet rate $> t$.

Multi-dimensional Filtering
Data Plane Filtering is Prevalent

<table>
<thead>
<tr>
<th>Paths</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A (0/1)</td>
</tr>
</tbody>
</table>

**Policy Compliance**
From all available paths, filter the paths not carrying tenant A’s or B’s traffic. Choose a path at random from the filtered paths to route a new flow from tenant C.

**Performance-aware Routing**
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set.

**Congestion-aware Load Balancing**
Filter path with minimum congestion or Filter d random egress ports. Choose the least queued port from those d ports.

**Resource-aware L4 Load Balancing**
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization.

**Data Plane Diagnosis**
Filter switch ports with packet rate > t

**Multi-dimensional Filtering**
Data Plane Filtering is Prevalent

Performance-aware Routing
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set.

Congestion-aware Load Balancing
Filter path with minimum congestion or Filter d random egress ports. Choose the least queued port port from those d ports.

Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization.

Data Plane Diagnosis
Filter switch ports with packet rate > t.

Policy Compliance
From all available paths, filter the paths not carrying tenant A's or B's traffic. Choose a path at random from the filtered paths to route a new flow from tenant C.

Chained Multi-dimensional Filtering
# Data Plane Filtering is Prevalent

## Performance-aware Routing
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set.

## Congestion-aware Load Balancing
Filter path with minimum congestion or
Filter d random egress ports. Choose the least queued port from those d ports.

## Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization.

## Data Plane Diagnosis
Filter switch ports with packet rate > t.

## Chained Multi-dimensional Filtering at Line Rate
State-of-the-Art

Current Programmable Switch Processing Pipeline

Match-Action

Does not support line rate multi-dimensional filtering
Thanos
Thanos

Thanos Switch Processing Pipeline

Match-Action

Multi-dimensional Table

Filter Processing Pipeline

Programmable Filter Module
Filter Abstractions and Primitives
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

\[
\text{UFPU: table} \rightarrow \text{table}
\]

Filter a load balancing server at \textit{random}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|}
\hline
\textbf{ID} & \textbf{mem} & \textbf{bw} & \textbf{cpu} & \textbf{# conn} \\
\hline
\textbf{Servers} & & & & \\
\hline
\hline
\end{tabular}
\end{table}
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

\{ random( )
\quad predicate(attr relop X) \}

Filter load balancing servers with avail mem $>$ X

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

Servers

UFPU

mem $>$ X
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

\[
\left\{
\begin{array}{l}
\text{random( )} \\
\text{predicate(attr relop X)} \\
\text{weighted-round-robin(attr)}
\end{array}
\right\
\]

Filter load balancing servers in a **round robin** manner weighted by **avail bw**

### Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

Servers

UFPU

w-rr(bw)
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

\{
  random()
  predicate(attr $\_relop$ X)
  weighted-round-robin(attr)
  min/max(attr)
\}

Filter load balancing server with min cpu utilization

<table>
<thead>
<tr>
<th>Attributes (attr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID mem bw cpu # conn</td>
</tr>
</tbody>
</table>

Servers

UFPU

min(cpu)
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table → table

\{ random()
    predicate(attr relop X)
    weighted-round-robin(attr)
    min/max(attr) \}

Filter N least cpu utilized load balancing servers
or
Filter N random load balancing servers

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

A chain of N UFPU's

\[ I \rightarrow UFPU \rightarrow I - O_1 \rightarrow UFPU \rightarrow \ldots \rightarrow I_{N-1} - O_{N-1} \rightarrow UFPU \rightarrow O_N = O \]
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

$\{$
  random()
  predicate(attr relop X)
  weighted-round-robin(attr)
  min/max(attr)$\}$

Filter $N$ least cpu utilized load balancing servers

or

Filter $N$ random load balancing servers

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

A chain of $N$ UFPUUs

$I \rightarrow O_1 \rightarrow O_2 \rightarrow \cdots \rightarrow O_{N-1} \rightarrow O_N = O$
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

\{
  \begin{align*}
  \text{random( )} \\
  \text{predicate(}\text{attr relop X}) \\
  \text{weighted-round-robin(}\text{attr}) \\
  \text{min/max(}\text{attr})
  \end{align*}
\}

Filter $N$ least cpu utilized load balancing servers
or
Filter $N$ random load balancing servers

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

Servers

\[ I \xrightarrow{\text{K-UFPU}} O \]
Abstractions and Primitives

 Unary Filter Processing Unit (UFPU)

UFPU: table $\mapsto$ table

\[
\{ \text{random( )} \\
\text{predicate(attr relop X)} \\
\text{weighted-round-robin(attr)} \\
\text{min/max(attr)} \}
\]

Filter **N least cpu utilized** load balancing servers

or

Filter **N random** load balancing servers

**Attributes (attr)**

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

**K-UFPU** comprises **N** UFPU units and adds a new configurable parameter **K**

**K** specifies the length of chain (**max N**) (by setting **K=1**, K-UFPU reduces to UFPU)
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

\[
\begin{align*}
\text{random}( ) \\
\text{predicate}(\text{attr relop } X) \\
\text{weighted-round-robin}(\text{attr}) \\
\text{min/max}(\text{attr})
\end{align*}
\]

Filter N least cpu utilized load balancing servers

or

Filter N random load balancing servers

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

Servers

K-UFPU comprises N UFPUs and adds a new configurable parameter K.

K specifies the length of chain (max N) (by setting K=1, K-UFPU reduces to UFPU)

We use K-UFPU (instead of UFPU) as the basic computing unit
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

- random()
- predicate(attr $\textit{relop}$ $X$)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table $\rightarrow$ table

- union()
- intersection()
- difference()

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

Servers
### Abstractions and Primitives

#### Unary Filter Processing Unit (UFPU)

- `UFPU`: table → table
- **random()**
- **predicate(attr relop X)**
- **weighted-round-robin(attr)**
- **min/max(attr)**

#### Binary Filter Processing Unit (BFPU)

- `BFPU`: table, table → table
- **union()**
- **intersection()**
- **difference()**

#### Filter all load balancing servers with **avail mem > X and avail bw > Y**

**Attributes (attr)**

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
</table>

**Servers**

- **K-UFPU**
  - K=1, mem > X
  - K=1, bw > Y

- **BFPU**
  - intersect
### Abstractions and Primitives

#### Unary Filter Processing Unit (UFPU)
- UFPU: table $\rightarrow$ table
- random($\cdot$)
- predicate(attr $relop$ $X$)
- weighted-round-robin(attr)
- min/max(attr)

#### Binary Filter Processing Unit (BFPU)
- BFPU: table, table $\rightarrow$ table
- union($\cdot$)
- intersection($\cdot$)
- difference($\cdot$)

Filter all load balancing servers with **avail mem > X and avail bw > Y and # conn < Z**

<table>
<thead>
<tr>
<th>Servers</th>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attributes (attr)**

- **ID**
- **mem**
- **bw**
- **cpu**
- **# conn**
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\rightarrow$ table

- random()
- predicate(attr $\text{rel}op$ X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table $\rightarrow$ table

- union()
- intersection()
- difference()

Filter all load balancing servers with avail mem > X and avail bw > Y and # conn < Z

Filter $m$ load balancing servers at random

<table>
<thead>
<tr>
<th>Attributes (attr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>ID</td>
</tr>
<tr>
<td>ID</td>
</tr>
</tbody>
</table>

K-UFPU

K=1, bw > Y

K-UFPU

K=1, mem > X

BFPU

K-UFPU

K=1, # conn < Z

K=1, # conn < Z

K=1, mem > X

K=1, # conn < Z

K=m, random()
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\mapsto$ table

- random( )
- predicate(attr $\mathit{relop}$ $X$)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table $\mapsto$ table

- union( )
- intersection( )
- difference( )

Filter all load balancing servers with \text{avail mem} > X \text{ and } \text{avail bw} > Y \text{ and } \# \text{ conn} < Z

Filter \text{m} load balancing servers at \text{random}:

Filter load balancing server with \text{min cpu util}

Attributes (attr)

<table>
<thead>
<tr>
<th>ID</th>
<th>mem</th>
<th>bw</th>
<th>cpu</th>
<th># conn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Filter all load balancing servers with \text{avail mem} > X \text{ and } \text{avail bw} > Y \text{ and } \# \text{ conn} < Z

Filter load balancing server with \text{min cpu util}
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)
UFPU: table → table
- random()
- predicate(attr relop X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)
BFPU: table, table → table
- union()
- intersection()
- difference()

A 5-stage serial chain filter pipeline
(outputs from stage i are inputs to stage i+1)
From Abstractions to Hardware Design
Two Hardware Components
Hardware Component # 1

1. Multi-dimensional Table
2. Programmable Filter Pipeline
How to design an efficient data structure for a multi-dimensional relational table?

Should allow line rate read, write, update!
Multi-Dimensional Table Data Structure

Limitations of Classic Data Structures

No universal data structure

- Range trees / B-Trees for range filtering
- Heap for min/max filtering
- Disjoint Set Data Structure for set operations
- Either compromise on performance of certain operations or pay the cost of maintaining multiple data structures over the same data

Hierarchical Structure

- Fundamental $O(\log(N))$ latency
- Hard to pipeline\(^1\)

Multi-Dimensional Table Data Structure

Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)
Multi-Dimensional Table Data Structure

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Example Table
SMBM Property # 1

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Store each dimension as flat list of flip-flops
Allows parallel access and processing
SMBM Property # 2

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Stored as
Sorted

ID
1
2
3
4

X
4
15
15
22

Y
6
8
19
19

Each list is kept sorted
Allows fast max/min filter operations
SMBM Property # 3

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

Each dimension is stored as an *independent* list

Allows parallel filter operations on multiple dimensions
SMBM Property # 4

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Bidirectional mapping between ID and attributes

Stored as
Sorted Multidimensional Bidirectional Map (SMBM)

Our Solution:

Bidirectional mapping between ID and attributes
Forward map keeps track of which attributes belong to which ID

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>
SMBM Property # 4

Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Bidirectional mapping between ID and attributes

Forward map keeps track of which attributes belong to which ID
Reverse map allows fast mapping of filtered attributes to their respective IDs
SMBM Performance

Our Solution:
Sorted Multidimensional Bidirectional Map (SMBM)

Example Table

<table>
<thead>
<tr>
<th>ID</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>22</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>8</td>
</tr>
</tbody>
</table>

Stored as

Can read the entire data structure in parallel in 1 clock cycle

Add and Delete can be issued every clock cycle with a latency of 2 cycles
How to design a **fully reconfigurable** and **fast** filter pipeline?

- Can express any arbitrary chain of filter operation with chain length $\leq k$ on the $n$ input lines
- Runs at line rate
Filter Pipeline Layout
Reconfigurable Pipeline

In each stage:
1. Ability to apply any filter operation to an input (or pair of input) line
2. Ability to connect the output of a filter operation to any output line

Sufficient condition for full reconfigurability
Single Pipeline Stage Design

Naive design with both K-UFPUs and BFPUs

Input lines

n = 4

n X 2n Crossbar

High overhead

Output lines

n/2 X n Crossbar

n = 4
Single Pipeline Stage Design

Our design with both K-UFPUs and BFPUs

n X n Crossbar

Input lines

n = 4

Output lines

n = 4

Cell

K-UFPUs and BFPUs

n = 4
Example Configuration

Apply a UFPU op on Input 1 and connect to Output 3

n = 4
Input lines

n X n
Crossbar

Cell
K-UFPU → BFPU
K-UFPU → BFPU
K-UFPU → BFPU

n = 4
Output lines
Example Configuration

Apply a UFPU op on Input 1 and connect to Output 3

- n = 4
- n X n Crossbar
- n = 4 Output lines
- n = 4 Input lines

Cell

K-UFPU

K-UFPU

BFPU

BFPU

K-UFPU

no-op

K-UFPU

BFPU
Example Configuration

Apply a BFPU op on Input 1 and 3 connect to Output 2

K-UFPU
BFPU
Cell

n = 4
Input lines

Cell

n = 4
Output lines

n X n
Crossbar
Example Configuration

Apply a BFPU op on Input 1 and 3 and connect to Output 2

Cell

Input lines

Output lines

n = 4

n X n
Crossbar
Filter Pipeline Performance

• Filter pipeline can process a new filter request every clock cycle

• Our implementation runs at clock speeds in excess of 1 GHz
  • 1 GHz is the typical clock speed of today’s switches

• However, scalability is limited to a few 1000s of table entries
  • …beyond that the clock speed falls below 1 GHz
  • Still sufficient for many applications where table entries include network paths, switch ports, servers in a cluster, etc.
Application Performance

Load balancing client requests

**Policy 1.** Select a server uniformly at random.

**Policy 2.** Select a server uniformly at random from the set of servers with CPU utilization $< X$ and available memory $> Y$ and available bandwidth $> Z$. If the filtered set is empty, select a server uniformly at random from the entire set.

![Graph showing response time for Policy 2 normalized w.r.t. Policy 1]

1.7x—1.3x better response time for 70% of client requests
Application Performance

In-network caching of relational graph filter queries

Response time with caching normalized w.r.t. no caching

4x—2.8x better response time for cached filter queries
Summary

- **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table.
Summary

- **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table.

- Use cases include performance-aware routing, load balancing, network diagnosis, security, firewall…
Summary

• **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table.

• Use cases include performance-aware routing, load balancing, network diagnosis, security, firewall…

• The design runs in excess of 1 GHz clock speed and scales to thousands of table entries.
Summary

• **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table.

• Use cases include performance-aware routing, load balancing, network diagnosis, security, firewall…

• The design runs in excess of 1 GHz clock speed and scales to thousands of table entries.

• Evaluations show up to 1.7x improvement in performance of key network functions, such as routing and load balancing.
Summary

• **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table

• Use cases include performance-aware routing, load balancing, network diagnosis, security, firewall...

• The design runs in excess of 1 GHz clock speed and scales to thousands of table entries

• Evaluations show up to 1.7x improvement in performance of key network functions, such as routing and load balancing

• **Overall, advances the current state of in-network computing**
  - enables **richer** algorithms for existing in-network applications
  - enables **new** in-network applications, e.g., caching of relational queries
Thank you!