

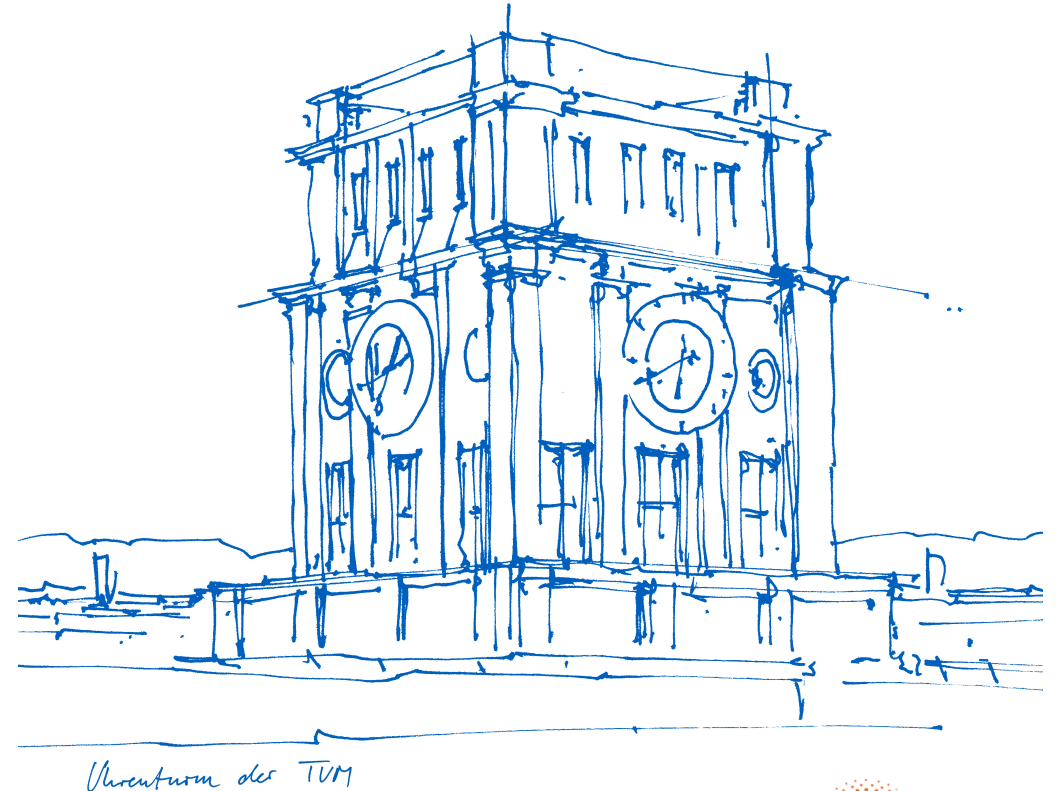
# Toward Consistent State Management of Adaptive Programmable Networks Based on P4

SIGCOMM – NEAT Workshop 2019,  
Beijing, August 19<sup>th</sup>, 2019

**Mu He**

[Mu.he@tum.de](mailto:Mu.he@tum.de)

Andreas Blenk, Stefan Schmid, Wolfgang Kellerer



This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program grant agreement No 647158 – **FlexNets (2015 – 2020)**.

# New Applications and Technologies



Augmented  
Reality



Virtual  
Reality

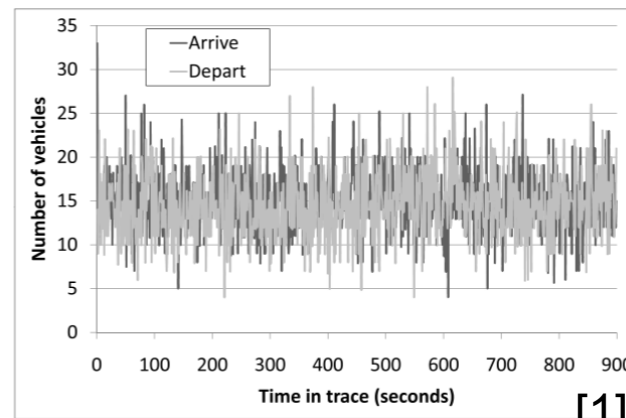


Tactile



Vehicular  
Networks

Require  
latency, pr



throughput,  
omizability!

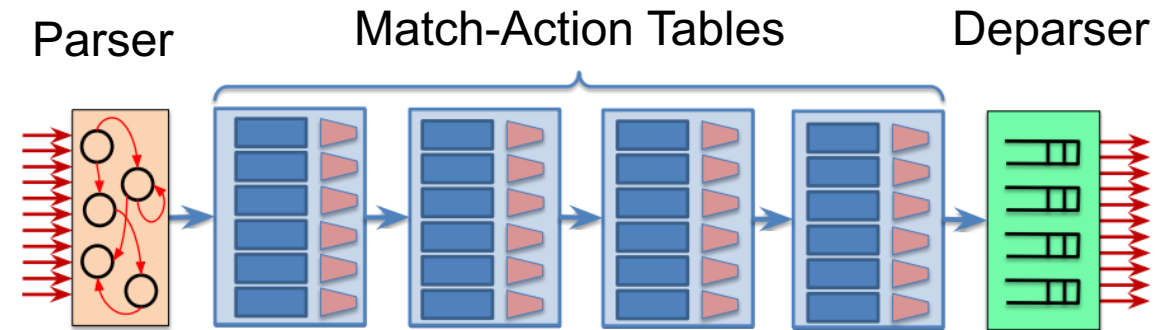


[1]

[1] Rowstron A, Pau G. Characteristics of a vehicular network[J]. University of California Los Angeles, Computer Science Department, Tech. Rep, 2009: 09-0017.

# Programmable Networks with P4

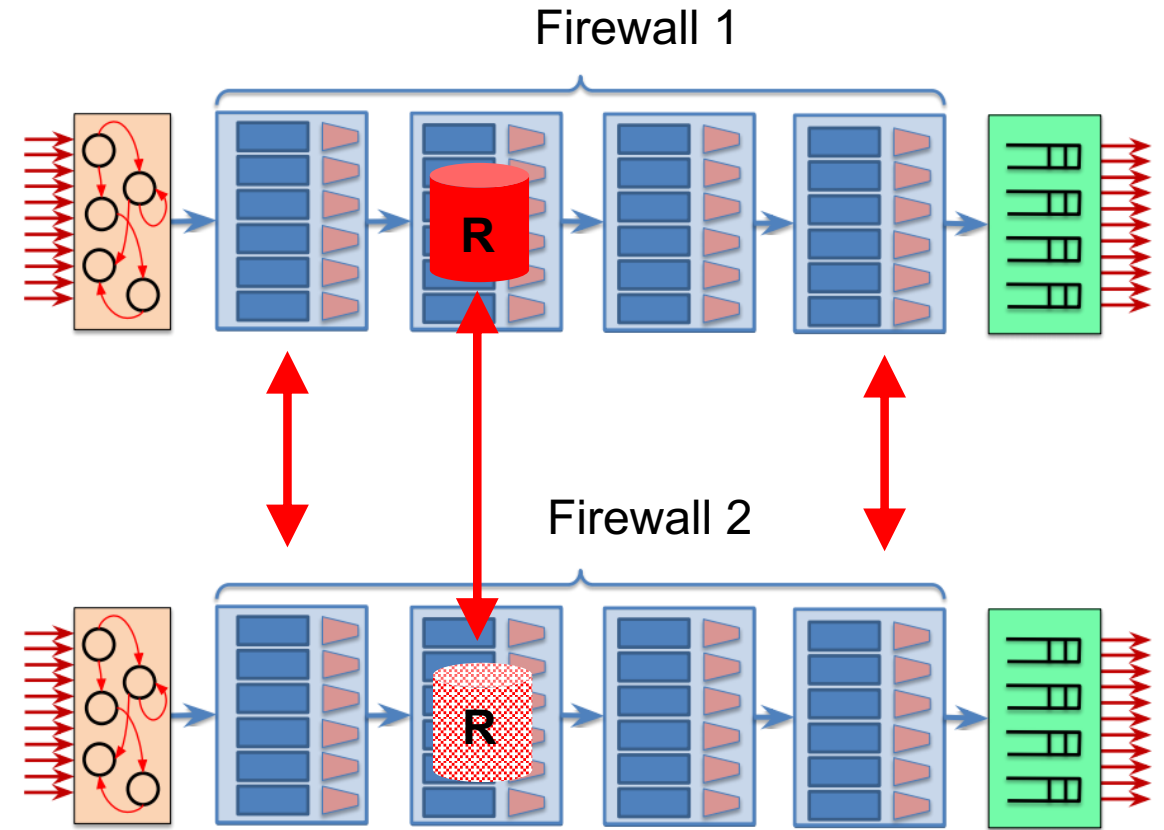
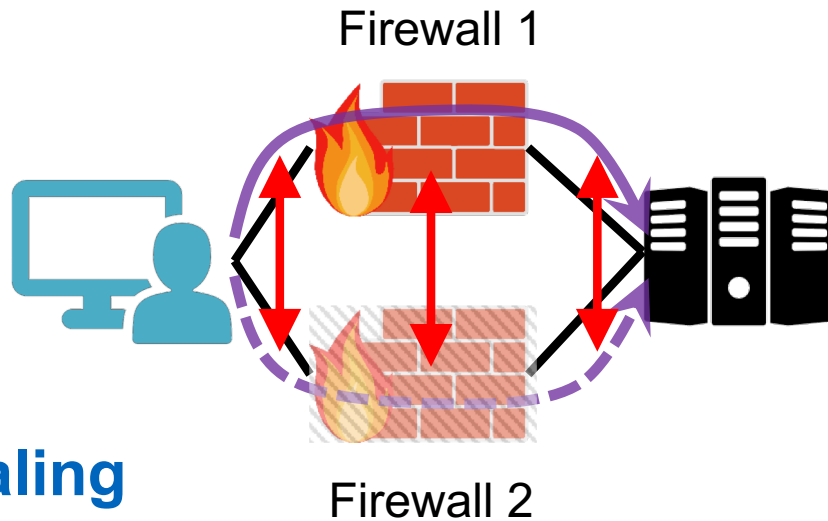
- Program your own data plane [1]
  - Protocol independent
  - Target independent
  - Field reconfigurable
- Advantages in many use cases
  - In-band network telemetry [2]
  - Flow monitoring [3]
  - Load balancing [4]
  - High throughput data center [5]



- [1] Bosshart, Pat, et al. "P4: Programming protocol-independent packet processors." ACM SIGCOMM Computer Communication Review 44.3 (2014): 87-95.
- [2] Gupta, Arpit, et al. "Sonata: Query-driven streaming network telemetry." SIGCOMM. ACM, 2018.
- [3] Liu, Zaoxing, et al. "One sketch to rule them all: Rethinking network flow monitoring with univmon." SIGCOMM. ACM, 2016.
- [4] Miao, Rui, et al. "Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics." SIGCOMM. ACM, 2017.
- [5] Handley, Mark, et al. "Re-architecting datacenter networks and stacks for low latency and high performance." SIGCOMM. ACM, 2017.

# Reconfiguration is needed!

## NF Scaling

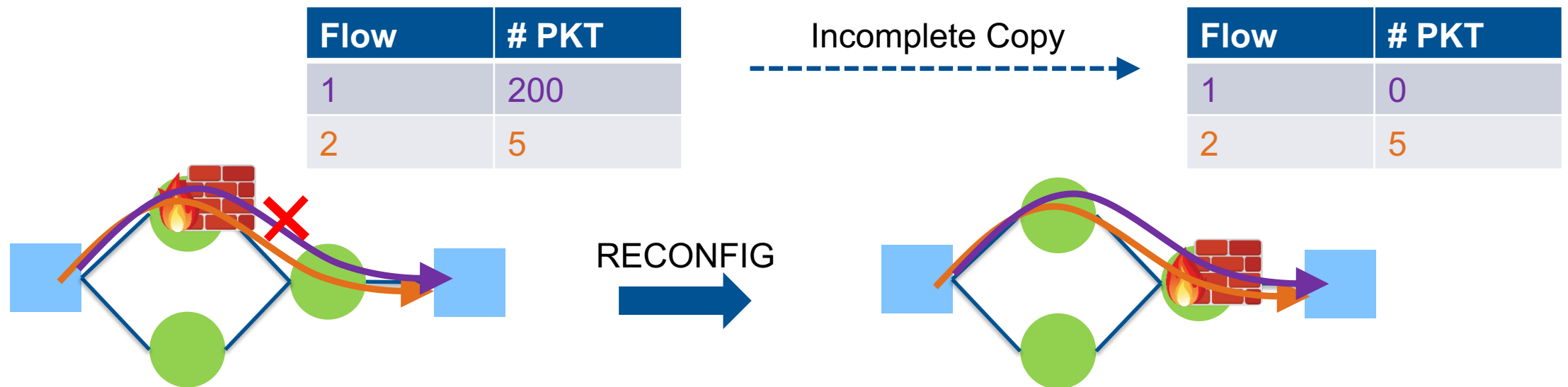


Field reconfigurability + Data plane states

How to manage them?

# State Consistency during Network Reconfiguration

- Stateful firewall based on packet counts (More than 100)
- State: # packets per flow



## Completeness

All necessary state variables

Updated version

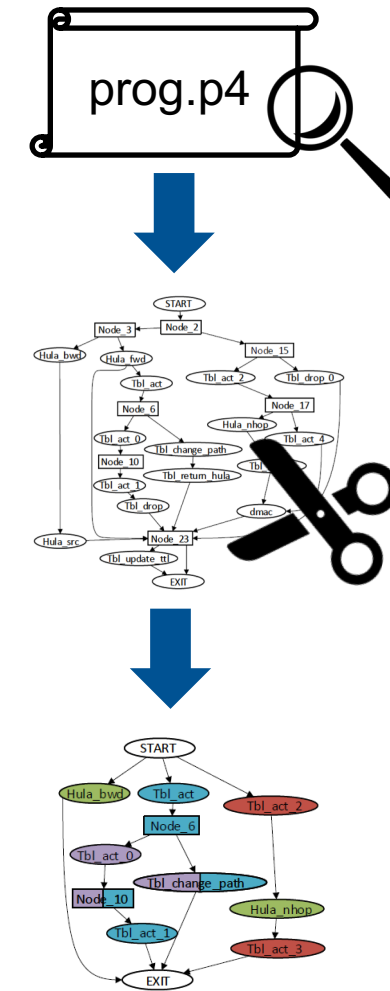
## Rapidness

Finish as fast as possible

Agility of data plane

# Our Proposal: P4State Analyzer

- States identification
  - Collect all state usage
  - Bind states to tables and if-conditionals
  
- Control Flow Graph (CFG) construction
  - Abstract match + action pipeline
  
- Control Flow Graph pruning
  - Delete stateless nodes
  
- Path and role identification



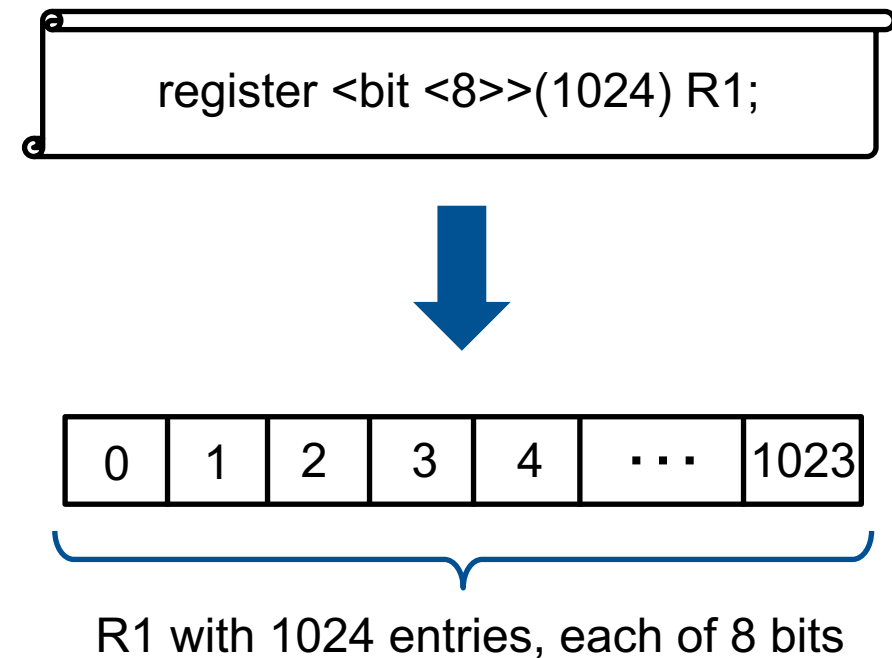
# State Variables in P4: Taxonomy

- Various types of states in P4 context
  - Table entries  $\leftarrow$  control plane
  - Temporary metadata: local information while processing each packet
  - **Stateful variables: register**
- Persistent beyond the packet processing loop, e.g.,
  - Packet counter
  - Port status
  - Pipeline control flag



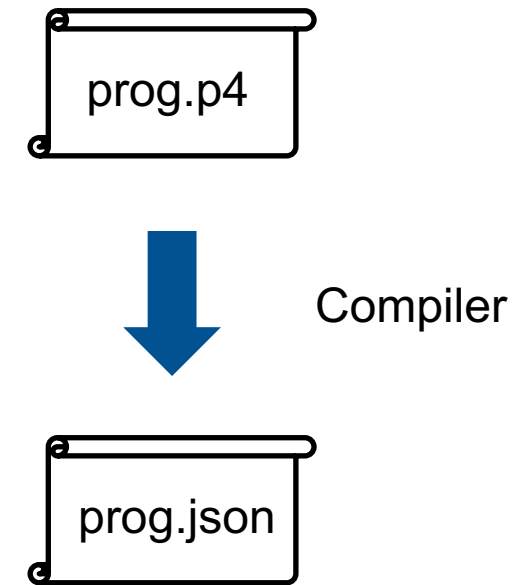
# State Variables in P4: Taxonomy

- Register = An array with values
  - Register type
  - Register entry
- Flow-based registers
  - Information per-flow (different granularity)
  - Large number of entries
  - Migrated through data plane or control plane
- Device-based registers
  - Device and pipeline information
  - Migrated through control plane



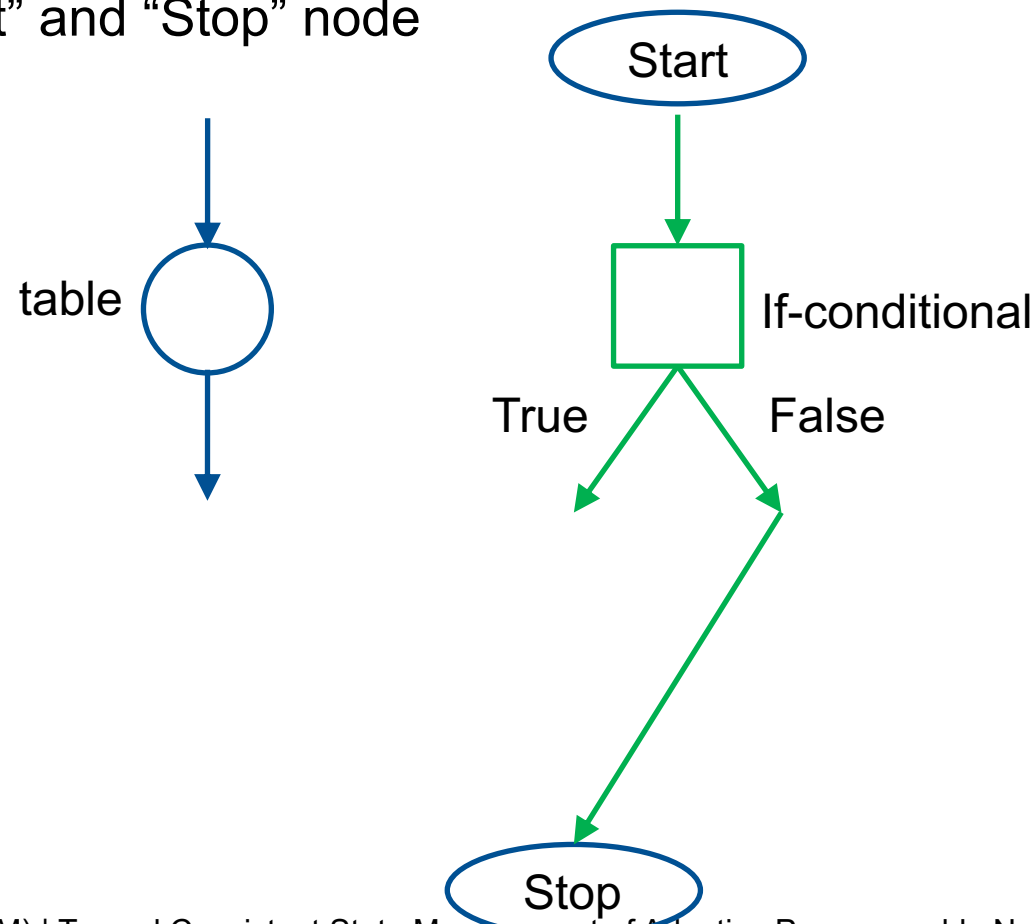
# States Identification

- Input: P4 program
- Output: Set of tables and if-conditionals with register access
- Compile the program to a JSON file
  - Consists definitions of all pipeline components
- Parse JSON file
  - Collect all register definitions
  - Collect all header, action, table, if-conditional definitions
  - Associate registers to tables and if-conditionals



# Control Flow Graph (CFG)

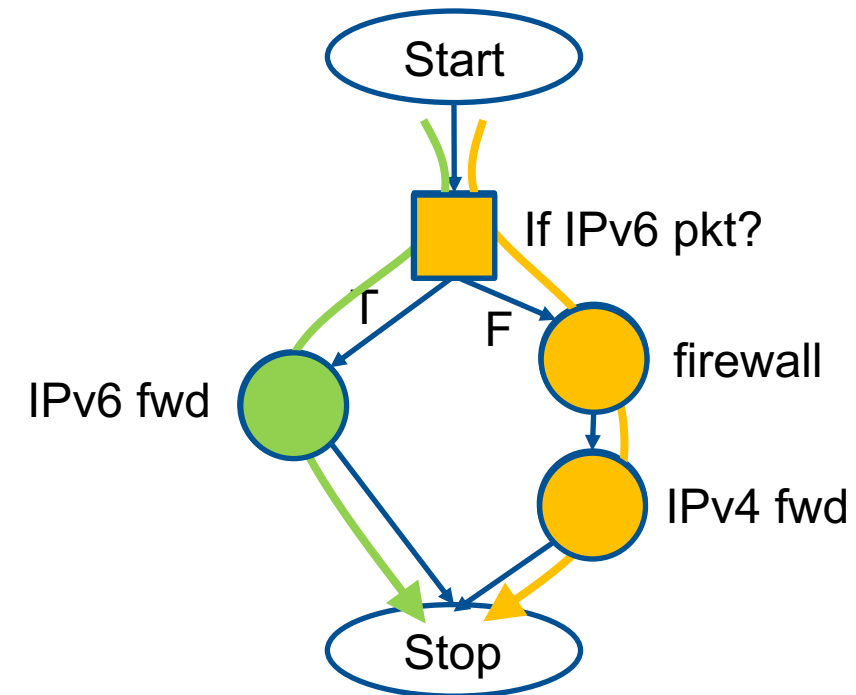
- Describes the packet processing pipeline
- Composed of **tables** and **if-conditionals**
- “Start” and “Stop” node



```
table ipv4_lpm {  
    key = {  
        hdr.ipv4.dstAddr: lpm;  
    }  
    actions = {  
        ipv4_forward;  
        drop;  
    }  
    size = 1024;  
}  
  
apply {  
    if (hdr.ipv4.isValid()) {  
        ipv4_lpm.apply();  
    }  
}
```

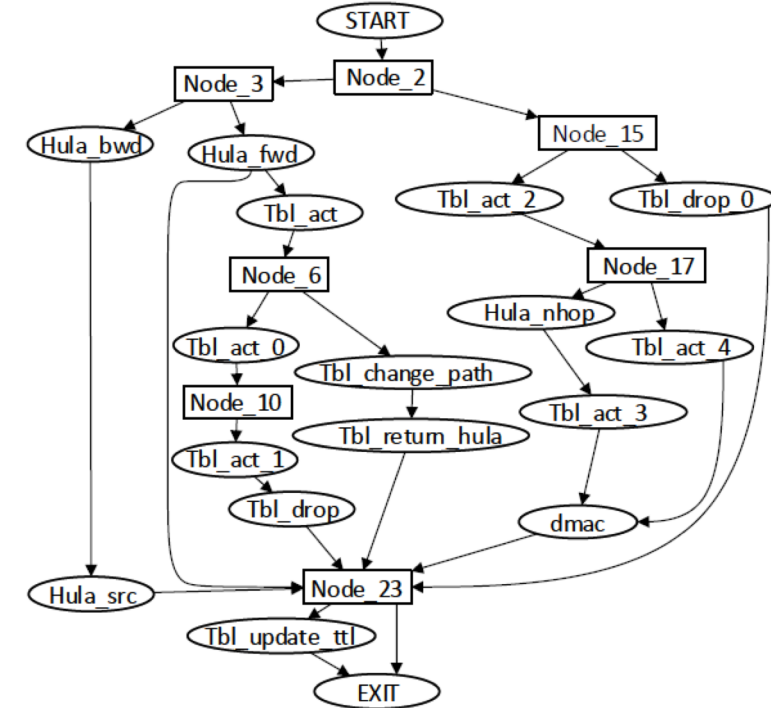
# Stateful CFG Construction

- From “Start”, recursively explore the child nodes of each node, until “Stop” is reached → Find a path
- Merge the common parts of all paths



# CFG Pruning

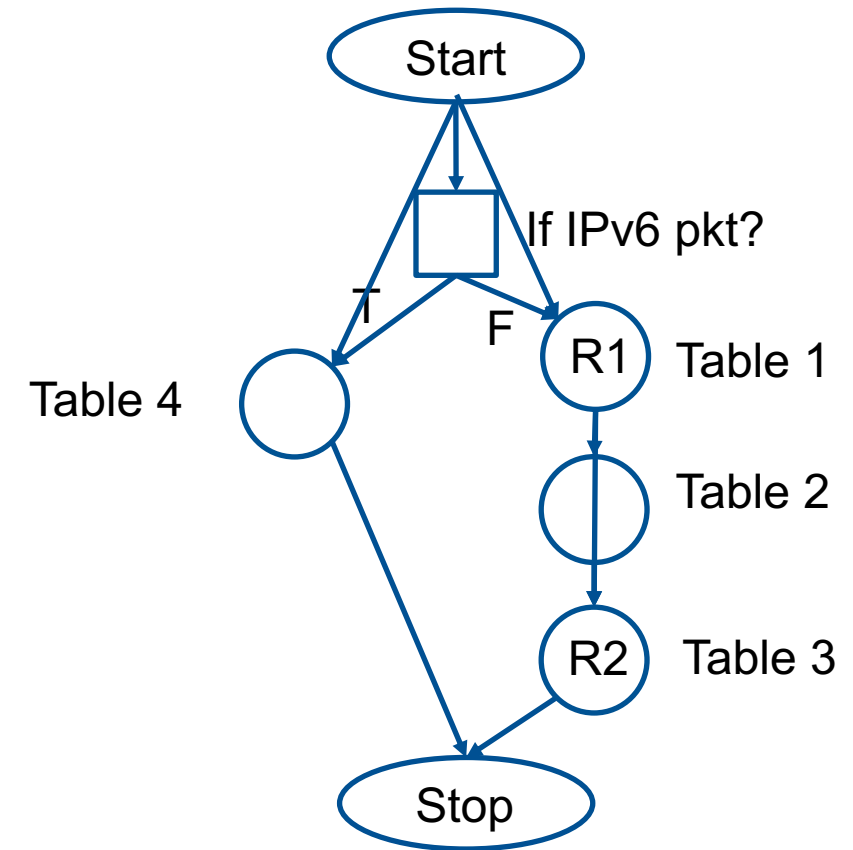
- Why pruning?
  - Original CFG is hard to digest
  - Stateless nodes in CFG have no effect
- Inspired by Thin Slicing [1]
- Two steps
  - 1. Remove all nodes without state access
  - 2. Merge consecutive tables with same state access on a single path



[1] Sridharan, Manu, Stephen J. Fink, and Rastislav Bodik. "Thin slicing." ACM SIGPLAN Notices. Vol. 42. No. 6. ACM, 2007.

# CFG Pruning

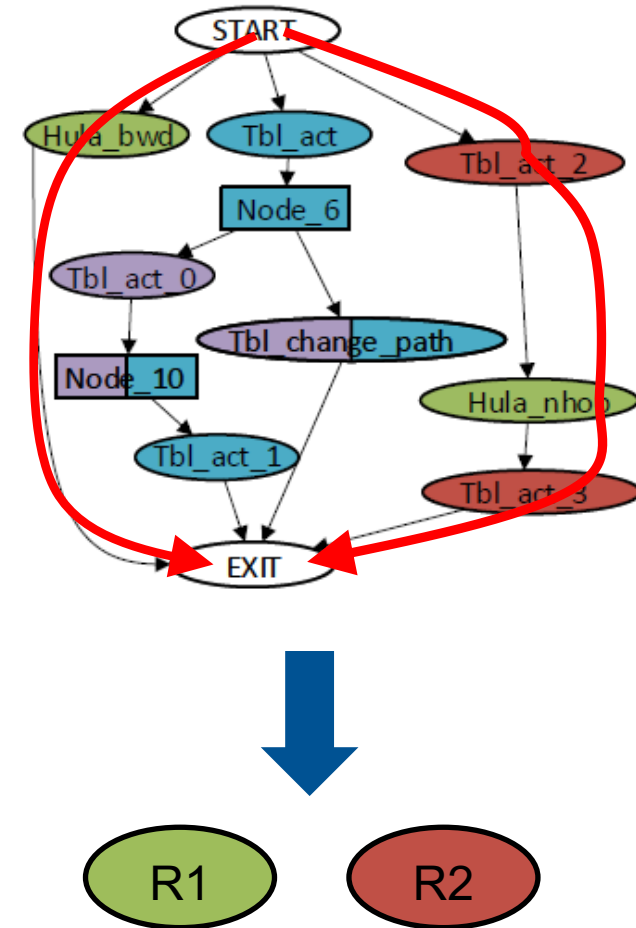
- Why pruning?
  - Original CFG is hard to digest
  - Stateless nodes in CFG have no effect
- Inspired by Thin Slicing [1]
- Two steps
  - 1. Remove all nodes without state access
  - 2. Merge consecutive tables with same state access on a single path



[1] Sridharan, Manu, Stephen J. Fink, and Rastislav Bodik. "Thin slicing." ACM SIGPLAN Notices. Vol. 42. No. 6. ACM, 2007.

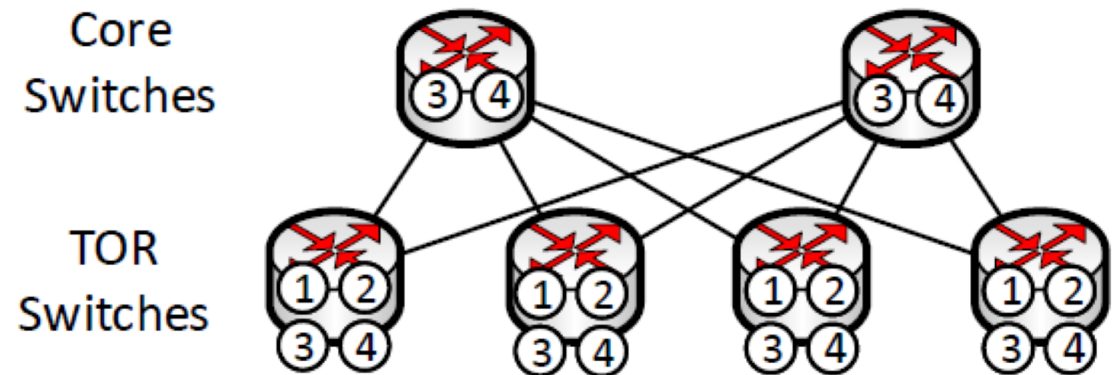
# Path & Role Identification

- Why identification?
  - Single P4 program consists of multiple functions
  - Different functions enabled on different P4 switches
- Input
  - Pruned CFG
  - Table entries during data plane initialization
  - Domain knowledge
  - (Human intervention)
- Output
  - States that need to be migrated for a certain P4 switch
  - (Migration schedule)



# Evaluation on HULA

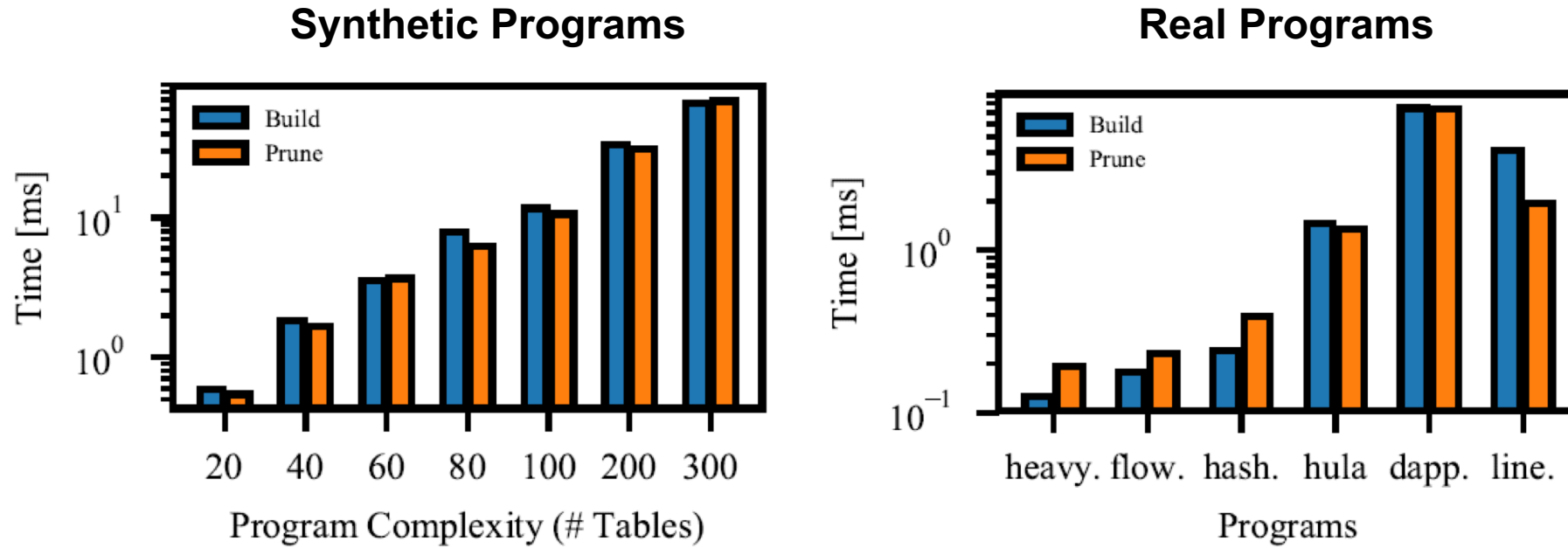
- Traffic engineering in data center networks [1]
- Probing
  - Maintain the current best paths between TOR switches considering queue length
- Forwarding
  - Send traffic on the best paths
- Probing: R3, R4
- Forwarding: R1, R2
- TOR switches: Probing + Forwarding
- Core switches: Forwarding



[1] Katta N, et al. Hula: Scalable load balancing using programmable data planes[C]//Proceedings of the Symposium on SDN Research. ACM, 2016: 10.



# General Performance: Algorithm Runtime

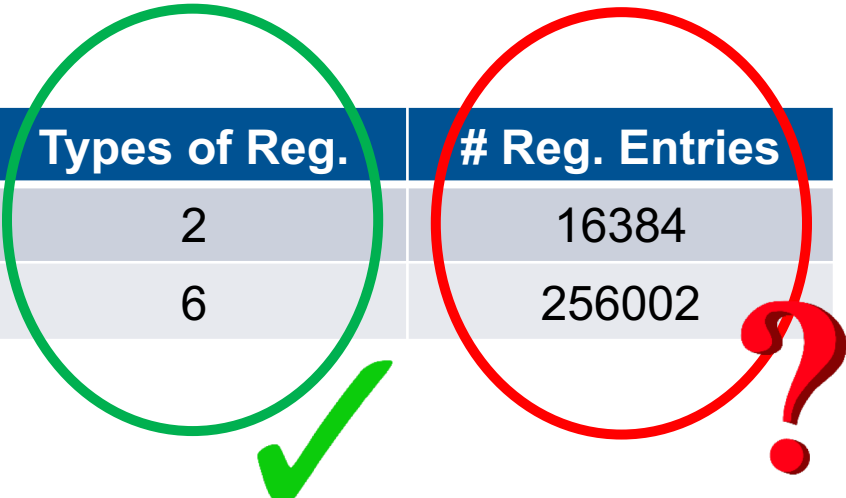


- Comparable CFG construction time and CFG pruning time
- Synthetic: scales with # tables, less than 100 ms
- Real: depends on pipeline structure (LinearRoad has higher LoC, but Dapper has more branches)

## Conclusion and Future Work

- P4State analyzer can **quickly** and **successfully** recognize the register types that need migration during data plane reconfiguration
  - Avoid unnecessary register migration
  - Output CFG with states that allows human intervention
- Next steps
  - Only migrate valid register entries
  - Scheduling of register migrations

| Program  | LoC | Types of Reg. | # Reg. Entries |
|----------|-----|---------------|----------------|
| Flowlet  | 203 | 2             | 16384          |
| netpaxos | 210 | 6             | 256002         |





Thank you! &  
Questions?