USENIX THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

nsdi '25

When P4 Meets Run-to-completion Architecture

Hao Zheng[†], Xin Yan[△], Wenbo Li[†], Jiaqi Zheng[†], Xiaoliang Wang[†], Qingqing Zhao[△], Luyou He[△], Xiaofei Lai[△], Feng Gao[△], Fuguang Huang[△], Wanchun Dou[†], Guihai Chen[†], Chen Tian[†]

NJU

HUAWEI

2025/04/30

Language and Programmable Data Plane

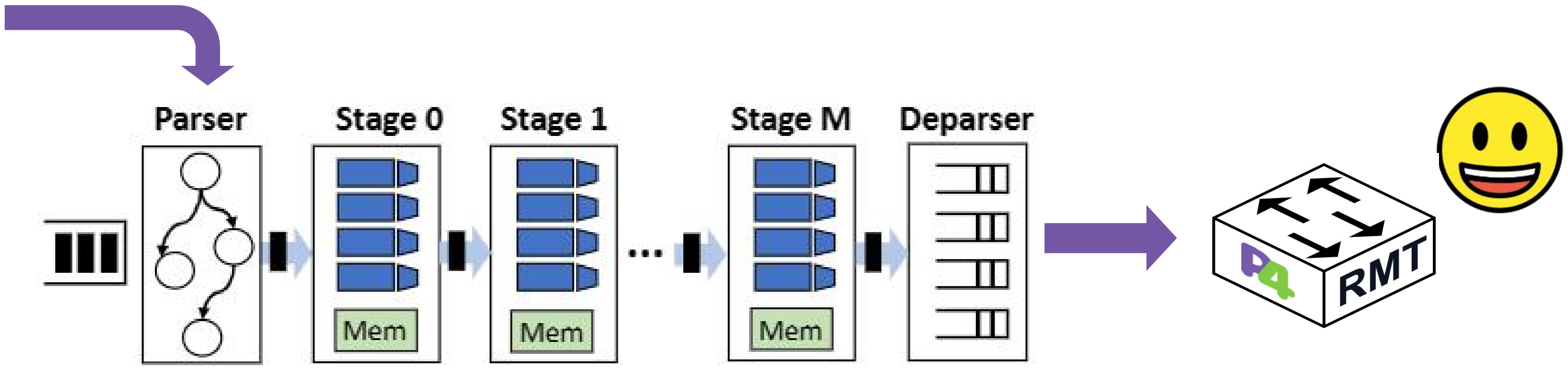
```
# include <core .p4 >
# include < arch_specific .p4 >

control ingress_block {
    Customized
    Packet Processing
    Logic
 }
 ...
Pipeline (parser, ingress, egress,
        deparser ) main
```

P4 Language

Developer

Without having to understand the underlying hardware details of the chip, network developers/researchers can:

- ➢ offload tasks to hardware
- ➢ verify new protocols or algorithms
- ➢ ...



P4 Programmable Data Plane (e.g., BMv2, Tofino)

Build a “new switch” in "compilation" time

P4 Language and Programmable Data Plane

In the past 10 years, P4 has received widespread attention in network community.

1

P4 makes Networks Better!

P4 accelerates the evolution of networks.

- **In-network computing:** NetCache(SOSP'17),SwitchML(NSDI'21)...
- **Congestion control:** HPCC(SIGCOMM'19), BFC(NSDI'22), ...
- **Load balancing:** HULA(SOSR'16)..
- **Network measurement**: In-band Telemetry, Sketches....
- ...

2

Researchers make P4 Better!

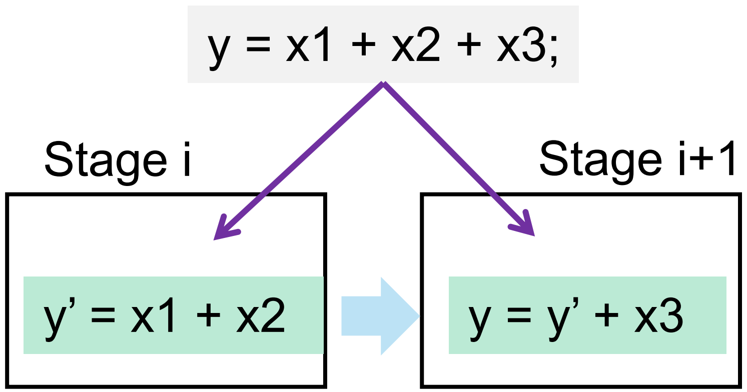
Many efforts have been made to strengthen or expand P4.

- **P4 Code Verification:** Vera(SIGCOMM'18), Aquila(SIGCOMM'21)...
- **Salable P4 Programming:** P4ALL(NSDI'22)...
- **Formal foundation of P4:** Petr4(PACMPL)...
- **Debugging tools for P4:** Tracking P4 Program (SOSR'20)...
- ...

Challenges of Current P4 Programmable Network

Challenge 1 : The programmability of P4 language is limited.

The number of MAU stages is limited: *e.g., 12 in Tofino and 20 in Tofino2*



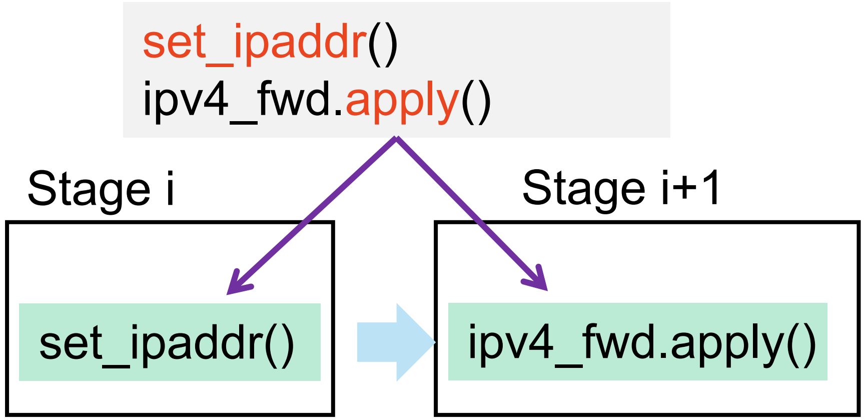
y = x1 + x2 + x3;

Stage i

y’ = x1 + x2

Stage i+1

y = y’ + x3



set_ipaddr()
ipv4_fwd.apply()

Stage i

set_ipaddr()

Stage i+1

ipv4_fwd.apply()

In the P4 pipeline, each MAU stage has limited cycles to process packets, and dependencies take up additional MAU stages.

The maximum logical length of P4 programs is limited.

Challenges of Current P4 Programmable Network

Challenge 1 : The programmability of P4 language is limited.

Limited match-action unit (MAU) stages: *e.g., 12 in Tofino and 20 in Tofino2*

The maximum logical length of P4 programs is limited.

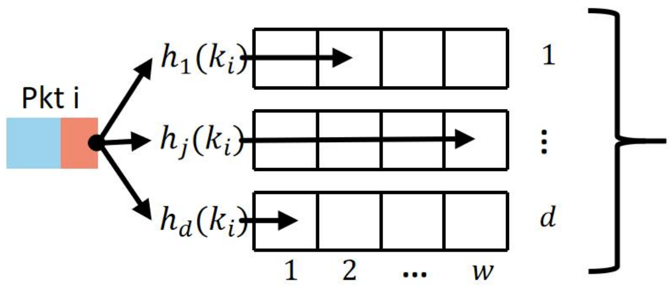
Challenges of Current P4 Programmable Network

Challenge 1 : The programmability of P4 language is limited.

Limited match-action unit (MAU) stages: *e.g., 12 in Tofino and 20 in Tofino2*

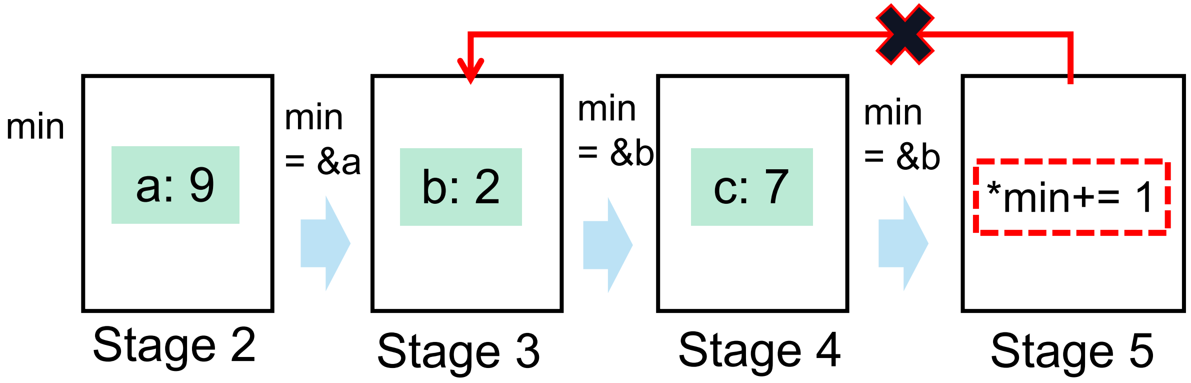
The maximum logical length of P4 programs is limited.

Restricted stateful memory operations: *isolated memory and limited size*



Conservative-update Count-Min sketch

```
int* minData = &a;
if (b < *minData)
        minData = &b;
if (c < *minData)
        minData = &c;
*minData += 1;
```



The algorithms that can be implemented by P4 are limited.

Challenges of Current P4 Programmable Network

Challenge 1 : The programmability of P4 language is limited.

Limited match-action unit (MAU) stages: *e.g., 12 in Tofino and 20 in Tofino2*

The maximum logical length of P4 programs is limited.

Restricted stateful memory operations: *isolated memory and limited size*

The algorithms that can be implemented by P4 are limited.

Challenges of Current P4 Programmable Network

Challenge 1 : The programmability of P4 language is limited.

Limited match-action unit (MAU) stages: *e.g., 12 in Tofino and 20 in Tofino2*

The maximum logical length of P4 programs is limited.

Restricted stateful memory operations: *isolated memory and limited size*

The algorithms that can be implemented by P4 are limited.

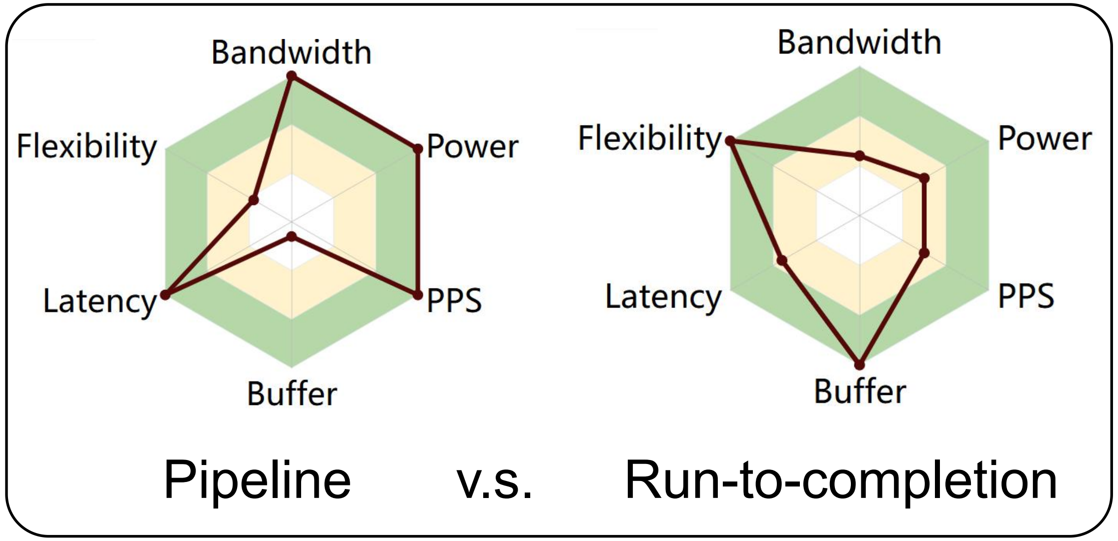
Challenge 2: The cessation of the next-generation Tofino chip.

⚠ The P4 language community needs a new member with **more flexible programmability** and **Tbps-level throughput!**

P4 Programmable Hardware Architecture

- **Traditional : <u>Pipelined Architecture</u>**

  The traditional P4 switch architecture is pipelined. Although the throughput is high, the flexibility and memory capacity are limited.



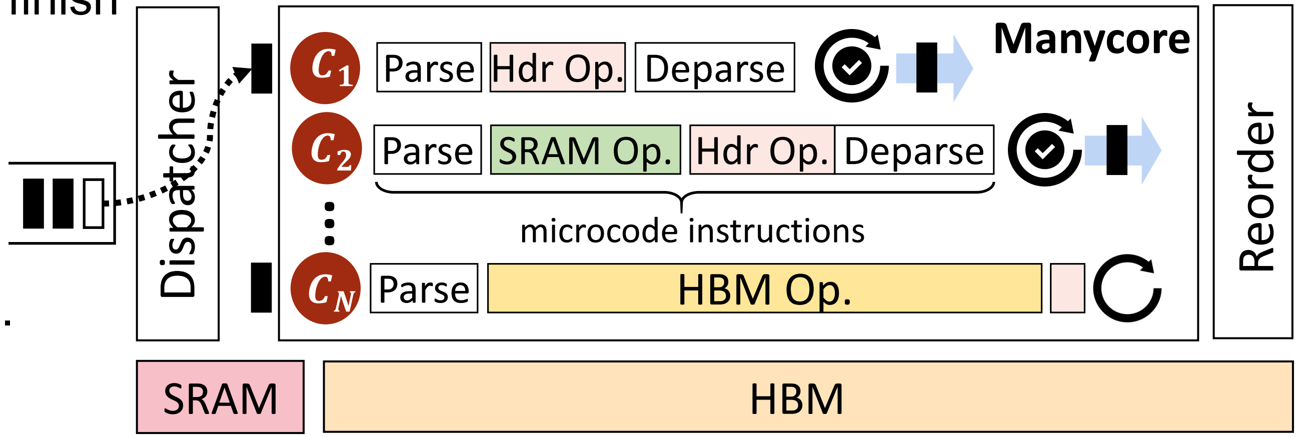
Pipeline v.s. Run-to-completion

- **<u>Run-to-completion Architecture</u>**

  Run-to-completion processing flavor
  - Each core handles a packet from start to finish
  - Unlimited logical length

  Manycore parallel processing
  - Thousands of threads, 1.2 Tbps

  Shared memory subsystem
  - <u>On-chip SRAM</u> for low-latency forwarding.
  - <u>Off-chip HBM</u> for large-size flow tables and deep buffers.



P4RTC: New Programmable Chip & P4 Extensions

P4RTC is a framework that supports P4 language on Huawei's run-to-completion chips.

Challenge 1

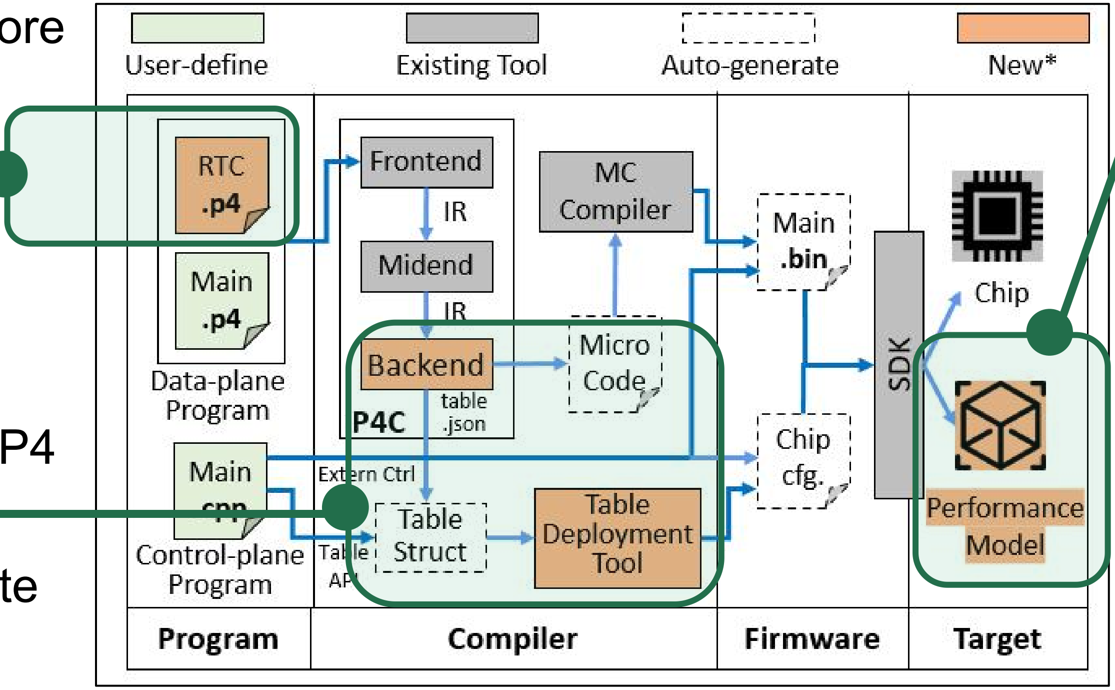
How to *extend P4* to fully leverage the flexibility of RTC without altering P416 core language?

Challenge 2

How to efficiently *map* P4 code to the RTC architecture and allocate resources.

Challenge 3

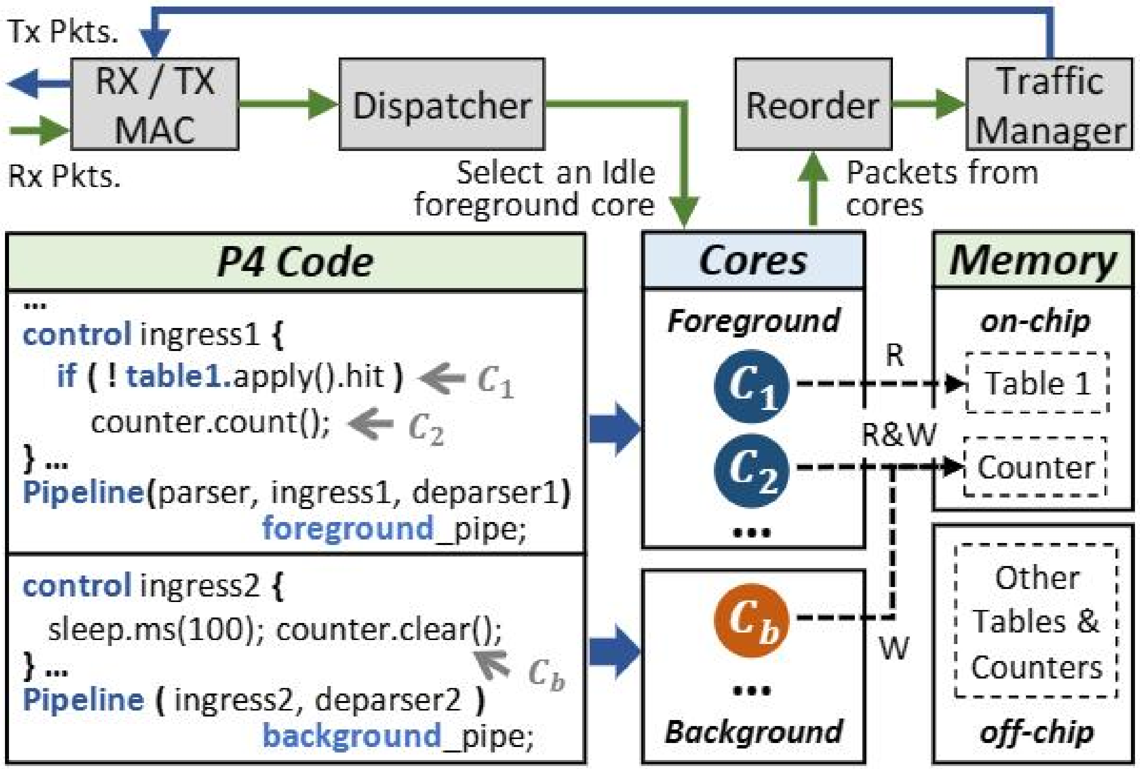
How to verify the performance of the chip before loading P4 codes in the production environment?



P4RTC design overview

P4RTC Programming

rtc.p4 = New Architecture Model + New P4 Extensions (externs)



```
...
control ingress1 {
  if ( ! table1.apply().hit )
     counter.count();
} ...
Pipeline(parser, ingress1, deparser1)
                       foreground_pipe;
control ingress2 {
  sleep.ms(100); counter.clear();
} ...
Pipeline ( ingress2, deparser2 )
                       background_pipe;
```

We introduce a new P4 architecture model:

1. **Similar to previous P4 programming**

   Parser, Ingress, Egress (Optional), Deparser, ...

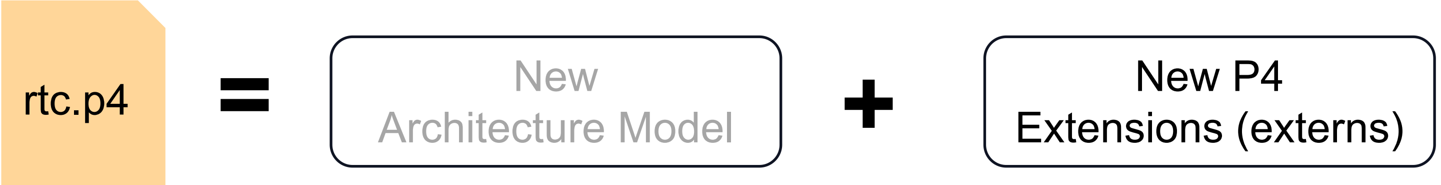
2. **Parallel Execution Model**

   Each core performs a logical pipeline for each packet.

3. **Shared Memory**

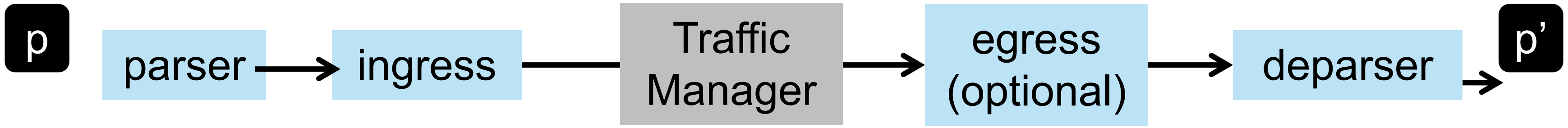
   Cores can simultaneously access a shared memory object (e.g., tables, counters)

P4RTC Programming



Pipeline Extensions: there are two pipeline roles in the new architecture model.

1. Foreground pipelines: running common P4 packet processing logic:

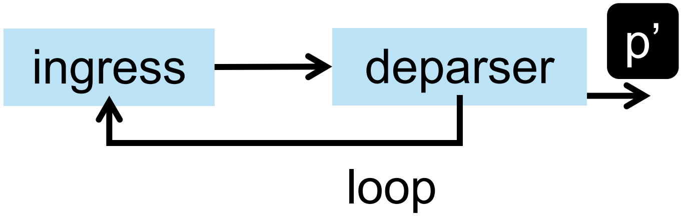


P4RTC Programming

rtc.p4 = New Architecture Model + New P4 Extensions (externs)

Pipeline extensions: there are two pipeline roles in the new architecture model.

1. Foreground pipelines: work as common P4 packet processing logic:
2. Background pipelines: don't accept packets, just run an infinite loop.



Background pipelines can generate customized packets by setValid() headers in the ingress control block.

Control flow extensions:

- Foreach
- continue(), break()
- sleep
- return()

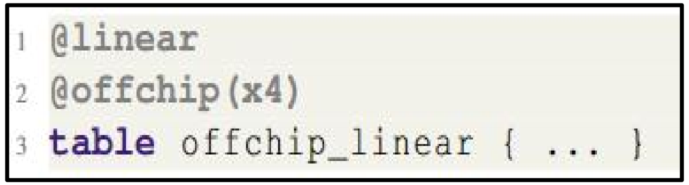
P4RTC Programming

rtc.p4 = New Architecture Model + New P4 Extensions (externs)

Table type extensions:

According to the deployment location:

- on-chip tables < 100 *ns latency, 10's MB*
- off-chip tables. 100ns ~*1 us latency, 1000's MB*



```
@linear
@offchip(x4)
table offchip_linear { ... }
```

P4RTC uses annotations to define different tables

According to the memory layout and indexing mechanism:

Key	Action ID	Action Data
10.0.0.1	0	2
10.0.0.2	1	5
...	...	...

content addressing tables (CAT)
(e.g., exact matching, LPM...)

Key	Action ID	Action Data
0	0	2
1	1	5
⋮	...	...

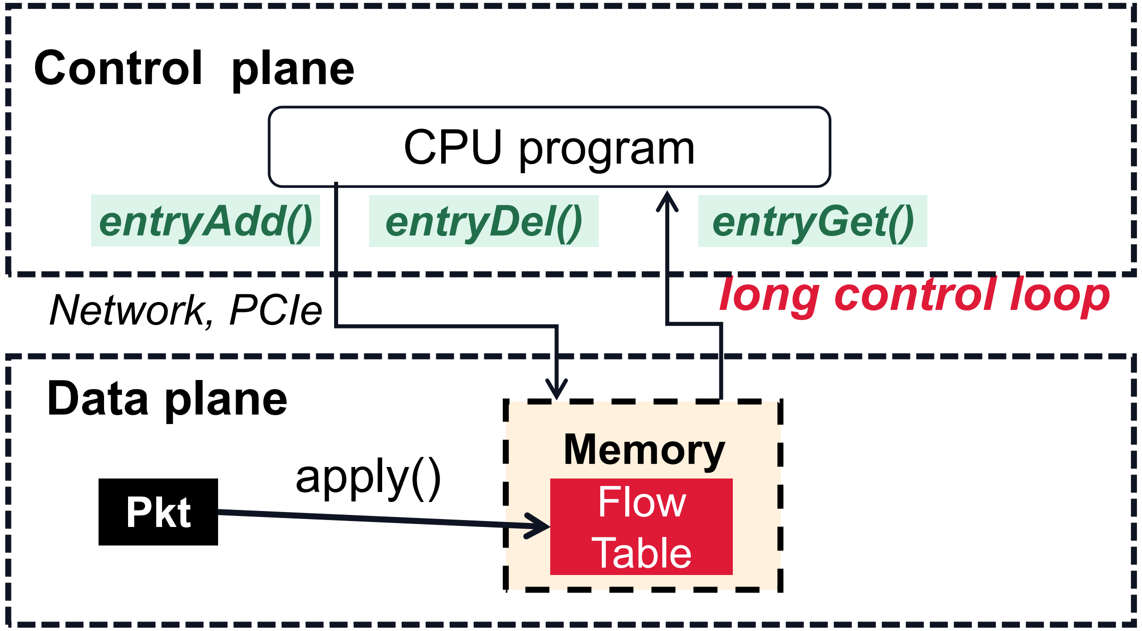
linear addressing tables (LAT)
(e.g., Counters, Registers, ...)

P4RTC Programming

rtc.p4 **=** New Architecture Model **+** New P4 Extensions (externs)

Traditional Table Operation :

P4 Table Operations are performed in the control plane...

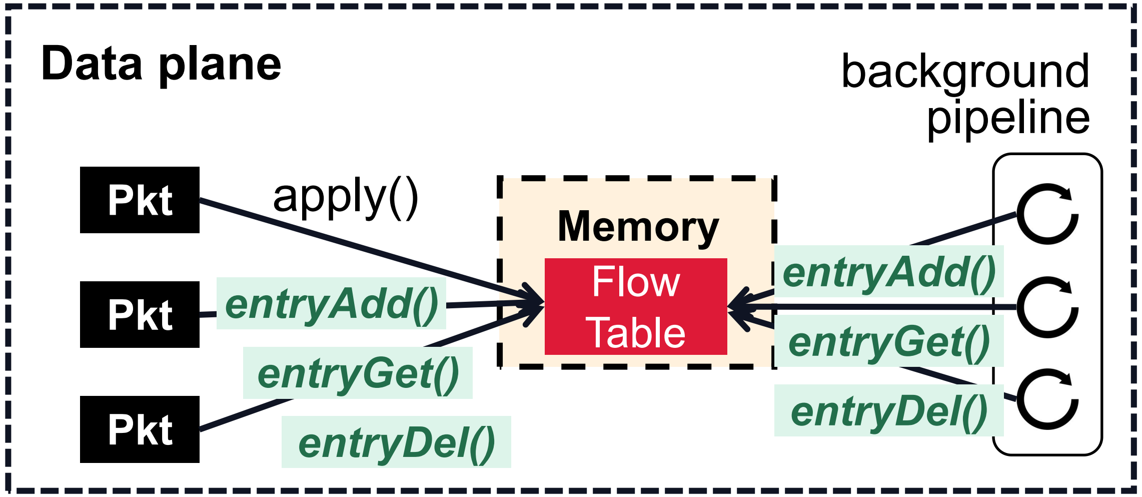


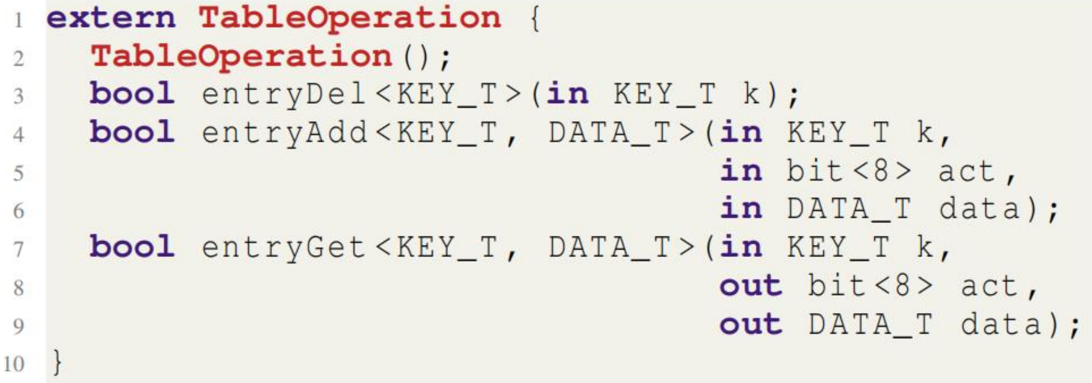
P4RTC Programming

rtc.p4 = New Architecture Model + New P4 Extensions (externs)

Table Operation Externsions:

Packets can *operate table entries directly in the data plane*, both for CAT and LAT tables.



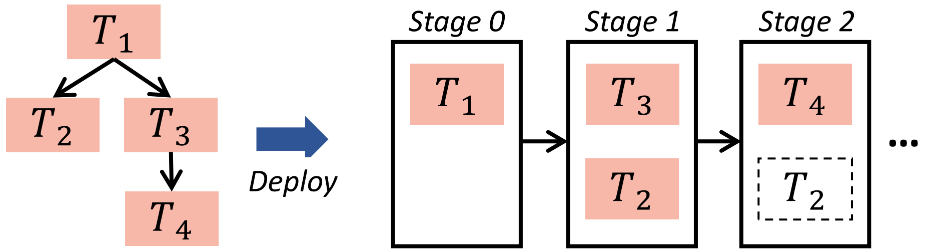


```
extern TableOperation {
  TableOperation();
  bool entryDel<KEY_T>(in KEY_T k);
  bool entryAdd<KEY_T, DATA_T>(in KEY_T k,
                               in bit<8> act,
                               in DATA_T data);
  bool entryGet<KEY_T, DATA_T>(in KEY_T k,
                               out bit<8> act,
                               out DATA_T data);
}
```

Bypass PCIe and software stacks, and reduce control loop latency to the range of 100 nanoseconds to a few microseconds.

P4RTC Compilation Table Deployments

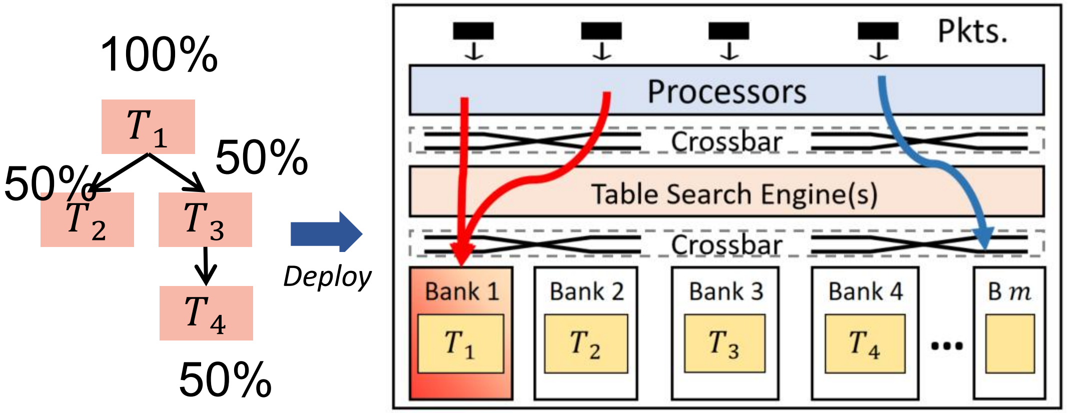
Table deployments in Pipelined Architecture:



The compiler needs to meet:

1. **Resource constraints:** place the tables on the data plane according to their type, size and the memory capacity of MAU stages.
2. **Dependency constraints:** satisfy the constraints of dependency between different tables during the deployment.

Table deployments in RTC Architecture:



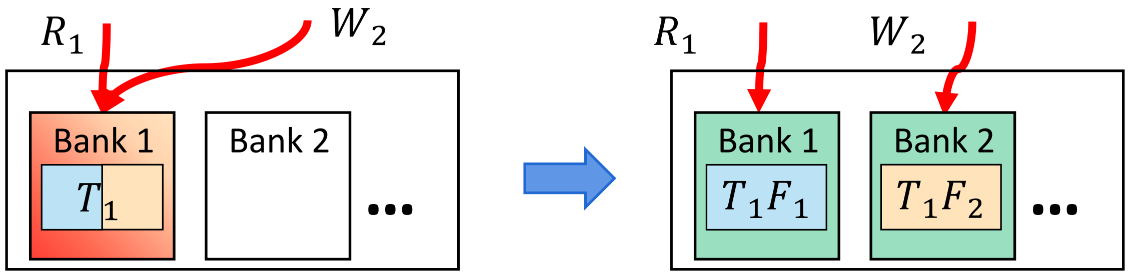
The compiler needs to meet:

1. **Resource constraints of memory banks:** place the tables on the data plane according to their type, size and the capacity of memory banks.
2. **Performance constraints:**
   When dealing with "hot" tables, off-chip memory often acts as a bottleneck.

P4RTC Compilation *Table Deployments*

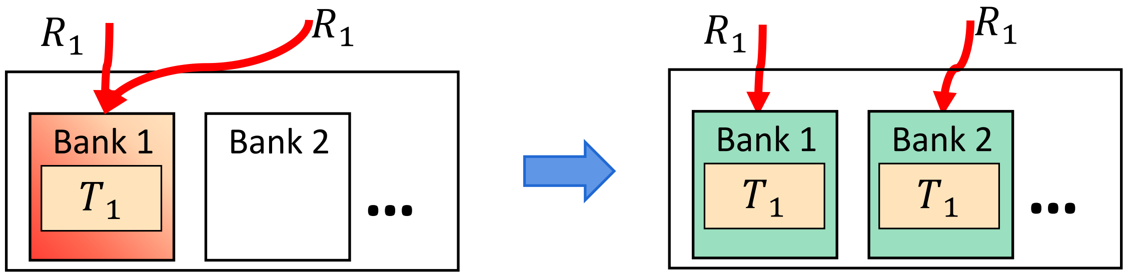
When deploying tables, <u>table fragmentation</u> and <u>table replication</u> are employed to facilitate the placement of tables and optimize performance.

Table fragmentation:



1. Accommodate tables that are too large for a bank.
2. Balance the load of the tables that are too heavy for a bank.

Table replication:



1. Balancing the load of tables with heavy *READ* loads helps to optimize the performance.
2. But imposes additional overhead on *WRITE* requests.

P4RTC Compilation *Table Deployments*

When deploying tables, table fragmentation and table replication are employed to facilitate the placement of tables and optimize performance.

$$
\begin{aligned}
\text{minimize} \quad & \max_{j\in\{1,\ldots,m\}} \sum_{i=1}^{n} x_{ij}\cdot l_i && (1)\\
\text{subject to} \quad & \sum_{j=1}^{m} x_{ij} = s_i, \quad \forall i \in \{1,\ldots,n\} && (2)\\
& \sum_{i=1}^{n} x_{ij} \le c_j, \quad \forall j \in \{1,\ldots,m\} && (3)\\
& \frac{x_{ij}}{s_i} \le y_{ij}, \quad \forall i \in \{1,\ldots,n\},\ j \in \{1,\ldots,m\} && (4)\\
& \sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} \le T && (5)\\
& x_{ij} \in \mathbb{N},\ \forall i \in \{1,\ldots,n\},\ j \in \{1,\ldots,m\} && (6)\\
& y_{ij} \in \{0,1\},\ \forall i \in \{1,\ldots,n\},\ j \in \{1,\ldots,m\} && (7)
\end{aligned}
$$

Minimize the maximum memory bank load

Automatic optimizations in the compilation:

- We build an Integer Linear Programming model to find a feasible and performance-optimized solution).

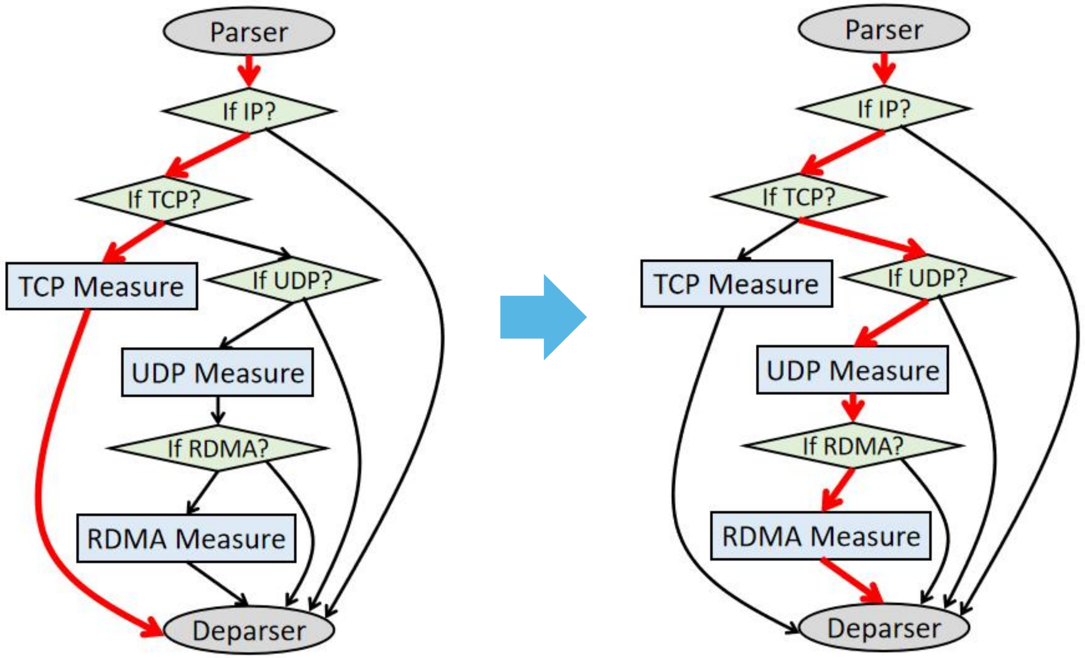
P4RTC Compilation

Table Deployments

+ Microcode Gen

Please see our paper for detail

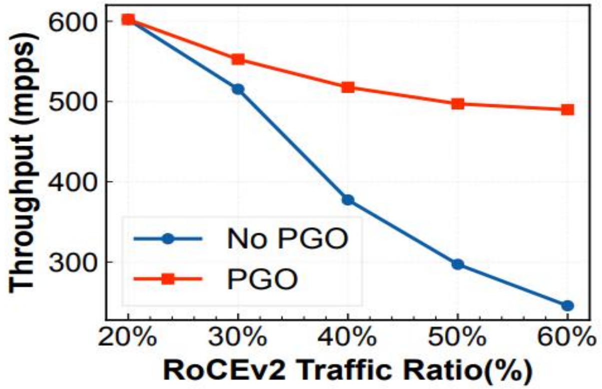
When deploying tables, table fragmentation and table replication are employed to facilitate the placement of tables and optimize performance.



Significant changes in traffic patterns may impact chip performance.

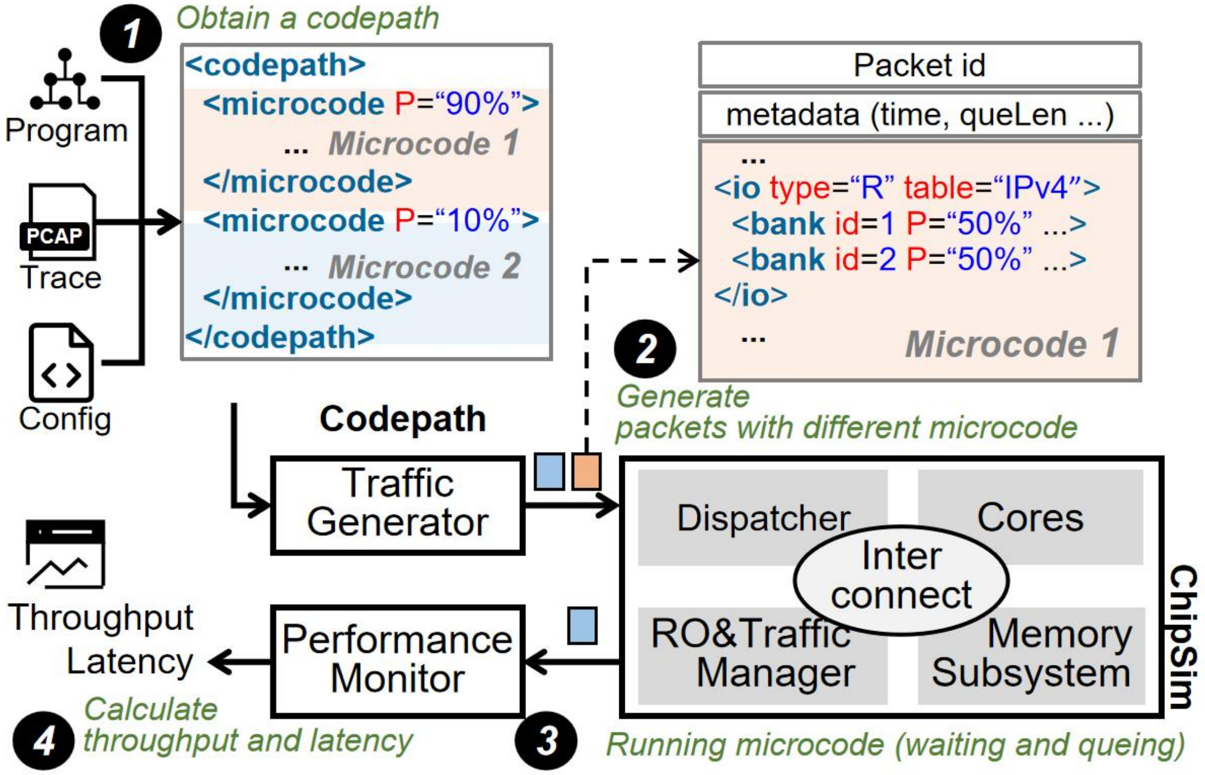
Automatic optimizations in the compilation:

- We build an Integer Linear Programming model to find a feasible and performance-optimized solution.
- We adopt Profile-guided optimization to adapt to significant workload changes.



Performance Model

The RTC architecture does not have deterministic performance like pipeline ASICs. Therefore, performance verification are required before deployment.



```
<codepath>
  <microcode P="90%">
      ... Microcode 1
  </microcode>
  <microcode P="10%">
      ... Microcode 2
  </microcode>
</codepath>
```

```
...
<io type="R" table="IPv4">
  <bank id=1 P="50%" ...>
  <bank id=2 P="50%" ...>
</io>
...                    Microcode 1
```

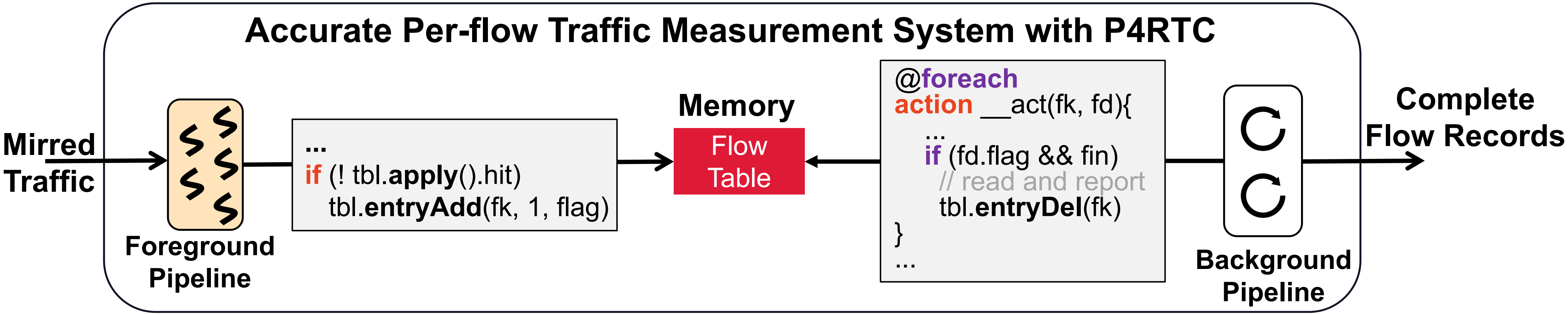
We build a performance model that uses the Electronic System Level (ESL) methodology.

The simulation process includes 4 parts:

1. Obtain a codepath based on P4 programs and a specific workload.
2. Generate traffic according to the codepath.
3. The chipsim (a cycle-approximate chip simulator) processes packets according to the microcode embedded in the packets.
4. The performance monitor measures and output the performance metrics (e.g., latency, throughput).

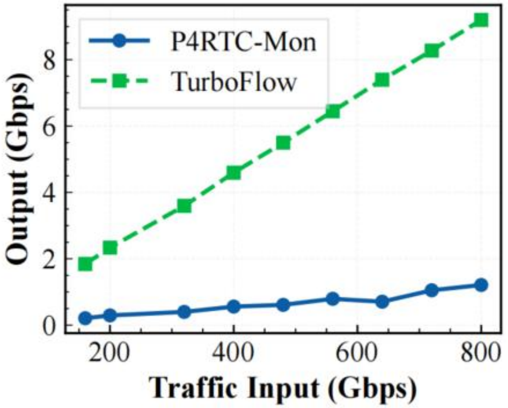
Case Study: Accurate Per-flow Monitoring

We prototype P4RTC on **Huawei NetEngine 8000 F1A-C (NE8000F1AC)** routers, with a 1.2 Tbps RTC chip with 8 GB high-bandwidth memory (HBM).



```
...
if (! tbl.apply().hit)
   tbl.entryAdd(fk, 1, flag)
```

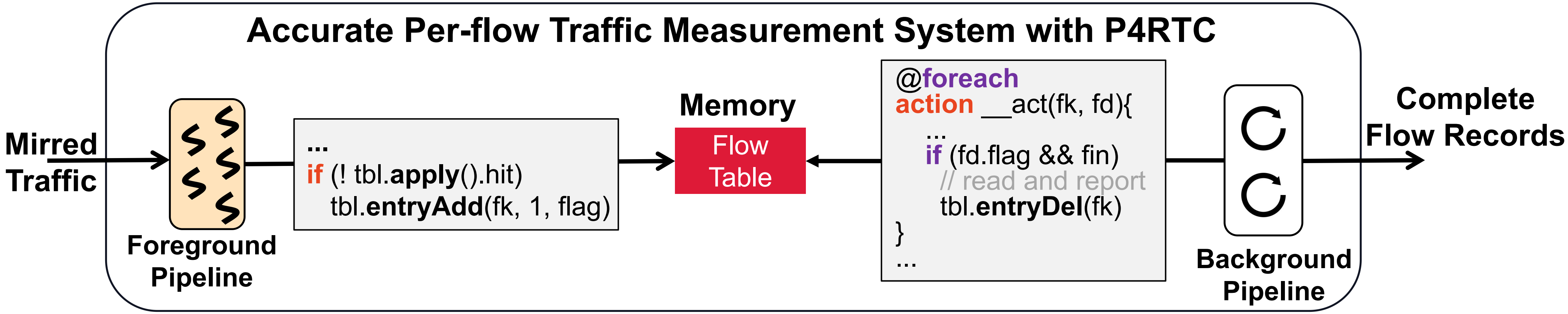
```
@foreach
action __act(fk, fd){
   ...
   if (fd.flag && fin)
      // read and report
      tbl.entryDel(fk)
}
...
```



1. It supports measuring 50M concurrent flows.

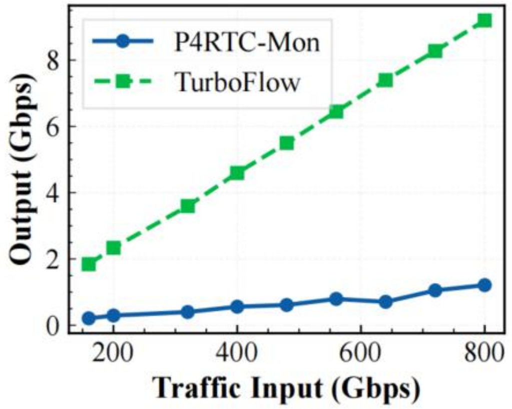
Case Study: Accurate Per-flow Monitoring

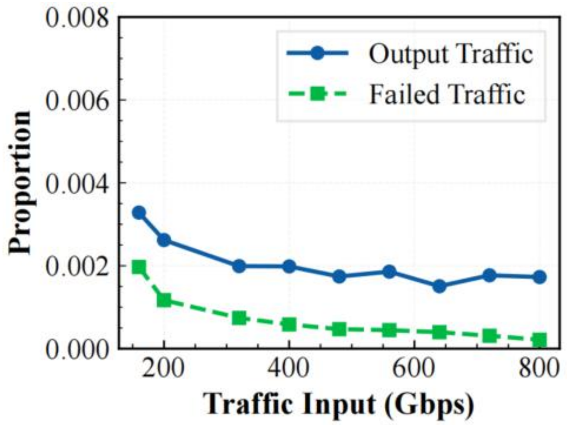
We prototype P4RTC on Huawei NetEngine 8000 F1A-C (NE8000F1AC) routers, with a 1.2 Tbps RTC chip with 8 GB high-bandwidth memory (HBM).



```
...
if (! tbl.apply().hit)
   tbl.entryAdd(fk, 1, flag)
```

```
@foreach
action __act(fk, fd){
   ...
   if (fd.flag && fin)
     // read and report
     tbl.entryDel(fk)
}
...
```





1. It supports measuring 50M concurrent flows.
2. It outputs highly aggregated measurement results in less than 0.5% of input:
   - Complete flow records (no hash collision triggered eviction)
   - Minimal failed packets due to lock acquisition during insertion of flow entries.

Conclusion

We proposed P4RTC, a comprehensive summary of our experiences in applying the P4 language to the RTC architecture (specifically, Huawei NetEngine 8000 F1A-C).

- P4RTC benefits P4 programmability. It incorporates RTC architectures into the P4 community, enabling functions that were previously unsupported in P4.
- P4RTC benefits RTC-based devices. It provides users with a simplified and general approach to developing functions.
- The performance model benefits P4 program profiling. It details the impact of different P4 codes on the chip under specific traffic inputs.

Thanks!