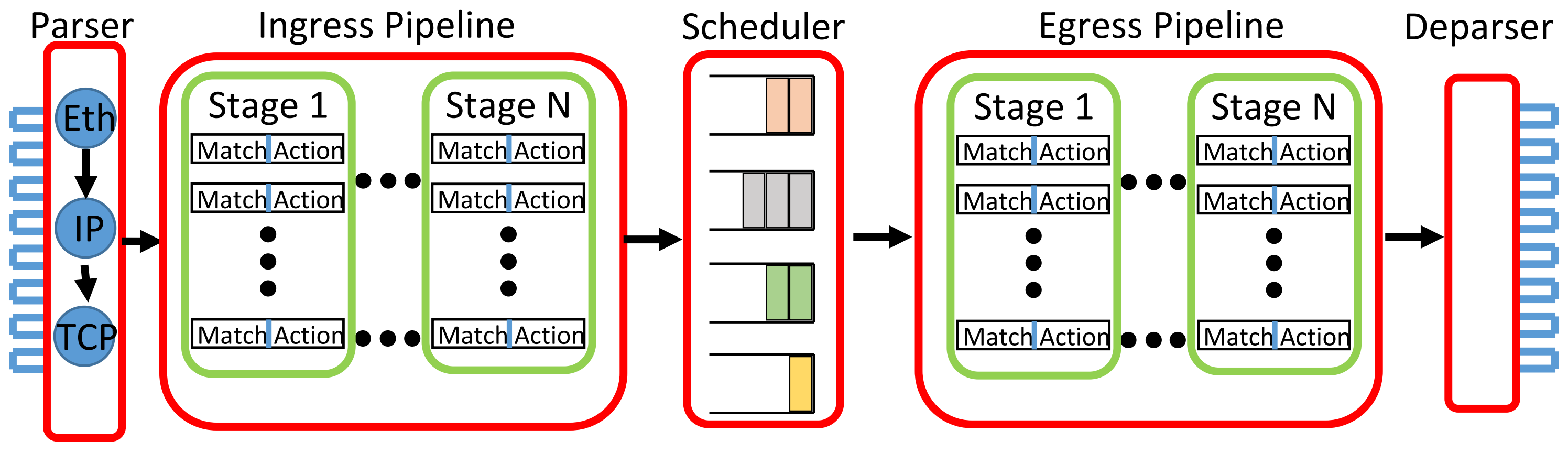
Packet Transactions: Programming the Data Plane at Line Rate

Anirudh Sivaraman, Mihai Budiu, Alvin Cheung, Changhoon Kim, Steve Licking, George Varghese, Hari Balakrishnan, Mohammad Alizadeh, Nick McKeown

Programming the data-plane at line rate

- Programmable: Can we express a new data-plane algorithm?
- Line-rate: Highest capacity supported by a communication standard

Programmability at line-rate



- OpenFlow: Match-Action interface, fixed fields, fixed actions
- P4, RMT, FlexPipe, Xpliant: Protocol-independent match-action pipeline.

Isn't P4 sufficient?

- Match-action is perfect for forwarding
- But, limiting for stateful algorithms

Packet Transactions

- Imperative code block in subset of C (domino) that is atomic and isolated from other such blocks
- One packet transaction per pipeline
- More familiar to NPU, Click programmers

Programming with Packet Transactions

Domino

```
#define NUM_FLOWLETS 8000
#define THRESHOLD    5
#define NUM_HOPS     10

struct Packet { int sport; int dport; ...};
int last_time [NUM_FLOWLETS] = {0};
int saved_hop [NUM_FLOWLETS] = {0};

void flowlet(struct Packet pkt) {
  pkt.new_hop = hash3(pkt.sport, pkt.dport, pkt.arrival)
                   % NUM_HOPS;
  pkt.id  = hash2(pkt.sport, pkt.dport) % NUM_FLOWLETS;
  if (pkt.arrival - last_time[pkt.id] > THRESHOLD) {
   saved_hop[pkt.id] = pkt.new_hop;
  }
  last_time[pkt.id] = pkt.arrival;
  pkt.next_hop = saved_hop[pkt.id];
}
```

P4

Stage 1

```
pkt.new_hop = hash3(pkt.sport,
                    pkt.dport,
                    pkt.arrival)
                %NUM_HOPS;
pkt.id = hash2(pkt.sport, pkt.dport)
           % NUM_FLOWLETS
```

Stage 2

```
pkt.last_time = last_time[pkt.id];
last_time[pkt.id] = pkt.arrival;
```

Stage 3

```
pkt.tmp = pkt.arrival – pkt.last_time;
```

Stage 4

```
pkt.tmp2 = pkt.tmp > 5;
```

Stage 5

```
pkt.saved_hop = saved_hop[pkt.id];
saved_hop[pkt.id] = pkt.tmp2 ?
                  pkt.new_hop :
                  pkt.saved_hop;
```

Stage 6

```
pkt.next_hop = pkt.tmp2 ?
             pkt.new_hop :
             pkt.saved_hop ;
```

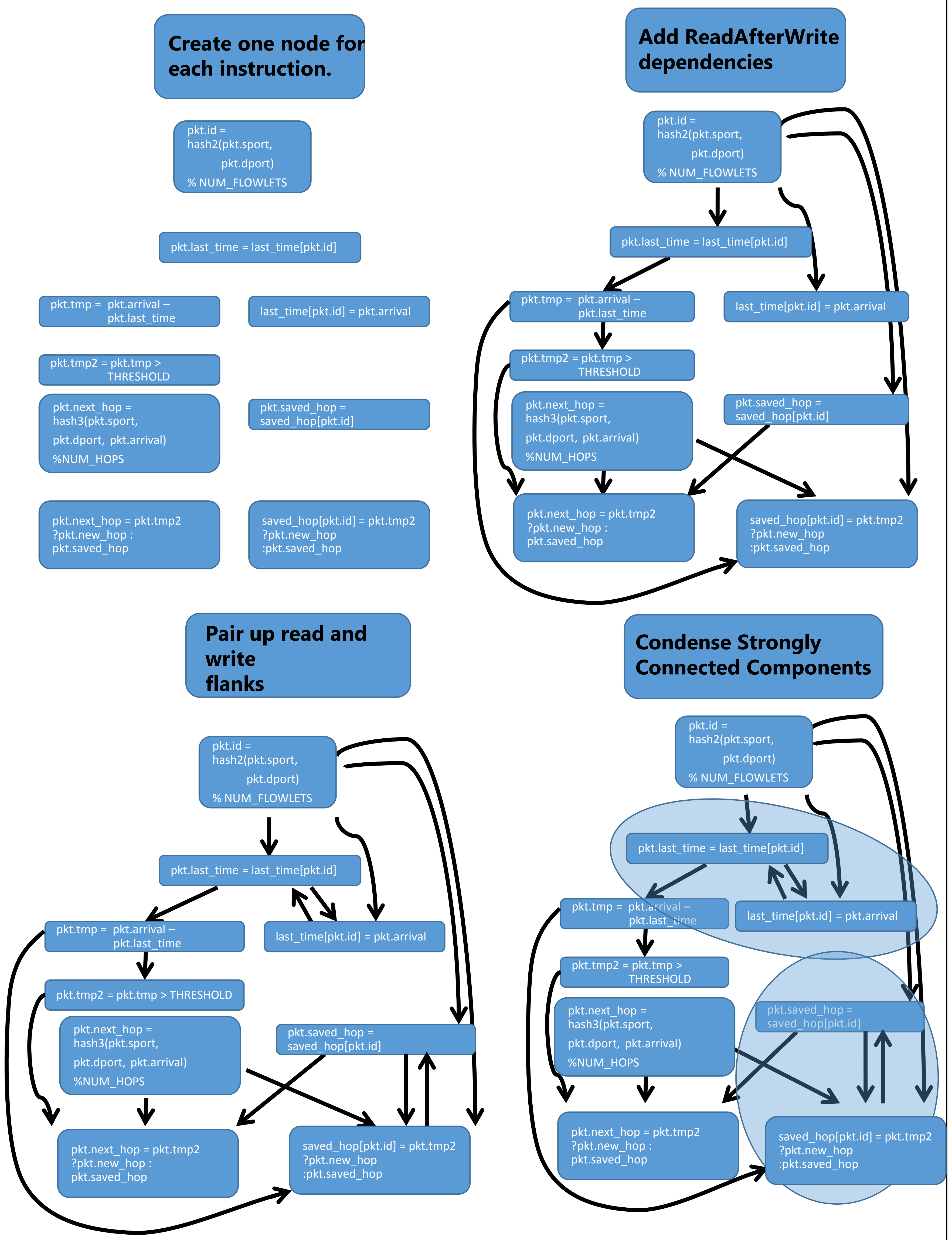
Compilation steps

If Conversion:
Rewrite branches into conditional operators

Read Write Flanks:
Read state variables into packet variables

Static Single Assignment Form:
Rename variables to have unique names

Critical Path Scheduling



Create one node for each instruction.

- pkt.id = hash2(pkt.sport, pkt.dport) % NUM_FLOWLETS
- pkt.last_time = last_time[pkt.id]
- pkt.tmp = pkt.arrival – pkt.last_time
- last_time[pkt.id] = pkt.arrival
- pkt.tmp2 = pkt.tmp > THRESHOLD
- pkt.next_hop = hash3(pkt.sport, pkt.dport, pkt.arrival) %NUM_HOPS
- pkt.saved_hop = saved_hop[pkt.id]
- pkt.next_hop = pkt.tmp2 ?pkt.new_hop : pkt.saved_hop
- saved_hop[pkt.id] = pkt.tmp2 ?pkt.new_hop :pkt.saved_hop

Add ReadAfterWrite dependencies

- pkt.id = hash2(pkt.sport, pkt.dport) % NUM_FLOWLETS
- pkt.last_time = last_time[pkt.id]
- pkt.tmp = pkt.arrival – pkt.last_time
- last_time[pkt.id] = pkt.arrival
- pkt.tmp2 = pkt.tmp > THRESHOLD
- pkt.next_hop = hash3(pkt.sport, pkt.dport, pkt.arrival) %NUM_HOPS
- pkt.saved_hop = saved_hop[pkt.id]
- pkt.next_hop = pkt.tmp2 ?pkt.new_hop : pkt.saved_hop
- saved_hop[pkt.id] = pkt.tmp2 ?pkt.new_hop :pkt.saved_hop

Pair up read and write flanks

- pkt.id = hash2(pkt.sport, pkt.dport) % NUM_FLOWLETS
- pkt.last_time = last_time[pkt.id]
- pkt.tmp = pkt.arrival – pkt.last_time
- last_time[pkt.id] = pkt.arrival
- pkt.tmp2 = pkt.tmp > THRESHOLD
- pkt.next_hop = hash3(pkt.sport, pkt.dport, pkt.arrival) %NUM_HOPS
- pkt.saved_hop = saved_hop[pkt.id]
- pkt.next_hop = pkt.tmp2 ?pkt.new_hop : pkt.saved_hop
- saved_hop[pkt.id] = pkt.tmp2 ?pkt.new_hop :pkt.saved_hop

Condense Strongly Connected Components

- pkt.id = hash2(pkt.sport, pkt.dport) % NUM_FLOWLETS
- pkt.last_time = last_time[pkt.id]
- pkt.tmp = pkt.arrival – pkt.last_time
- last_time[pkt.id] = pkt.arrival
- pkt.tmp2 = pkt.tmp > THRESHOLD
- pkt.next_hop = hash3(pkt.sport, pkt.dport, pkt.arrival) %NUM_HOPS
- pkt.saved_hop = saved_hop[pkt.id]
- pkt.next_hop = pkt.tmp2 ?pkt.new_hop : pkt.saved_hop
- saved_hop[pkt.id] = pkt.tmp2 ?pkt.new_hop :pkt.saved_hop

Generating P4 code

- Required changes to P4
  - Sequential execution semantics
  - Expression support
  - Both available in v1.1
- Encapsulate every SCC in a default action