Feature Rich Flow Monitoring with P4

John Sonchack
University of Pennsylvania

Outline

- Introduction: Flow Records
- Design and Implementation: P4 Accelerated Flow Record Generation
- Benchmarks and Optimizations

Outline

- **Introduction: Flow Records**
- Design and Implementation: P4 Accelerated Flow Record Generation
- Benchmarks and Optimizations

Introduction: Flow Records

- A flow record summarizes groups of packets in the same TCP / UDP stream

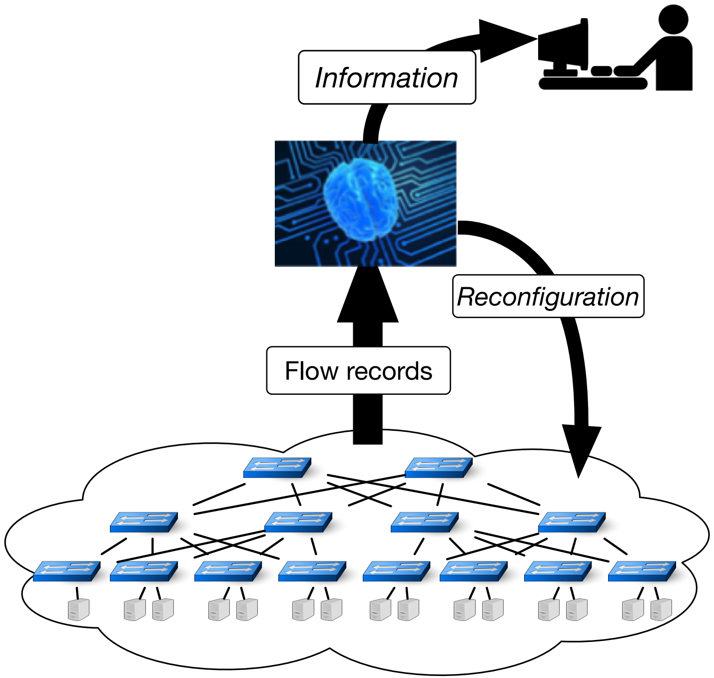
Flow **key**
(IP 5-tuple)

Flow **features**
(statistics and meta-data)

Key Fields	**Flow 1**	**Flow 2**
1. Source IP	10.1.1.1	10.1.1.6
2. Dest. IP	10.1.1.2	10.1.1.7
3. Source Port	34562	12520
4. Dest. Port	80	88
5. Protocol	TCP	UDP
Flow Summary Fields		
6. Packet Count (bytes)	5	7
7. Byte Count (bytes)	88647	3452
8. Timestamp (ms)	1473874	1473878
9. Duration (ms)	1025	535
Packet Inter-arrival		
10. Minimum (ms)	14	1
11. Maximum (ms)	3082	421
Packet Size Statistics		
12. Minimum (bytes)	64	64
13. Maximum (bytes)	1522	182
14. First Packet (bytes)	64	64
15. Second Packet (bytes)	1522	64

Introduction: Flow Record Applications

- Flow records are input to ***analysis applications***
- Examples:
  - **Security***:* Detect botnets, intrusions, DoS attacks, port scans
  - **Traffic management:** Traffic classification, QoS routing, flow scheduling, load balancing
  - **Debugging:** Performance monitoring, loop and black hole detection
  - More:
    - "A survey of network flow applications" *B. Li, et al.*
    - "An overview of ip flow-based intrusion detection" *A.Sperotto, et al.*
    - NetFlow/IPFIX applications

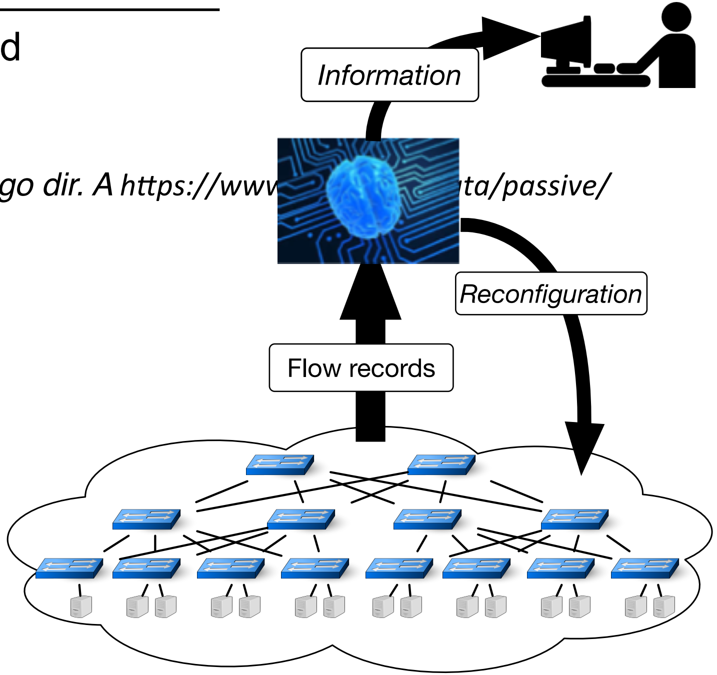


Introduction: Flow Record Benefits

- Flow records **reduce monitoring costs**

	Packet Headers	Flow Records
Volume	2GB per second	1.38MB per second
Rate	328k per second	8.9k per second

Table source: 1 hour 10 Gbit/s core router trace (CAIDA 02/2015 Chicago dir. A https://www ta/passive/ trace_stats/)



- Important for **high coverage monitoring**
  - visibility into many (or all) links
  - costs add up

Introduction: Flow Record Richness

- An important quality of flow records is their ***information richness:***

Accuracy: Account for every packet and flow on monitored links

Feature richness: provide the features that applications use

Key Fields	**Flow 1**	**Flow 2**
1. *Source IP*	10.1.1.1	10.1.1.6
2. *Dest. IP*	10.1.1.2	10.1.1.7
3. *Source Port*	34562	12520
4. *Dest. Port*	80	88
5. *Protocol*	TCP	UDP
Flow Summary Fields		
6. *Packet Count (bytes)*	5	7
7. *Byte Count (bytes)*	88647	3452
8. *Timestamp (ms)*	1473874	1473878
9. *Duration (ms)*	1025	535
Packet Inter-arrival		
10. *Minimum (ms)*	14	1
11. *Maximum (ms)*	3082	421
Packet Size Statistics		
12. *Minimum (bytes)*	64	64
13. *Maximum (bytes)*	1522	182
14. *First Packet (bytes)*	64	64
15. *Second Packet (bytes)*	1522	64

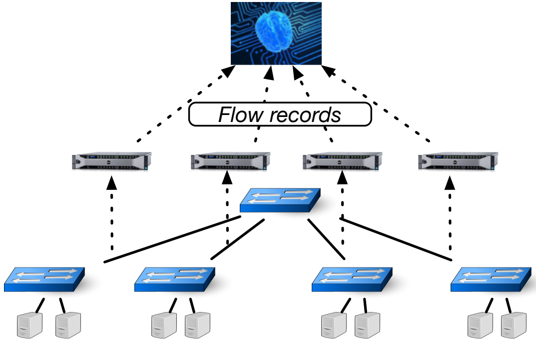
Introduction: Flow Record Generation

- The problem: current approaches trade **overhead** for **information richness**

Software Generation

+ Information rich

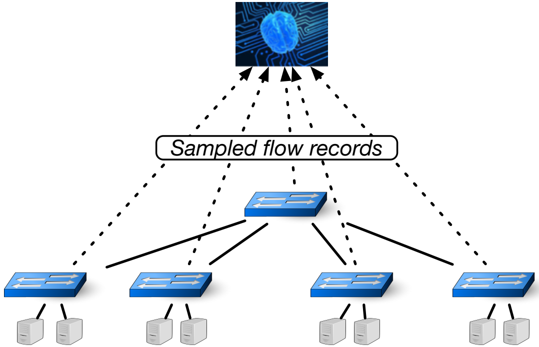
- High overhead



Sampling Switches

- Reduces accuracy

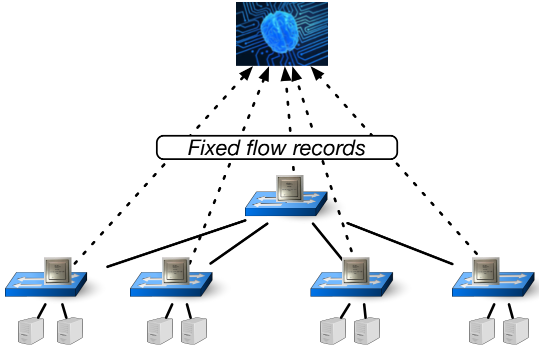
+Low overhead



Dedicated ASICs

- Limited feature set

+Low overhead



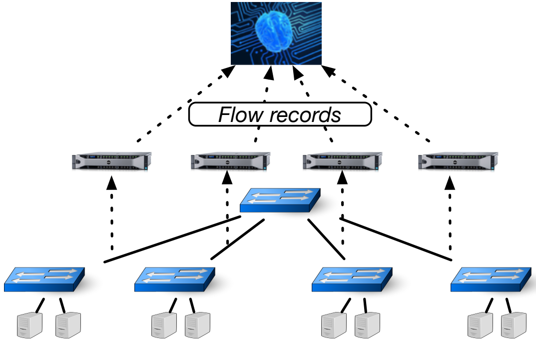
Introduction: our Goal

- Flow record generation that is *efficient*, *accurate*, and *feature rich*

Software generators

+ Information rich

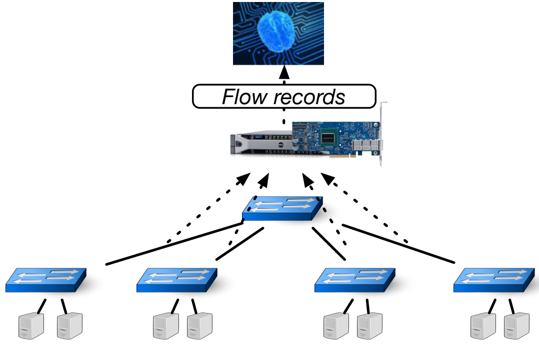
- High overhead



This work: software generation + P4 hardware acceleration

+ Information rich

+ Low overhead



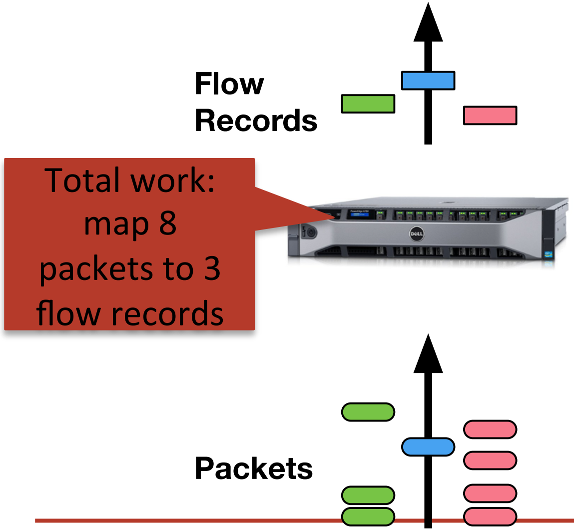
Outline

- Introduction: Flow Records
- **Design and Implementation: P4 Accelerated Flow Record Generation**
- Benchmarks and Optimizations

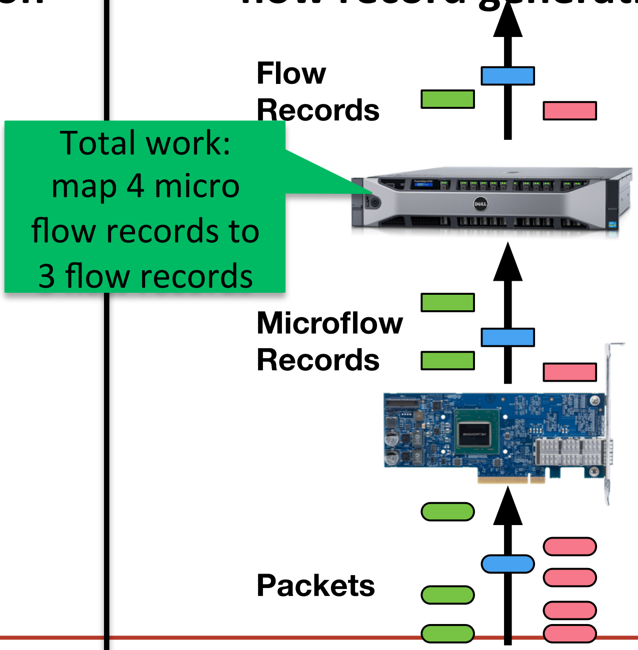
P4 Accelerated Flow Record Generation

- Main Idea: Use P4 hardware to preprocess packets into *micro flow records* that summarize per-flow packet bursts

Software flow record generation

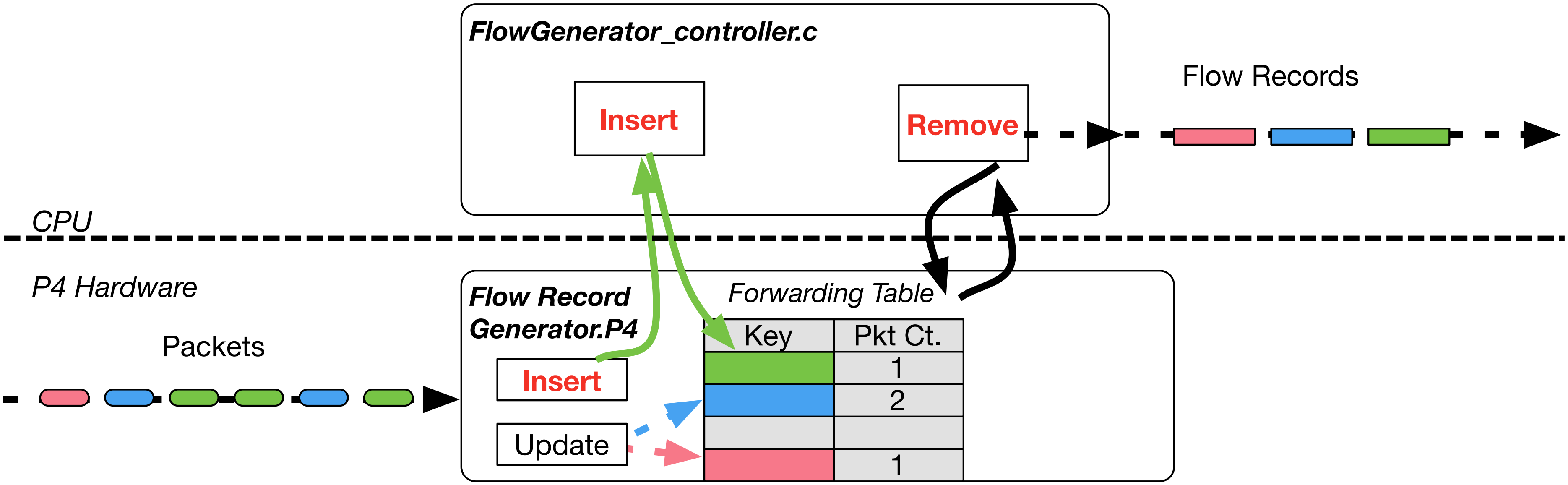


P4 accelerated flow record generation



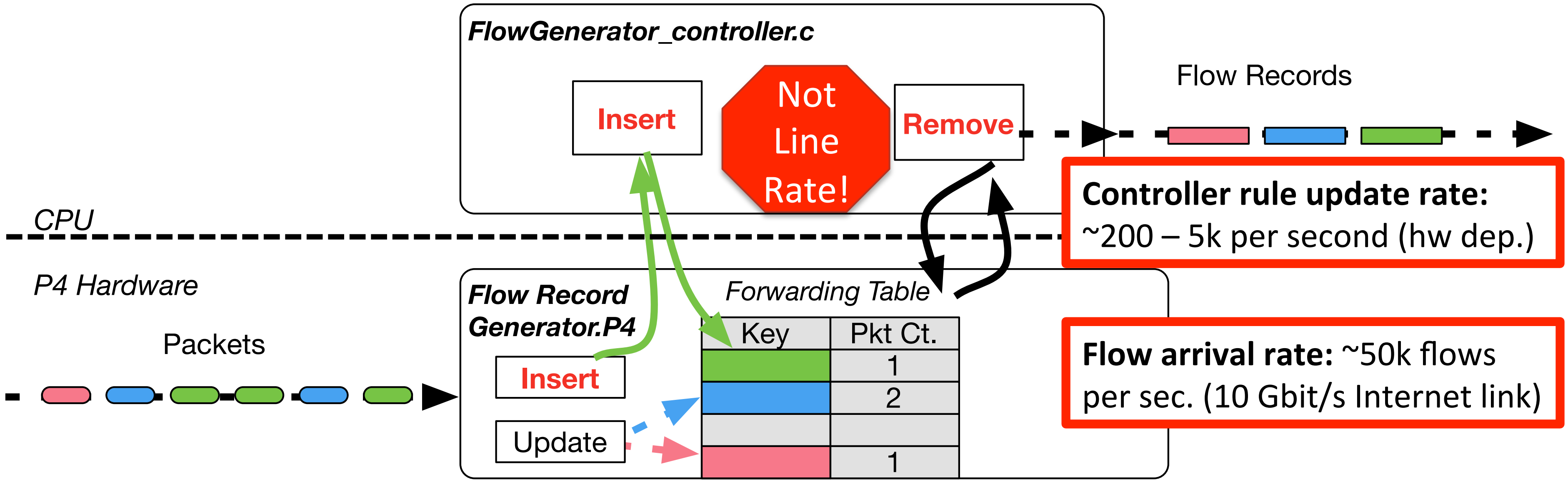
- **Information rich**
  - features are fully customizable
- **Efficient**
  - P4 hardware reduces CPU workload

First attempt: CPU Managed Tables



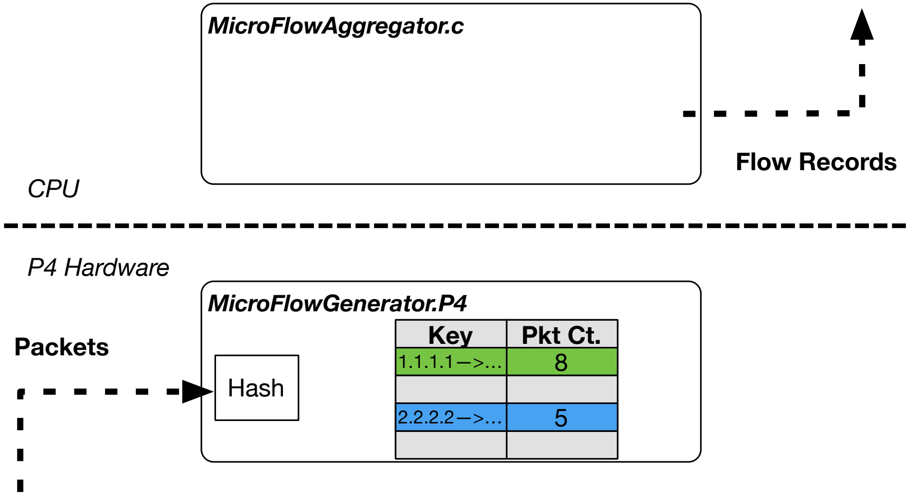
First attempt: CPU Managed Tables

- ***Bottleneck: table update operations***
- ***P4 designed for forwarding tables that are not highly dynamic***



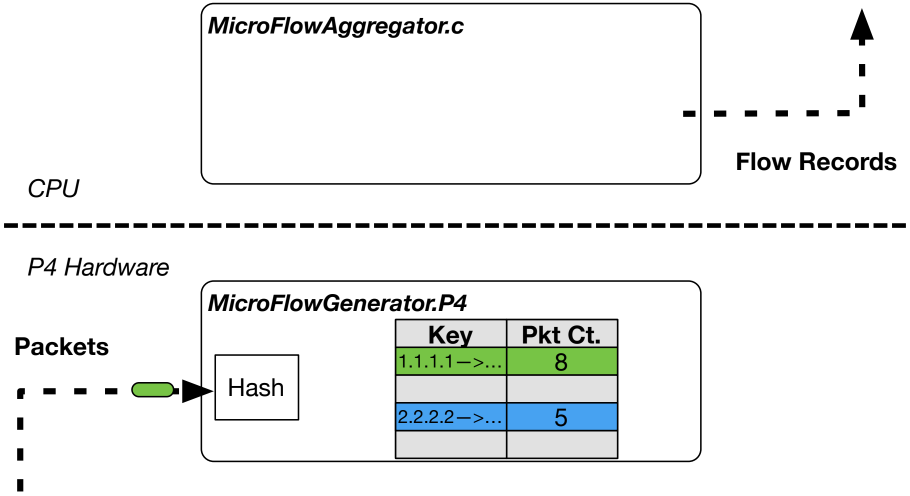
A Better Design: Minimal CPU Interaction

1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*



A Better Design: Minimal CPU Interaction

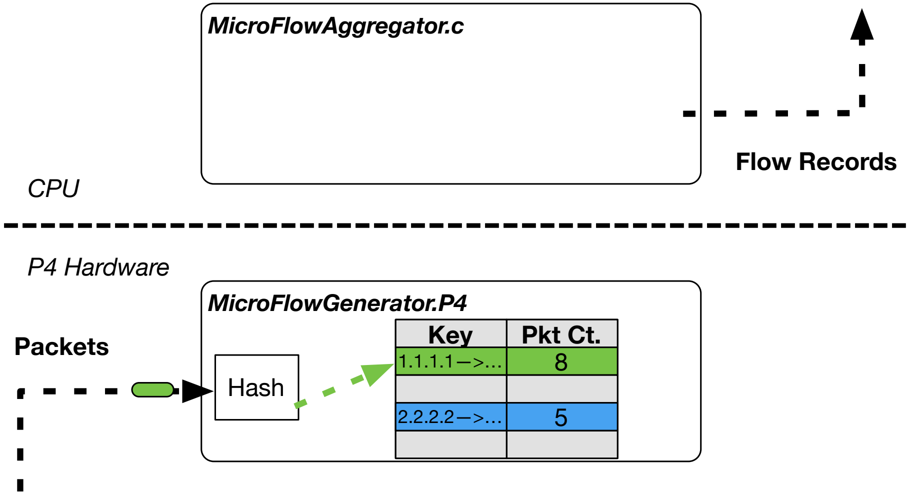
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);
```

A Better Design: Minimal CPU Interaction

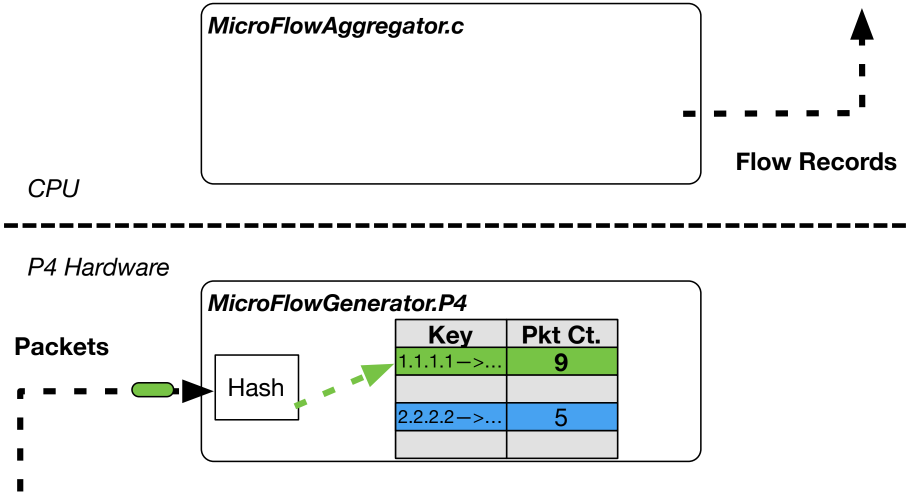
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);
```

A Better Design: Minimal CPU Interaction

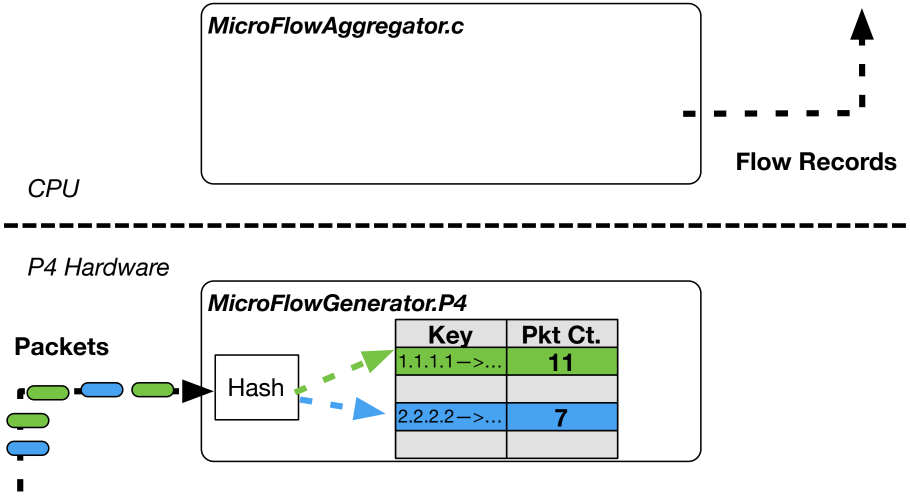
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
```

A Better Design: Minimal CPU Interaction

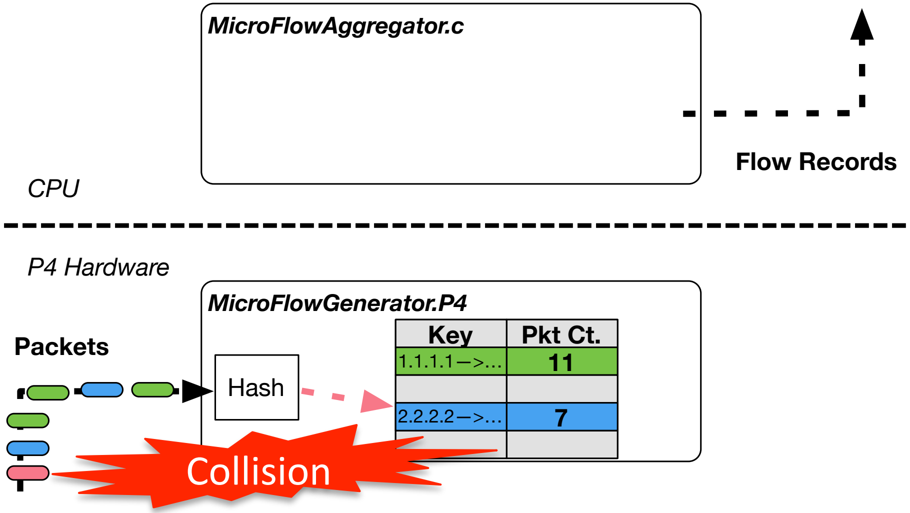
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
```

A Better Design: Minimal CPU Interaction

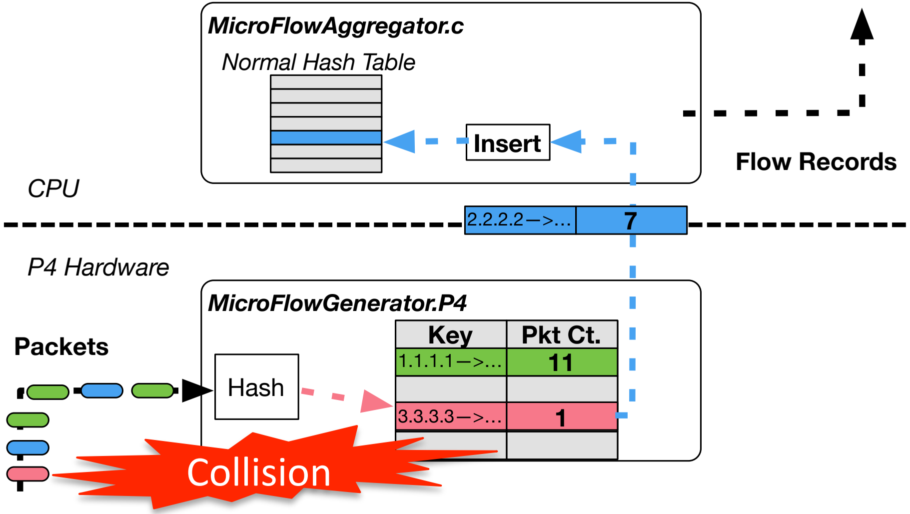
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
```

A Better Design: Minimal CPU Interaction

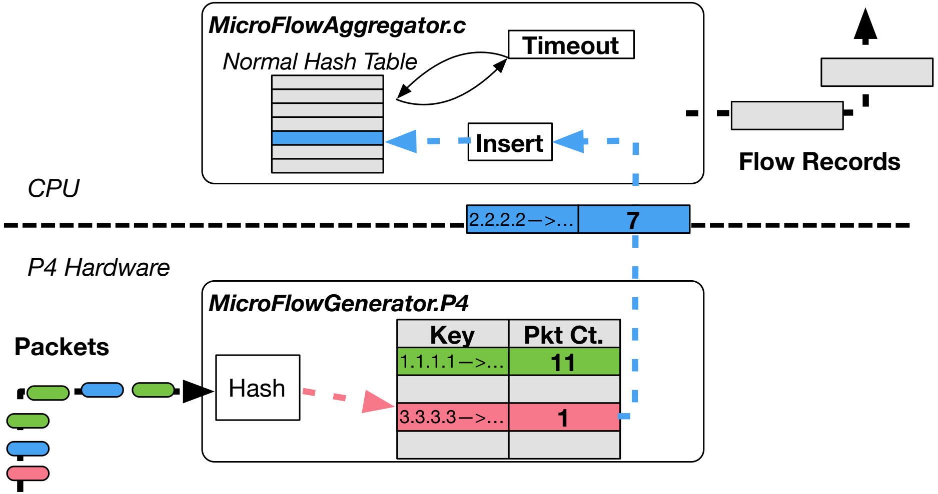
1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*
2. ***Evict to CPU on collision.***




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

A Better Design: Minimal CPU Interaction

1. *Use a flat array, i.e., P4 register, indexed by hash to store records.*
2. ***Evict to CPU on collision.***




```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

A Simple Implementation

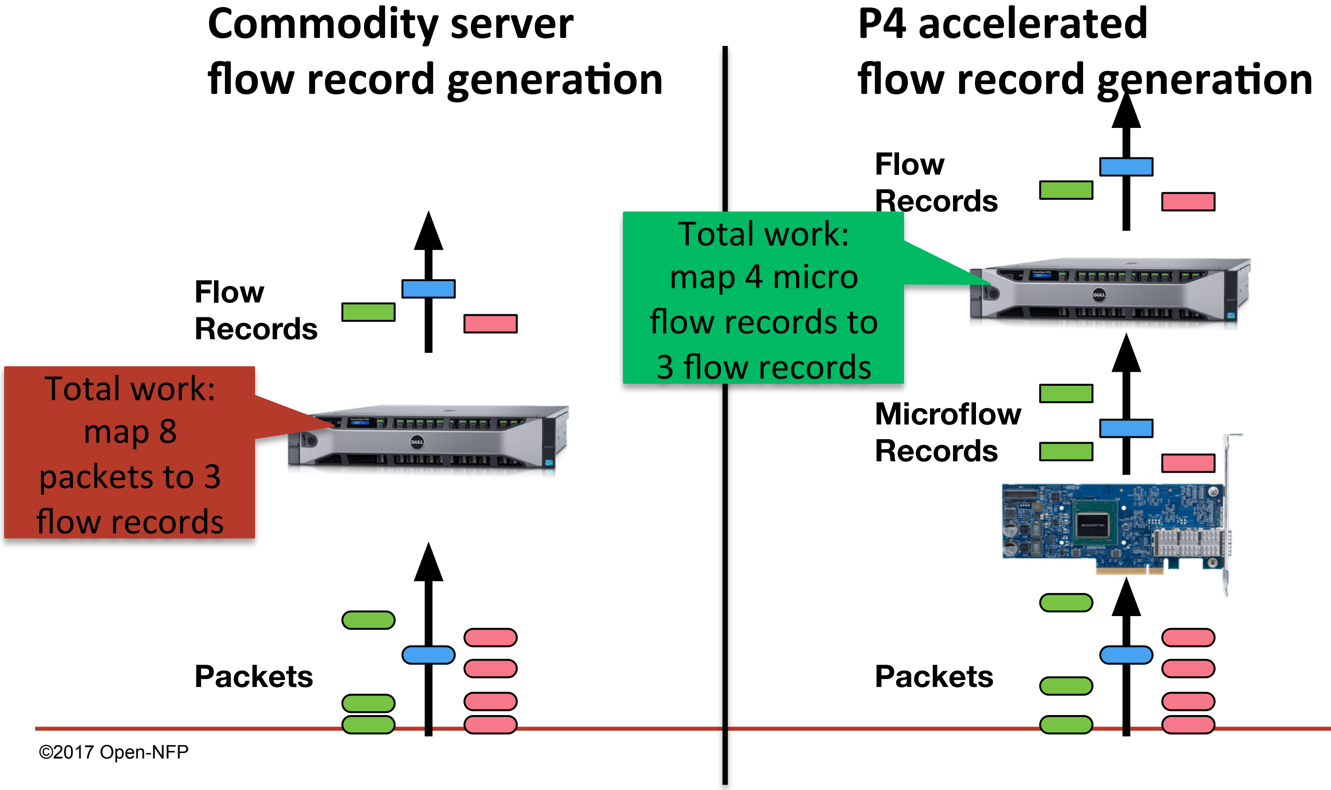
```
action compute_hash_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
}

action get_key_action(){
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}

action update_action(){
    // Read current values.
    register_read(temp_mf.packetCount, pktCt_reg, em.hash16);
    register_read(temp_mf.byteCount, byteCt_reg, em.hash16);
    // Increment or update.
    modify_field(temp_mf.packetCount, temp_mf.packetCount+1);
    modify_field(temp_mf.byteCount, temp_mf.byteCount+ipv4.totalLen);
    // Write back.
    register_write(pktCt_reg, em.hash16, temp_mf.packetCount);
    register_write(byteCt_reg, em.hash16, temp_mf.byteCount);
}

action replace_action(){
    // Read current feature values (need to send to CPU)
    register_read(temp_mf.packetCount, pktCt_reg, em.hash16);
    register_read(temp_mf.byteCount, byteCt_reg, em.hash16);
    // Reset key.
    register_write(src_reg, em.hash16, ipv4.srcAddr);
    register_write(dst_reg, em.hash16, ipv4.dstAddr);
    register_write(ports_reg, em.hash16, tcp.ports);
    // Reset features.
    register_write(pktCt_reg, em.hash16, 1);
    register_write(byteCt_reg, em.hash16, ipv4.totalLen);
}
action to_cpu_action(){
    // clone packet, fill with temp_mf (not shown).
    clone_ingress_to_ingress();
}
```

```
@pragma netro reglocked
register src_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register dst_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register ports_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register pktCt_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register byteCt_reg { width: 32; instance_count : TOTAL_SIZE; }
```

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

P4 Accelerated Flow Record Generation

- Main Idea: Use P4 hardware to preprocess packets into *micro flow records* that summarize per-flow packet bursts



- **Information rich**
  - features are fully customizable
- **Efficient**
  - P4 HW reduces cpu workload

Outline

- Introduction: Flow Records
- Design and Implementation: P4 Accelerated Flow Record Generation
- **Benchmarks and Optimizations**

Benchmarks and Optimizations

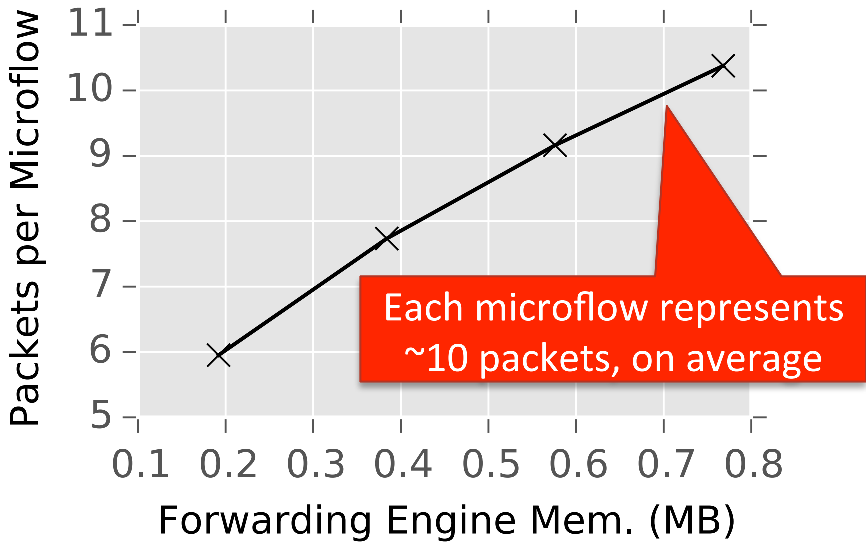
1. How much does the P4 hardware reduce CPU workload?
2. What is the maximum throughput of the P4 component?
3. How can we optimize for the NFP-4000?

Benchmarks and Optimizations

1. **How much does the P4 hardware reduce CPU workload?**
2. What is the maximum throughput of the P4 component?
3. How can we optimize for the NFP-4000?

Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?

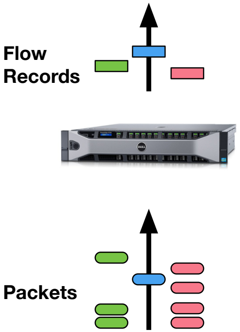


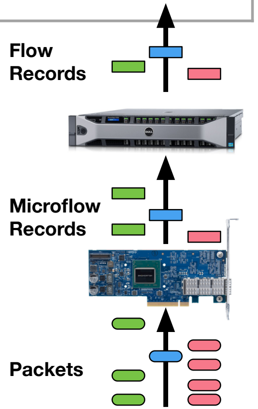
Workload: 1 hour 10 Gbit/s core router trace (CAIDA 02/2015 Chicago dir. A https://www.caida.org/data/passive/trace_stats/)

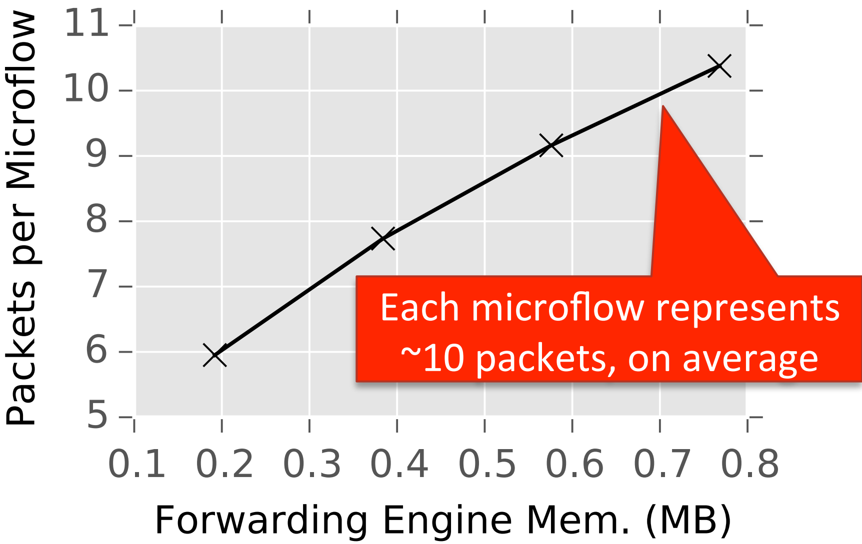
Benchmarks and Optimizations

1. *How much does the P4 hardware reduce CPU workload?*

Statistic	Packets aggregation	Microflow aggregation
Avg. rate for 10 Gbit/s link	~500k	~50k
CPU aggregator throughput (per core)	~600k	~600k
total CPU monitoring capacity (per core)	**~1 x10 Gbit/s link**	**~10 x 10 Gbit/s links**







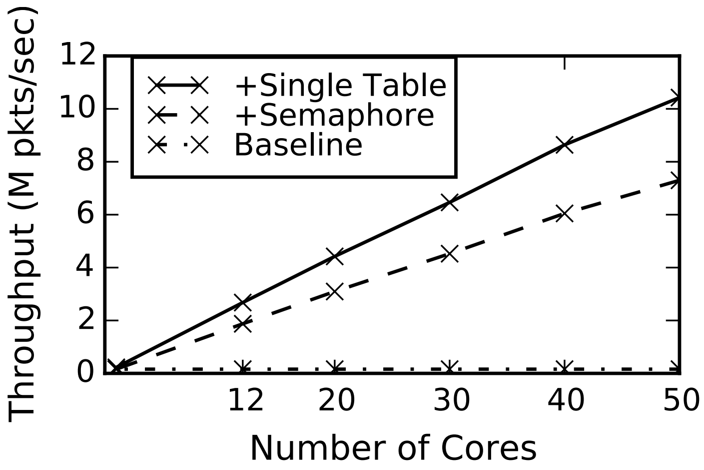
Workload: 1 hour 10 Gbit/s core router trace (CAIDA 02/2015 Chicago dir. A https://www.caida.org/data/passive/trace_stats/)

Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload? *(~10x with 1 MB of P4 HW memory)*
2. **What is the maximum throughput of the P4 component?**
3. How can we optimize it for the NFP-4000?

Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload? *(~10x with 1 MB of P4 HW memory)*
2. **What is the maximum throughput of the P4 component?**

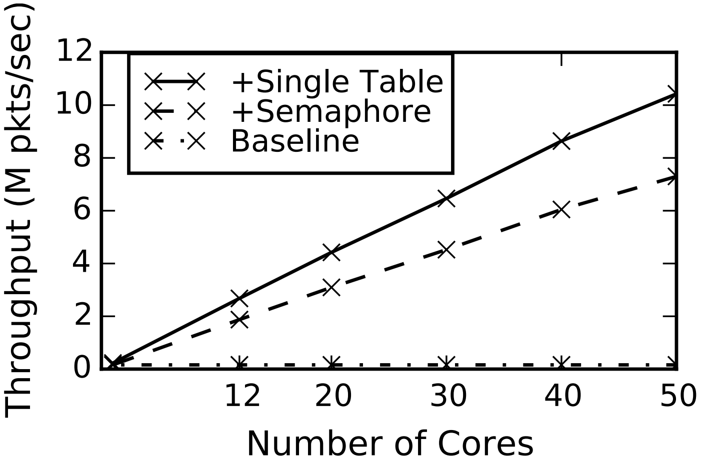


Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. **What is the maximum throughput of the P4 component?**

packet size	bit rate @ 10M pkts/sec
64	~ 5 Gbit/s
128	~ 10 Gbit/s
256	~ 20 Gbit/s
512	**~ 40 Gbit/s**
1024	**~ 80 Gbit/s**

Average packet sizes on 10 Gbit/s internet router links (caida)

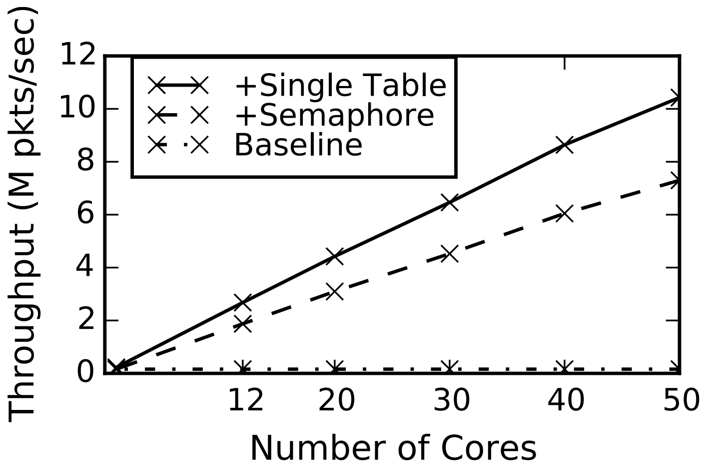


Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. **What is the maximum throughput of the P4 component?**
   (~40-80 Gbit/s with average size packets)

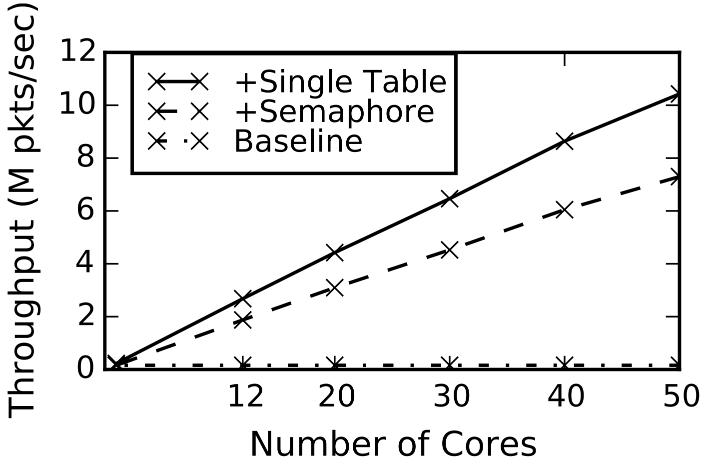
packet size	bit rate @ 10M pkts/sec
64	~ 5 Gbit/s
128	~ 10 Gbit/s
256	~ 20 Gbit/s
512	**~ 40 Gbit/s**
1024	**~ 80 Gbit/s**

Average packet sizes on 10 Gbit/s internet router links (caida)



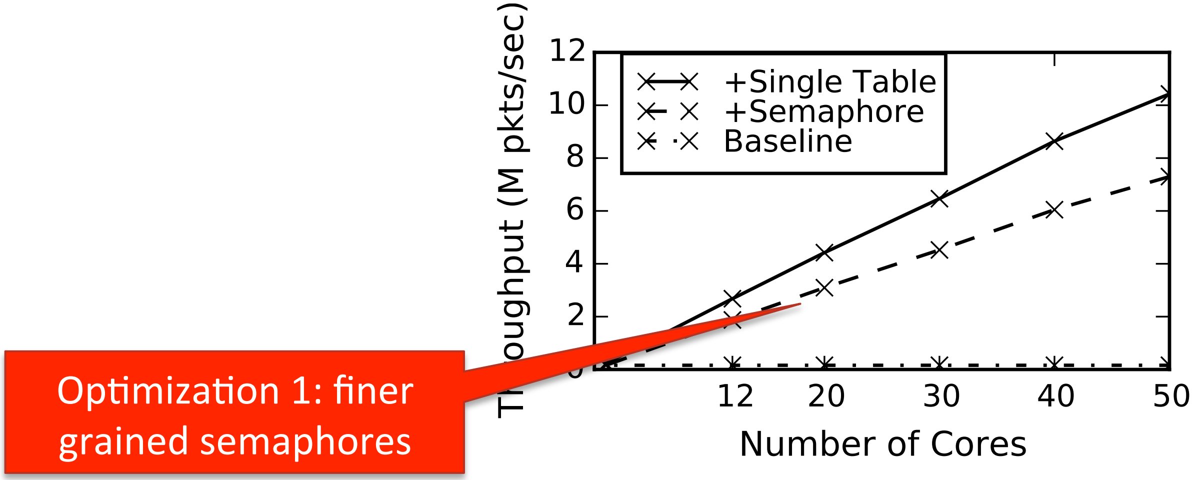
Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. What is the maximum throughput of the P4 component?
   (~40-80 Gbit/s with average size packets)
3. **How can we optimize for the NFP-4000?**



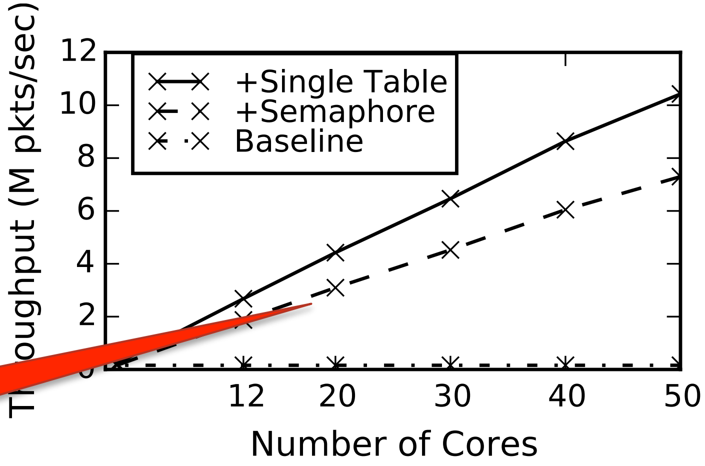
Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. What is the maximum throughput of the P4 component?
   (~40-80 Gbit/s with average size packets)
3. **How can we optimize for the NFP-4000?**



Benchmarks and Optimizations

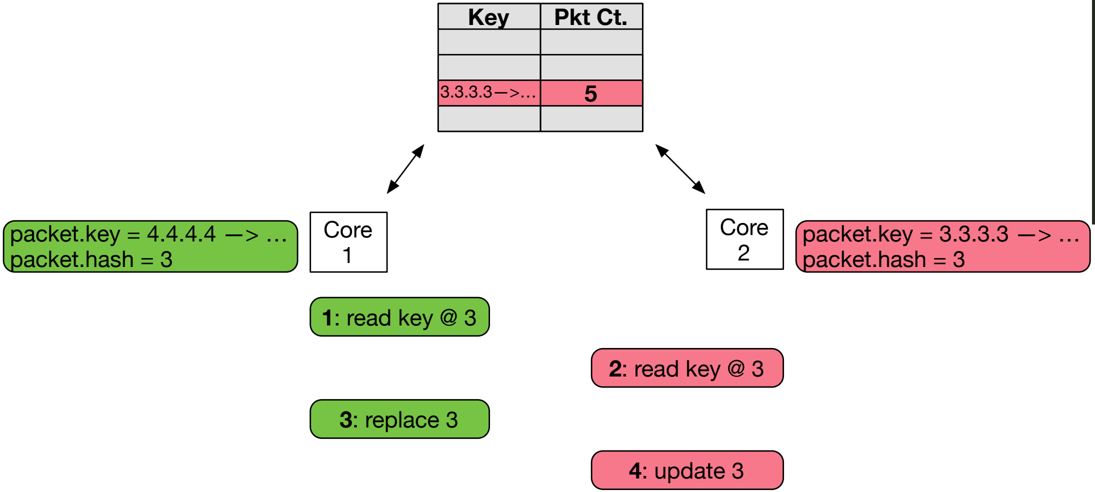
1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. What is the maximum throughput of the P4 component?
   (~40-80 Gbit/s with average size packets)
3. **How can we optimize for the NFP-4000?**



Optimization 1: finer grained semaphores

Optimization 1: Fine Grained P4 Semaphores

Q: Are there race conditions?



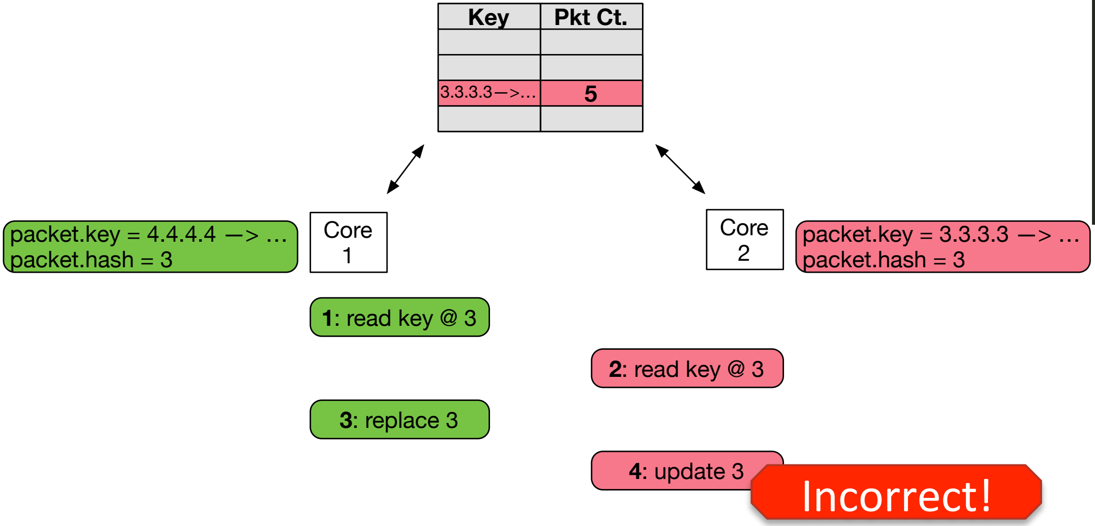

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

Optimization 1: Fine Grained P4 Semaphores

Q: Are there race conditions?

A: Yes.

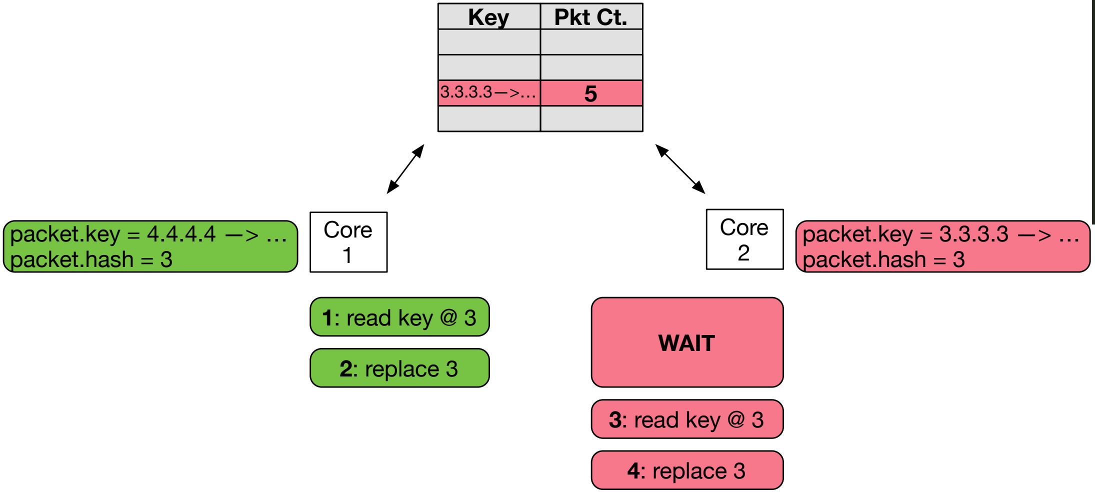
```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```



Optimization 1: Fine Grained P4 Semaphores

Q: Are there race conditions?

A: Yes.



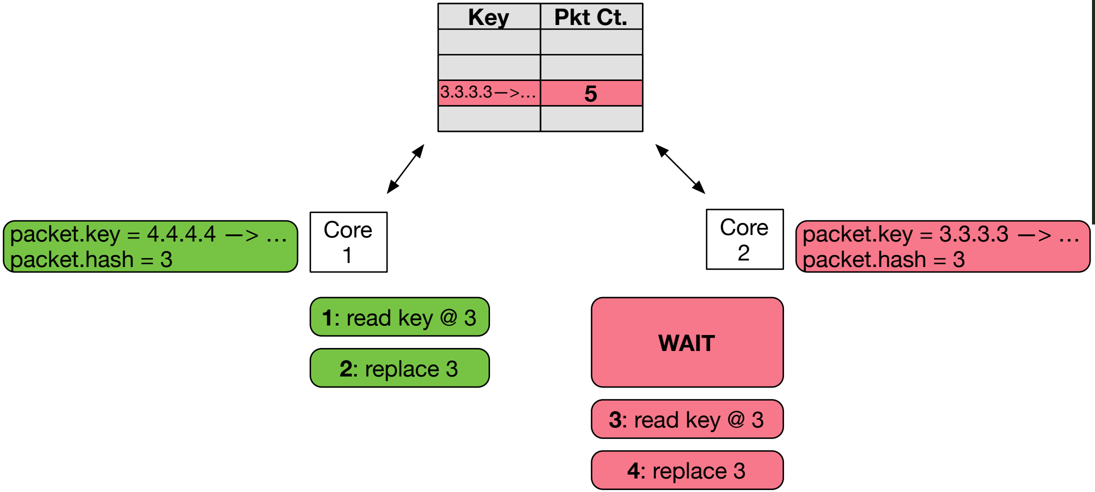

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

Optimization 1: Fine Grained P4 Semaphores

OpenNFP

```
@pragma netro reglocked
register src_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register dst_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register ports_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register pktCt_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register byteCt_reg { width: 32; instance_count : TOTAL_SIZE; }
```

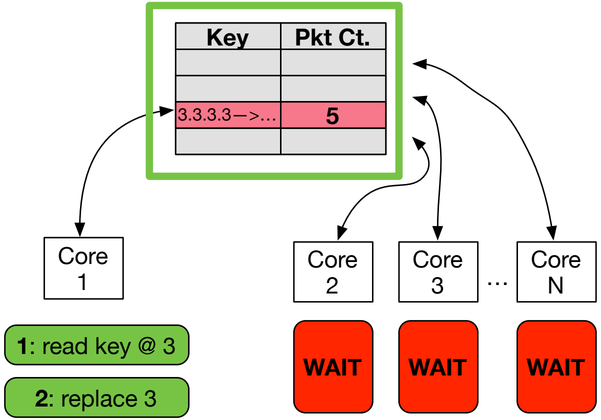
```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

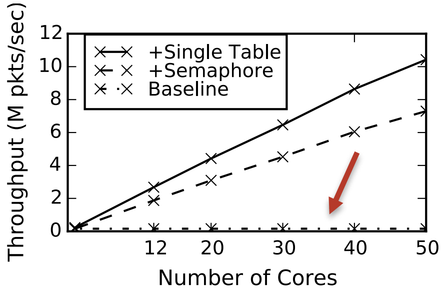


Optimization 1: Fine Grained P4 Semaphores

```
@pragma netro reglocked
register src_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register dst_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register ports_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register pktCt_reg { width: 32; instance_count : TOTAL_SIZE; }
@pragma netro reglocked
register byteCt_reg { width: 32; instance_count : TOTAL_SIZE; }
```

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```





Optimization 1: Fine Grained P4 Semaphores

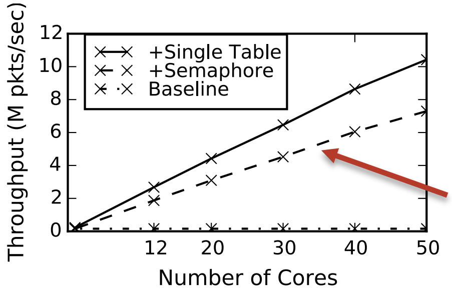
```
register src_reg { width: 32; instance_count : TOTAL_SIZE; }
register dst_reg { width: 32; instance_count : TOTAL_SIZE; }
register ports_reg { width: 32; instance_count : TOTAL_SIZE; }
register pktCt_reg { width: 32; instance_count : TOTAL_SIZE; }
register byteCt_reg { width: 32; instance_count : TOTAL_SIZE; }

#define MAX_SEM_ID 65535
header_type semaphore_t {
    fields {
        sid : 16;
    }
}
metadata semaphore_t sm;
// Lock semaphore @ sm.sid.
primitive_action lock_sem();
// Release semaphore @ sm.sid.
primitive_action release_sem();
```

lock/unlock by calling a P4 action.

```
action compute_hash_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
}

action get_key_action(){
    modify_field(sm.sid, em.hash16);
    lock_sem();
    // ...
}

action update_action(){
    // ...
    release_sem();
}

action replace_action(){
    // ...
    release_sem();
}
```

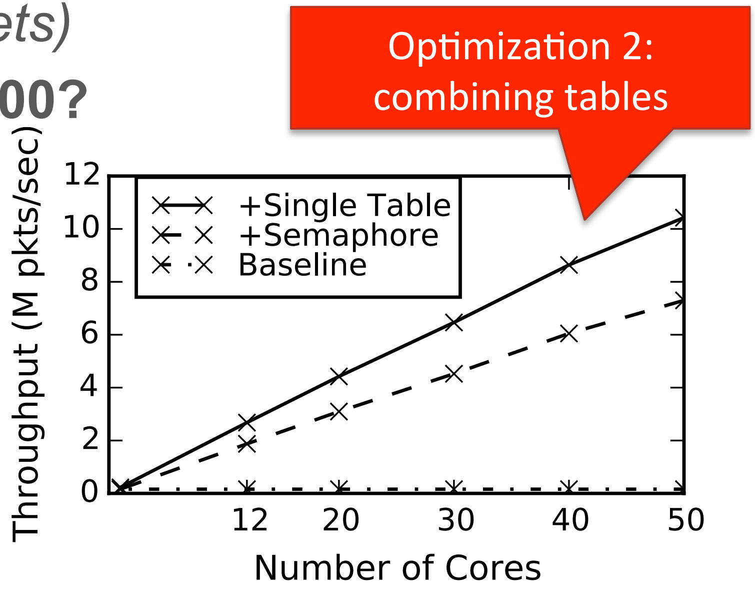
Lock a single entry, not the entire array.

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```



Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload? *(~10x with 1 MB of P4 HW memory)*
2. What is the maximum throughput of the P4 component? *(~40-80 Gbit/s with average size packets)*
3. **How can we optimize for the NFP-4000?**
   1. *Fine grained P4 semaphores*



Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)   == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}


action compute_hash_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
}

action get_key_action(){
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}
```

Version	Cycles / packet	Improvement
Baseline	5084	-

Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)   == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}


action compute_hash_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
}

action get_key_action(){
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}
```

```
    // // get index into microflow record table.
    // apply(compute_hash);
    // // get key of flow record at that position in table.
    // apply(get_key);
    // get index and load key.
    apply(compute_hash_and_get_key);
```

```
action compute_hash_and_get_key_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}
```

Version	Cycles / packet	Improvement
Baseline	5084	-
compute hash and get key merged	4191	18%

Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}

action compute_hash_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
}

action get_key_action(){
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}
```

```
    // // get index into microflow record table.
    // apply(compute_hash);
    // // get key of flow record at that position in table.
    // apply(get_key);
    // get index and load key.
    apply(compute_hash_and_get_key);
```

```
action compute_hash_and_get_key_action(){
    modify_field_with_hash_based_offset(em.hash16, 0, hashfcn, 16);
    register_read(temp_mf.srcAddr, src_reg, em.hash16);
    register_read(temp_mf.dstAddr, dst_reg, em.hash16);
    register_read(temp_mf.ports, ports_reg, em.hash16);
}
```

Changing tables is as expensive as ~5 emem ops (register_reads/writes).

Version	Cycles / packet	Improvement
Baseline	5084	-
compute hash and get key merged	4191	18%

Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)   == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

```
control update_microflow_table {
    apply(do_everything);
}
```

Can we merge *all* the tables? Challenge: branches.

Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)   == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

```
control update_microflow_table {
    apply(do_everything);
}
```

Can we merge *all* the tables? Challenge: branches.

```
action do_everything_action_psuedocode_1(){
    // compute hash.
    hash = compute_hash(packet.key);

    // load record in current slot.
    temp_mf.key = table[hash].key;
    temp_mf.features = table[hash].features;

    // compute not match flag.
    int collisionFlag = (pkt.key ^ temp_mf.key); // 0 if equal, else arbitrary.

    // conditionally update key.
    collisionFlag ? table[hash].key = pkt.key : table[hash].key = temp_mf.key;
    // conditionally update features.
    collisionFlag ? table[hash].pktCt = 1 : table[hash].pktCt = temp_mf.pktCt++;
    // conditionally set evict / clone flag.
    collisionFlag ? temp_mf.clone_flag = 1 : temp_mf.clone_flag = 0;
}
```

In C, we could use conditional assignments.

Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)  == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

```
control update_microflow_table {
    apply(do_everything);
}
```

Can we merge *all* the tables? Challenge: branches.

In P4, we can use a *conditional mask.*

```
action do_everything_action_psuedocode_2(){
    // compute hash.
    hash = compute_hash(packet.key);

    // load record in current slot.
    temp_mf.key = table[hash].key;
    temp_mf.features = table[hash].features;

    // compute not match flag.
    int collisionFlag = (pkt.key ^ temp_mf.key); // 0 if equal, else arbitrary.
    int collisionMask = (collisionFlag | -collisionFlag) >> 31; // 0 if equal, else 0xffffffff

    // conditionally update key.
    // collisionFlag ? table[hash].key = pkt.key : table[hash].key = temp_mf.key;
    table[hash].key = (collisionMask & pkt.key) | (~collisionMask & temp_mf.key);
    // conditionally update features.
    // collisionFlag ? table[hash].pktCt = 1 : table[hash].pktCt = temp_mf.pktCt++;
    table[hash].pktCt = (collisionMask & 1) | (~collisionMask & temp_mf.pktCt++);
    // conditionally set evict / clone flag.
    // collisionFlag ? temp_mf.clone_flag = 1 : temp_mf.clone_flag = 0;
    temp_mf.clone_flag = (collisionMask & 1) | (~collisionMask & 0);
}
```

Encode as modify_field + register_write in P4.

http://homolog.us/blogs/blog/2014/12/04/the-entire-world-of-bit-twiddling-hacks/

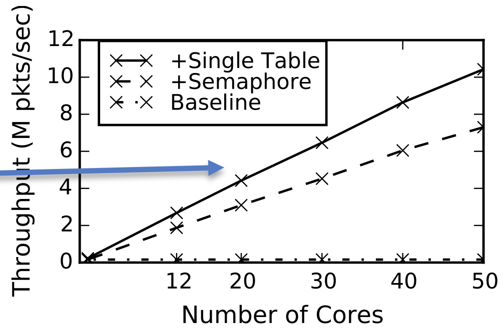
Optimization 2: Combining Tables

```
control update_microflow_table {
    // get index into microflow record table.
    apply(compute_hash);

    // get key of flow record at that position in table.
    apply(get_key);

    // if the packet belongs to that flow, update record.
    if ((temp_mf.srcAddr ^ ipv4.srcAddr) +
        (temp_mf.dstAddr ^ ipv4.dstAddr) +
        (temp_mf.ports   ^ tcp.ports)   == 0){
        apply (update_entry);
    }
    else{
        // else, replace with new record.
        apply(replace_entry);
        // send old record to cpu.
        apply(to_cpu);
    }
}
```

```
control update_microflow_table {
    apply(do_everything);
}
```

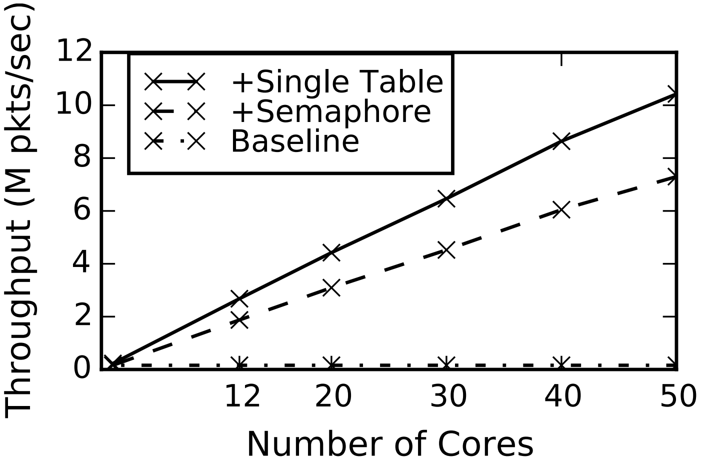
Version	Cycles / packet	Improvement
Baseline	5084	-
compute hash and get key merged	4191	18%
single table	**2915**	**42%**

End result: ~20-30% better throughput.



Benchmarks and Optimizations

1. How much does the P4 hardware reduce CPU workload?
   (~10x with 1 MB of P4 HW memory)
2. What is the maximum throughput of the P4 component?
   (~40-80 Gbit/s with average size packets)
3. ***How can we optimize for the NFP-4000?***
   1. *Fine grained P4 semaphores*
   2. *Combine tables, use conditional masks*



Summary

- Flow records are powerful for **high coverage, low overhead** network monitoring.
- Generating them **efficiently**, without sacrificing **information richness**, is a challenge.
- Our P4 accelerator can increase flow generator capacity by factor of 10 or more, ***without sacrificing information richness.***
- Table merging, conditional masks, and P4 accessible semaphores are **useful and portable optimization techniques**.
- Code available (soon) at: https://github.com/jsonch/p4_code
- Thank you for attending!

THANK YOU!