Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN

Presenters: Simran Kaur, Clover Zheng, Tamjeed Azad, Cindy Zhang

Recap: Software Defined Network (SDN)

Separates **control plane** from **data plane** via open interface (e.g., Openflow)

Makes routing decisions

Forwards packets through router

Recap: Software Defined Network (SDN)

Separates **control plane** from **data plane** via open interface (e.g., Openflow)

Makes routing decisions Forwards packets through router

Openflow is based on the **match-action** approach

1. Match (subset of) packet headers to *fixed* headers in table
2. Matched entry specifies *fixed* action(s) to apply to packet

Match-action tables are implemented with switch chips for speed

Abstracting Matching Semantics

Single Match Table

- Single table that stores every combination of headers
- Wasteful
  - Header behaviors might be orthogonal
  - What if a match on the first header determines a disjoint set of values to match on second header?

Abstracting Matching Semantics

Single Match Table

- Single table that stores every combination of headers
- Wasteful
  - Header behaviors might be orthogonal
  - What if a match on the first header determines a disjoint set of values to match on second header?

Multiple Match Table (a refinement of SMT)

- Smaller match tables arranged into pipeline of stages
- Processing at current stage dependent on processing at previous stages
- Existing switch chips implement a small number of tables, but…
  - Table width, depth, and execution order are *fixed*
  - Often limited set of common actions (OpenFlow specifies a subset of these)

SDN with OpenFlow

Benefits: Control plane is programmable

Limitations: Data plane is not programmable.

Discussion question: What causes this limitation?
Why would we want programmability here?
Hint: think about what you might want to change in a network

SDN with OpenFlow

Benefits: Control plane is programmable

Limitations: Data plane is not programmable.

Discussion question: What causes this limitation?
Why would we want programmability here?
Hint: think about what you might want to change in a network

Possible answer:

- Source of limitation: current switch hardware + OpenFlow
  - Match-action processing is on fixed set of fields
  - Openflow limits packet processing actions
- What if you want the flexibility to…
  - Add new headers (and match on them)
  - Add new actions
  - Tune table sizes to optimize for network

Motivating Question

Can we build a programmable switch (1) at a reasonable cost (2) without sacrificing speed?

Motivating Question

Can we build a programmable switch (1) at a reasonable cost (2) without sacrificing speed?

Reconfigurable Match Table (RMT)

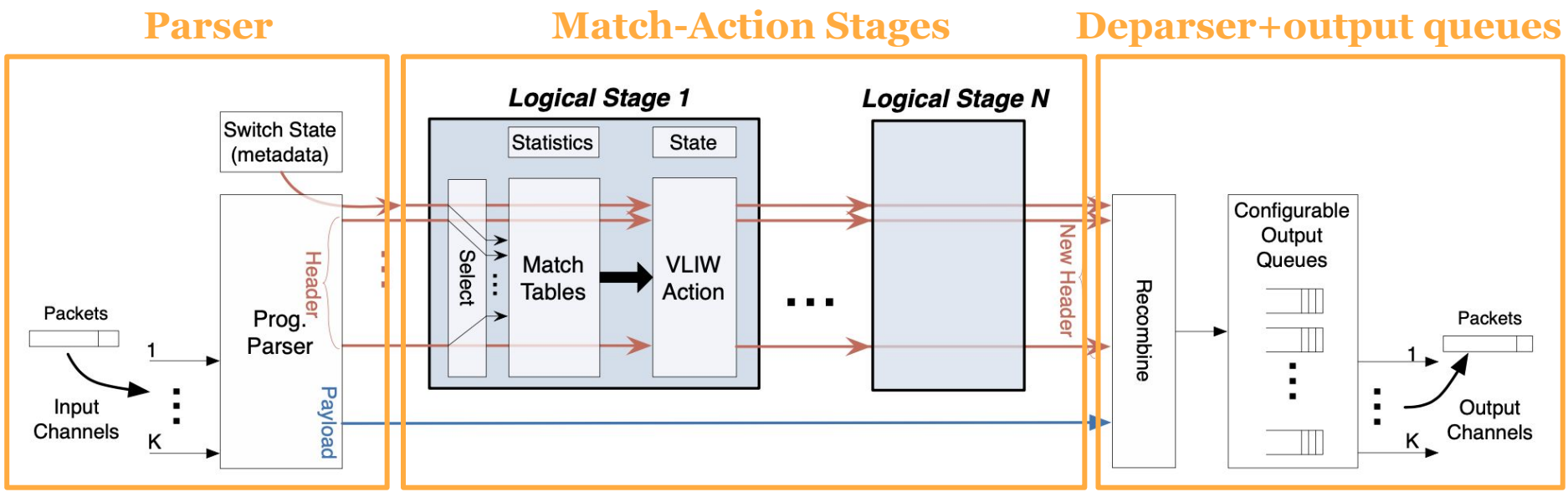
- Use pipeline of match-action tables (like in MMT)
- More flexibility
  - Add new headers (and match on them)
  - Define new actions
  - Configure number, topology, width, and depth of match tables

RMT Architecture

Techniques for **flexibility**:

- Programmable parser and deparser
- Match stage unit RAM configurability
- Ingress/egress resource sharing
- Allows multiple tables per stage

RMT Architecture



(a) RMT model as a sequence of logical Match-Action stages.

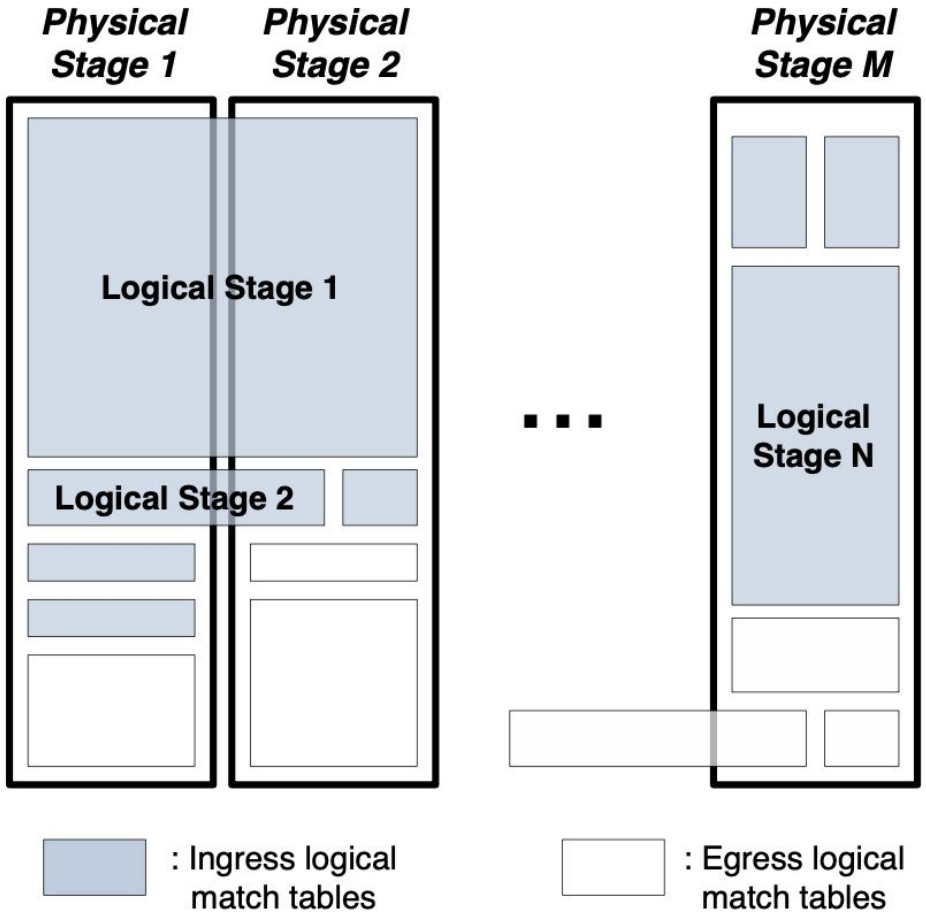
Programmable parser

- Can be configured with any
  - header fields and
  - header orders
- Outputs a packet header vector
  - a set of header fields such as IP dest, Ethernet dest, etc
- Allow field definitions to be modified or added
- Programmable deparser recombines the modified header fields

Match tables

The match-action pipeline consists of a series of tables of programmable dimensions

- Each table is one stage of the pipeline
- Resources are shared across all stages and can be allocated flexibly:
  - *more memory can be assigned with multiple contiguous physical stages*
  - IP forwarding: 256K 32-bit prefixes
  - Ethernet: 64K 48-bit addresses

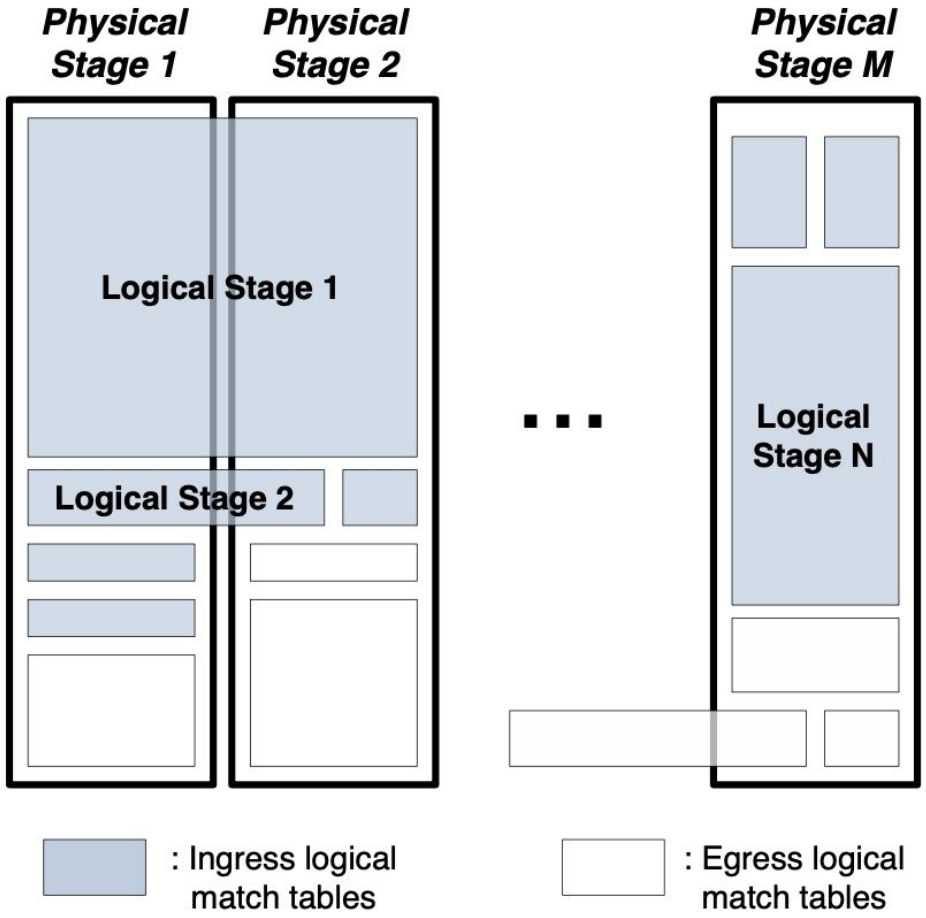


(b) Flexible match table configuration.

Match tables

The match-action pipeline consists of a series of tables of programmable dimensions

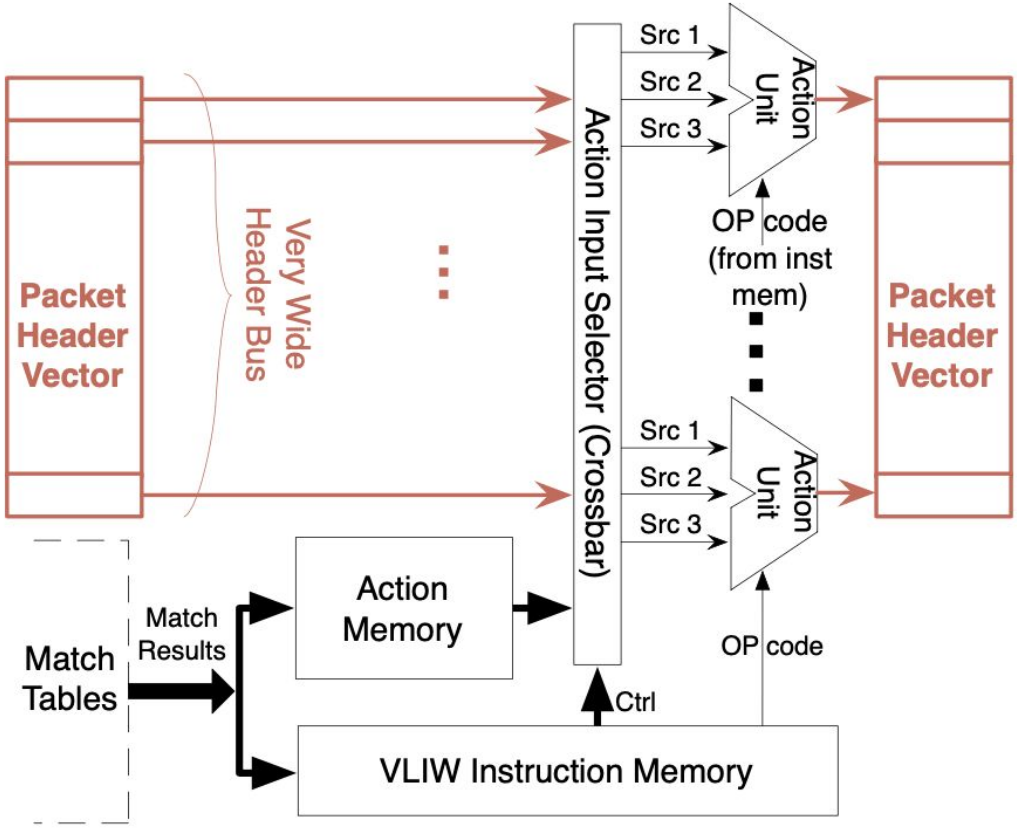
- Each table is one stage of the pipeline
- Resources are shared across all stages and can be allocated flexibly:
- Issues:
  - Large number of physical stages inflates **power requirements**
  - Conflates **processing and memory** allocation
    - A logical stage that requires more processing get allocated 2 physical stages, but it gets 2x memory even though it may not need it => potential resource waste!



(b) Flexible match table configuration.

Action

- *Input selector* picks the fields to be matched upon, and *VLIW*–Very long instruction word– modifies
  - Operate on all fields in the header vector concurrently
- An *action unit* for each field F in the header vector
  - Input: up to 3 arguments, including fields in the header vector, the action data results of the match
  - Output: rewrite F.
- The ingress and egress side separated by a switching fabric, but this distinction is mainly logical rather than physical.



(c) VLIW action architecture.

RMT Architecture Pros and Cons

Pros:

- Factoring states leads to better abstraction
  - Developers can write programs with a reasonably general “mental model” of what the switch looks like
  - Programming language like P4
- Minimize resource waste
- Layout optimality
- More actions than OpenFlow

Cons:

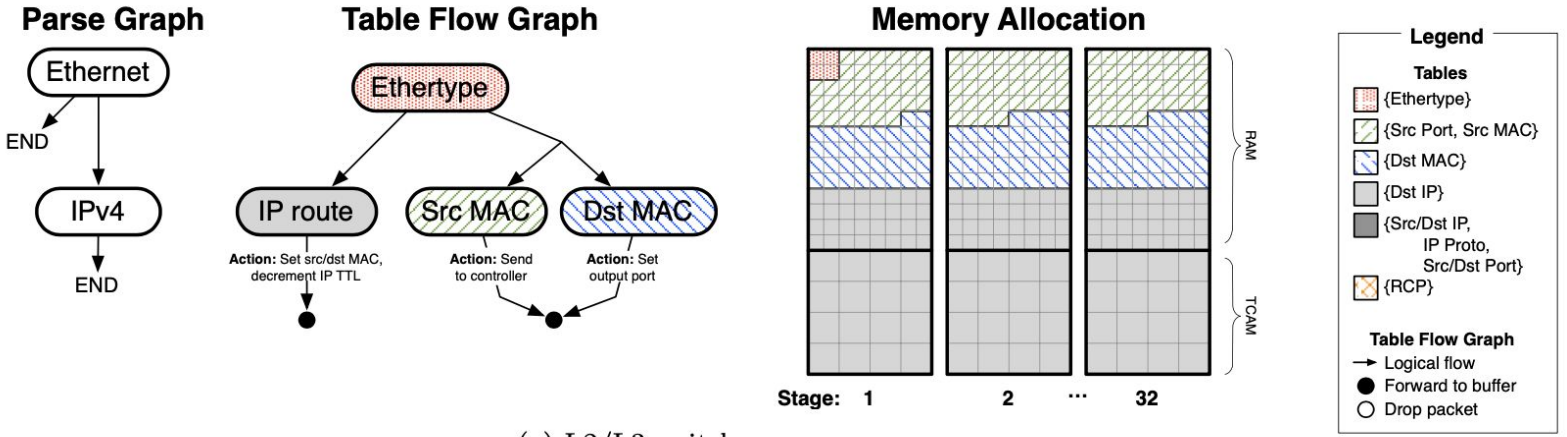
- Power consumption +15%
- Conflates processing and memory
- Match restrictions
  - N=32?
- Packet header limits
- Action restrictions
- Still not Turing-complete

Use cases for RMT

- Two main use cases that we should highlight:
  - L2/L3 switch.
  - RCP and ACL support.

Case 1: L2/L3 switch

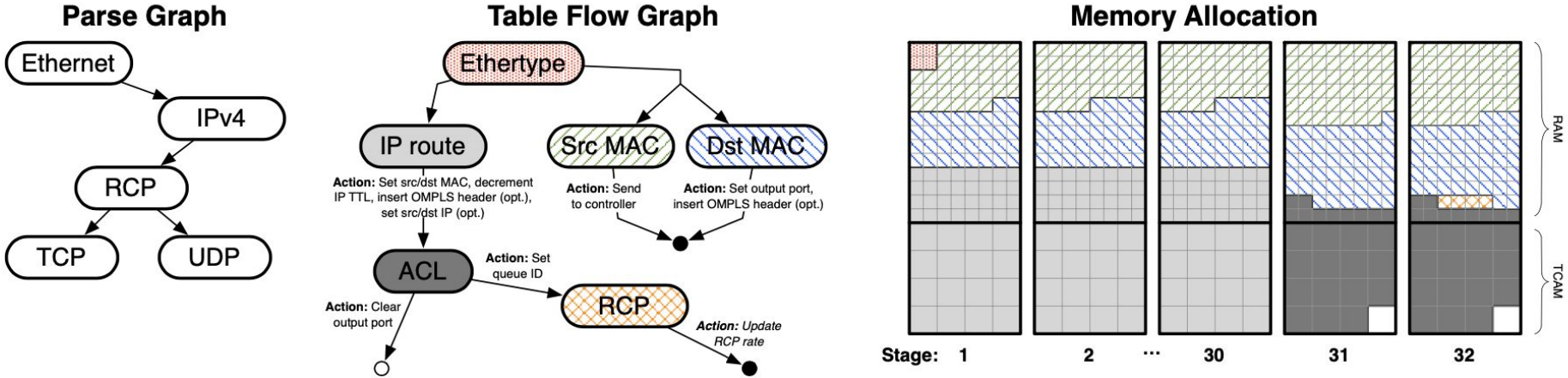
- Switching at either the Layer 2 level or the Layer 3 level.
  - The parser, match tables, and the action tables need configuration.
  - Memory: Ethertype table in stage 1, other tables spread out to maximize size.
  - Upon configuration, the control plane can start populating each table.



(a) L2/L3 switch.

Case 2: RCP and ACL support

- Rate Control Protocol (RCP) and Access Control List (ACL) for firewalls.
  - RCP indicates explicitly fair-share rate to flows; stamped into RCP header.
  - 20K ACL entries (each 120b wide) created along with RAM entries to hold associated actions, last 2 stages.



(b) RCP and ACL support.

Chip Design

- 1GHz operating frequency lets a single pipeline process all input port data.
- Parsers accept packets, individual fields are moved from various locations to fixed locations in a 4 Kb packet header vector. Results are multiplexed into a single stream.
- Queuing system associated with common data buffer.
- Egress and ingress pipelines share same match tables -> minimizes cost.

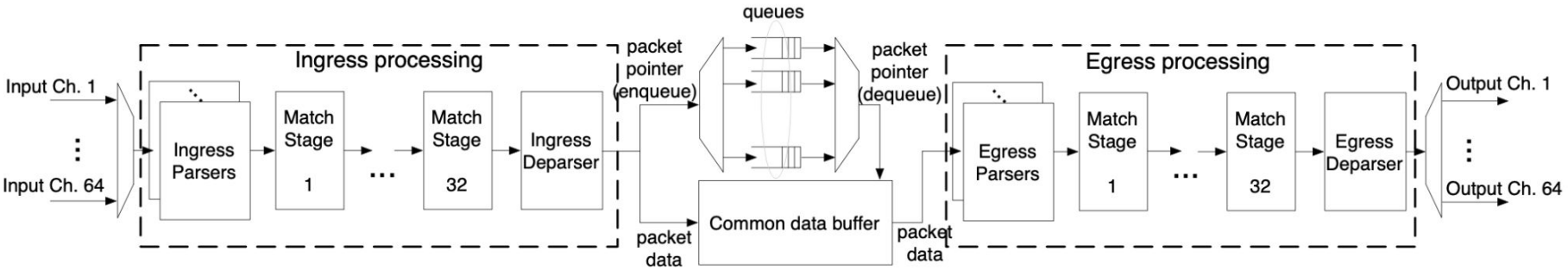


Figure 3: Switch chip architecture.

Chip Design: Configurable Parser

- Parser: turns incoming packet data into 4k sized packet header vector.
- The result of a TCAM match triggers an action.
  - Updates parser state.
  - Shifts incoming data.
  - Directs fields in input packet to final output packet fixed positions.

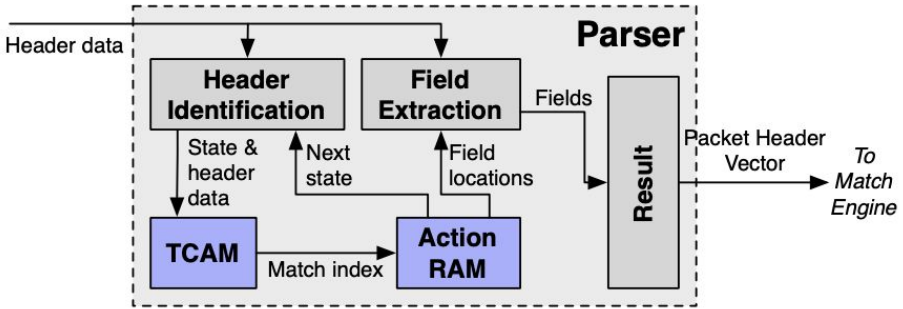


Figure 4: Programmable parser model.

Chip Design: Configurable Match Memories

- Each match stage has two wide match units: a TCAM for ternary matches and an SRAM-based hash table for exact matches.
- Ingress, egress match pipelines are the same block! Uses a crossbar.
  - Packet headers shared between input, output vectors: ownership configured to be either ingress or egress thread.
  - Function units per field allocated in the same way to ingress / egress.
  - Each memory block only allocated to either ingress / egress.
- Chip stores packet and byte statistics counters for each flow table entry.

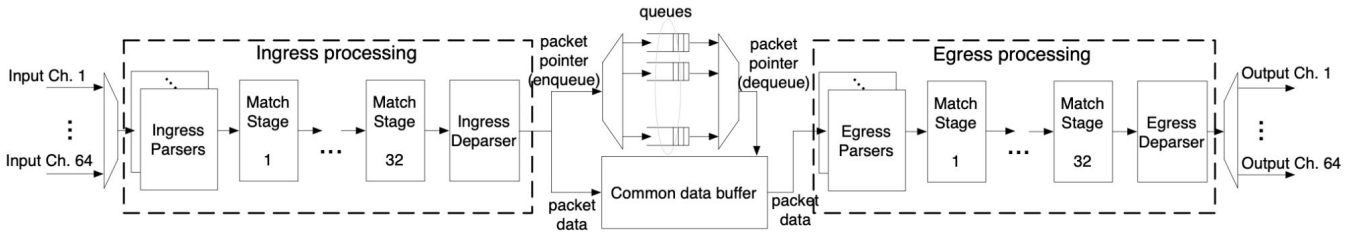


Figure 3: Switch chip architecture.

Chip Design: Configurable Action Engine

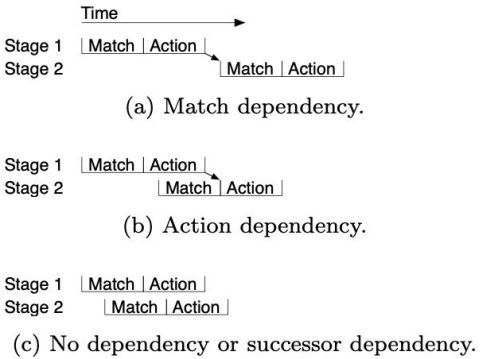
- Separate processing unit provided for each packet header field.
- OpenFlow specifies simple actions and complex operations; complex ones have to be flattened into single-cycle operations.
- Enumeration of a subset of instructions:

Category	Description
logical	and, or, xor, not, ...
shadd/sub	signed or unsigned shift
arith	inc, dec, min, max
deposit-byte	any length, source & dest offset
rot-mask-merge	IPv4 $\leftrightarrow$ IPv6 translation uses
bitmasked-set	$S_1 \& S_2 \mid \overline{S_1} \& S_3$; metadata uses
move	if V_{S_1} $S_1 \rightarrow D$
cond-move	if $\overline{V_{S_2}} \& V_{S_1}$ $S_1 \rightarrow D$
cond-mux	if V_{S_2} $S_2 \rightarrow D$ else if V_{S_1} $S_1 \rightarrow D$

Table 1: Partial action instruction set.
(S_i means source i; V_x means x is valid.)

Chip Design: Other Notable Features

- Latencies reduced by dependency analysis.
- Multicast and ECMP processing is split between ingress and egress.
- Meters measure and classify flow rates of matching table entries, which are then potentially used to modify or drop incoming packets.
- Version IDs flow through pipeline with each packet, allowing for version compatibility to be checked for in matches.
- Note: short wiring drives success!



(a) Match dependency.

(b) Action dependency.

(c) No dependency or successor dependency.

Figure 5: Match stage dependencies.

Cost Comparison

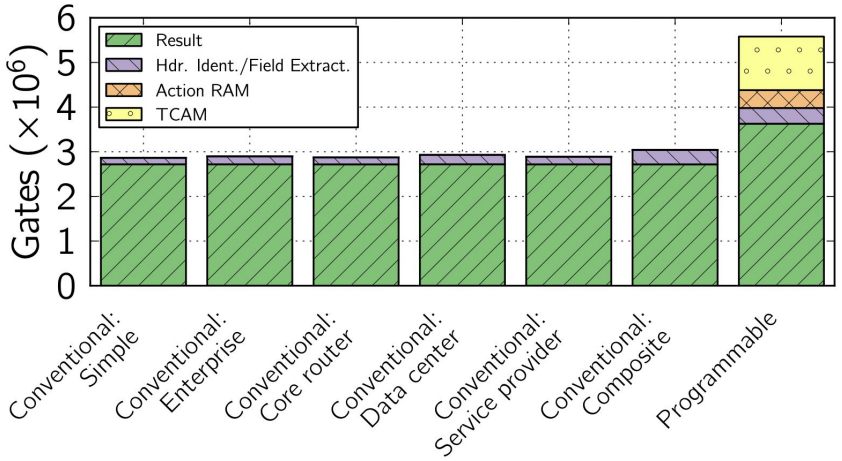
How does the proposed RMT design compare to a conventional switch chip?

Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Programmable parser:

- Uses 256 x 40 bit TCAM and 256 x 128 bit action RAM
- 5.6 million gates (in comparison to 2.9 - 3 million for conventional design)



Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Memory costs:

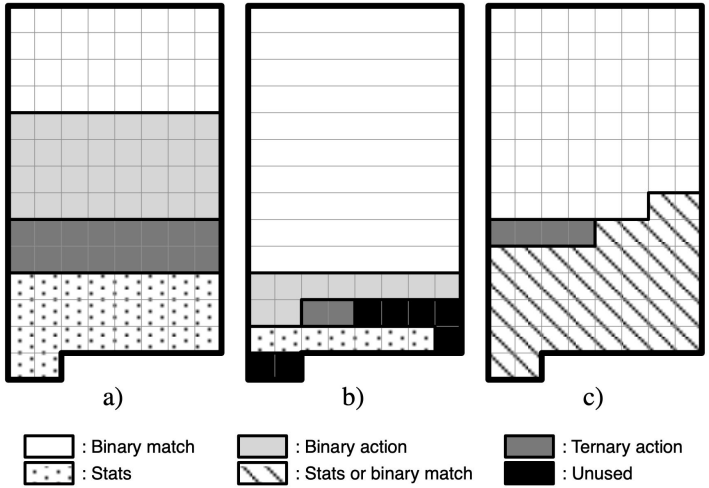
- Memory technology
    - Extra cost incurred by hash table, TCAMs

Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Memory costs:

- Memory technology
- Action specification
  - Flow entries have overhead bits for a pointer to action memory, action size, etc.
  - Amount of overhead varies in different configurations of memory blocks



Case	MatchWidth	Match	Action	Stats	Relative
a_1	80	32	48	25	1.000×
a_2	160	52	34	18	1.625×
a_3	320	72	22	12	2.250×
b	640	80	12	7	2.500×
c_1	80	62	4	40	1.900×
c_2	80	102	4	0	3.250×

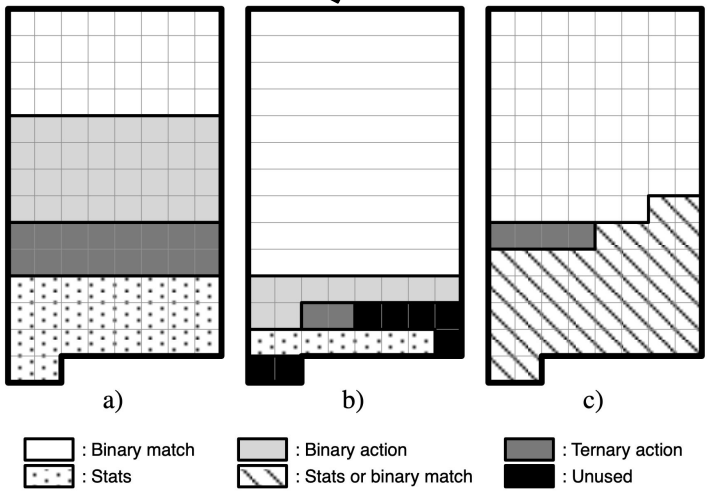
Cost Comparison

Flow table match width increases
Action, statistics capacity decreases
Flow entry density increases

How does the proposed RMT design compare to a conventional switch chip?

Memory costs:

- Memory technology
- Action specification
  - Flow entries have overhead bits for a pointer to action memory, action size, etc.
  - Amount of overhead varies in different configurations of memory blocks



Case	MatchWidth	Match	Action	Stats	Relative
a_1	80	32	48	25	1.000×
a_2	160	52	34	18	1.625×
a_3	320	72	22	12	2.250×
b	640	80	12	7	2.500×
c_1	80	62	4	40	1.900×
c_2	80	102	4	0	3.250×

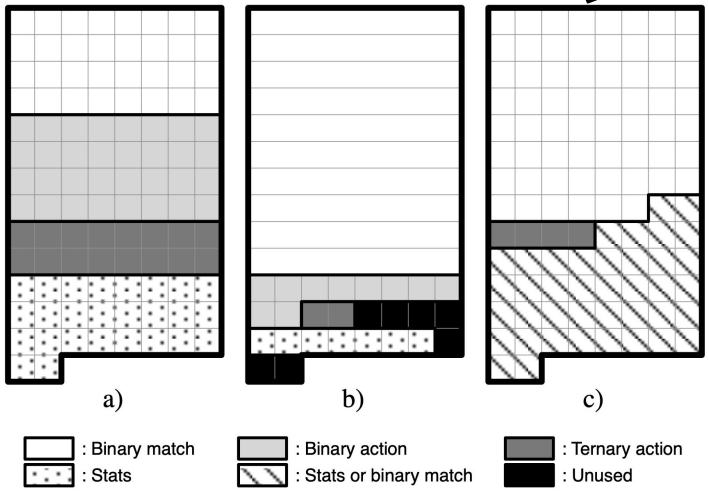
Cost Comparison

Less action memory
More memory blocks for match or statistics

How does the proposed RMT design compare to a conventional switch chip?

Memory costs:

- Memory technology
- Action specification
  - Flow entries have overhead bits for a pointer to action memory, action size, etc.
  - Amount of overhead varies in different configurations of memory blocks



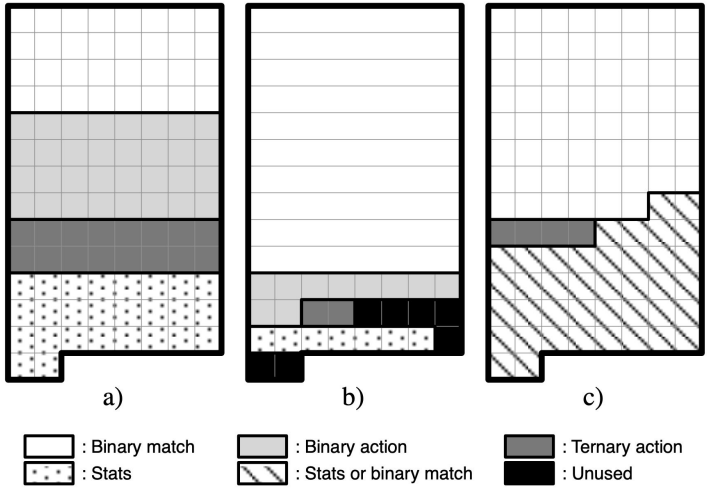
Case	MatchWidth	Match	Action	Stats	Relative
a_1	80	32	48	25	1.000×
a_2	160	52	34	18	1.625×
a_3	320	72	22	12	2.250×
b	640	80	12	7	2.500×
c_1	80	62	4	40	1.900×
c_2	80	102	4	0	3.250×

Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Memory costs:

- Memory technology
- Action specification
- Fragmentation costs
  - Reduce by allowing sets of flow entries to be packed together



Case	MatchWidth	Match	Action	Stats	Relative
a_1	80	32	48	25	1.000×
a_2	160	52	34	18	1.625×
a_3	320	72	22	12	2.250×
b	640	80	12	7	2.500×
c_1	80	62	4	40	1.900×
c_2	80	102	4	0	3.250×

Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Area and power costs:

- Large match table capacity contributes substantially to chip area estimates

Section	Area	Cost
IO, buffer, queue, CPU, etc	37.0%	0.0%
Match memory & logic	54.3%	8.0%
VLIW action engine	7.4%	5.5%
Parser + deparser	1.3%	0.7%
Total extra cost:		**14.2%**

Cost Comparison

How does the proposed RMT design compare to a conventional switch chip?

Area and power costs:

- Large match table capacity contributes substantially to chip area estimates
- Increased memory bitcount and increased functionality of match-action pipeline dissipates more power

Section	Area	Cost
IO, buffer, queue, CPU, etc	37.0%	0.0%
Match memory & logic	54.3%	8.0%
VLIW action engine	7.4%	5.5%
Parser + deparser	1.3%	0.7%
Total extra cost:		**14.2%**

Section	Power	Cost
I/O	26.0%	0.0%
Memory leakage	43.7%	4.0%
Logic leakage	7.3%	2.5%
RAM active	2.7%	0.4%
TCAM active	3.5%	0.0%
Logic active	16.8%	5.5%
Total extra cost:		**12.4%**

Summary

- Discussion question: *What are the long-term benefits of programmable switches?*

Summary

- Discussion question: *What are the long-term benefits of programmable switches?*
- This paper's main contributions:
  - Describes RMT switch architecture
  - Proves that this architecture is realizable on proposed chip design
  - **Greatly increased flexibility** is possible at a cost of less than 15% increase in area and power consumption of the chip!

Follow-Up Work

- RMT commercialized in Tofino switch by Barefoot Networks
- P4 is a standard language used for different programmable switching devices
- dRMT disaggregates the memory and compute resources of the switch
- Ongoing research still pushing the boundaries on leveraging the flexibility of RMT