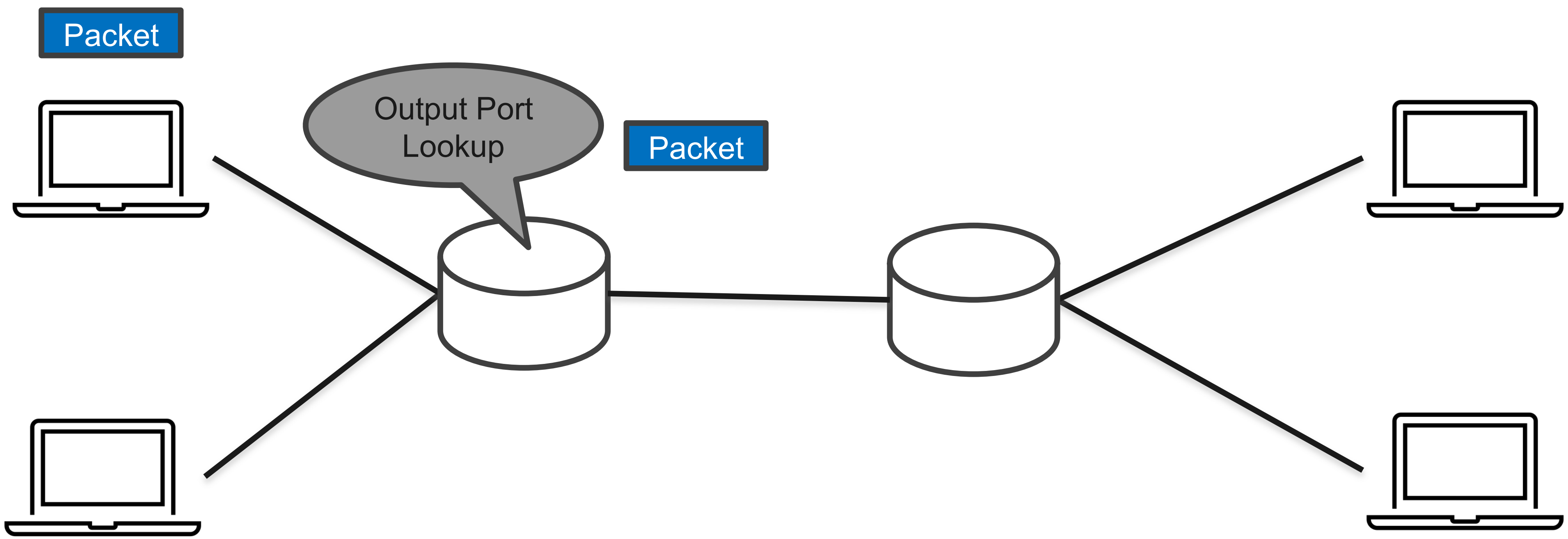
Automatic Path Verification

Sundararajan Renganathan, Bruce Spang, Nate Foster, Nick McKeown

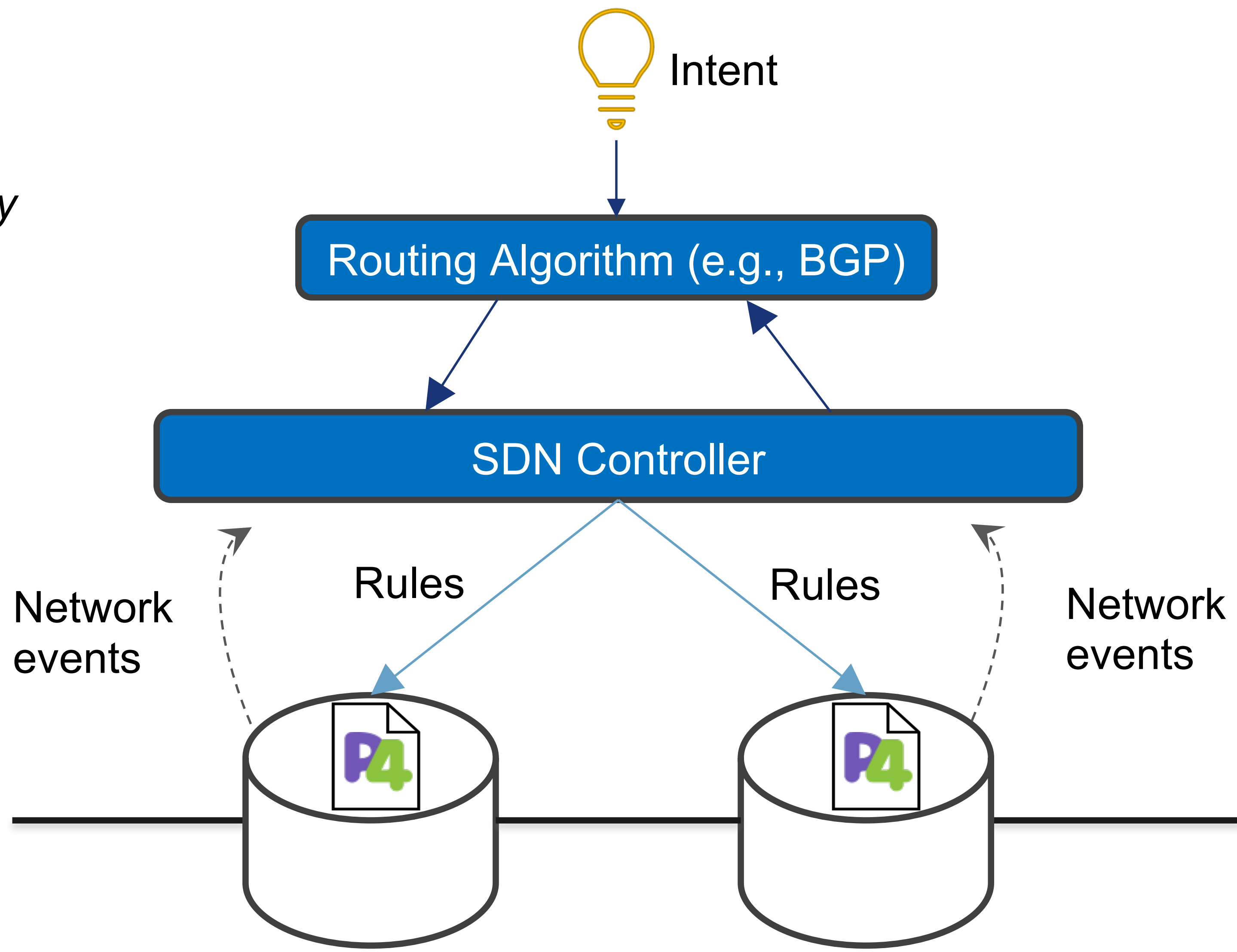
What does a network do?



Deeply programmable networks

Top-to-bottom programmability

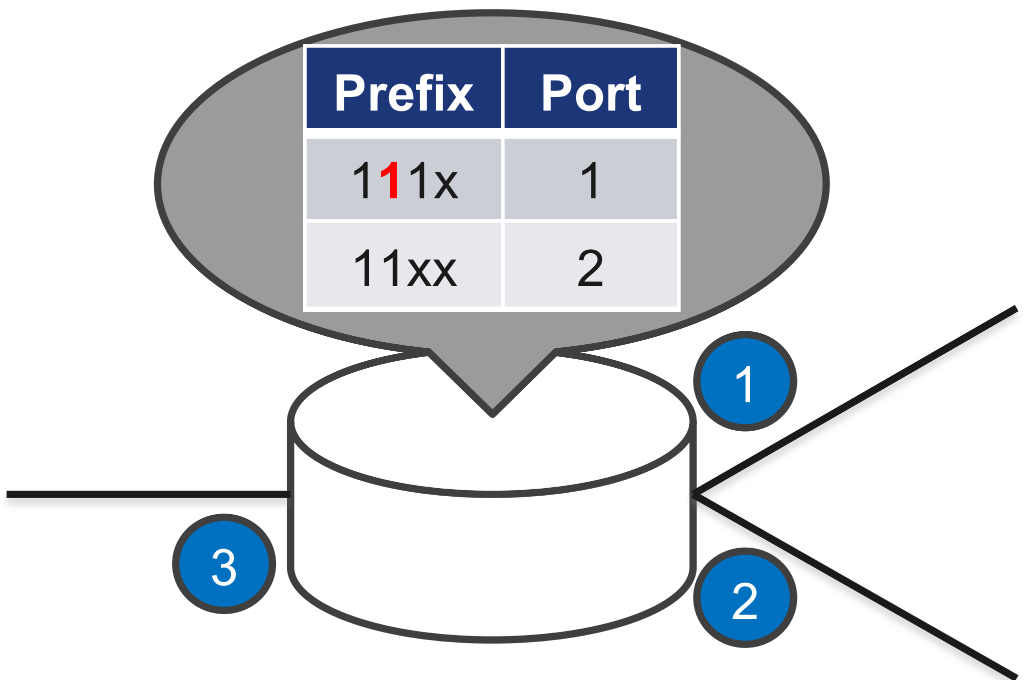
End-to-end programmability



Can packets deviate from intended behavior?

Secondary particle (cosmic ray)

Bit flip!



Prefix	Port
111x	1
11xx	2

Packets with address 111x shouldn't be sent here ☹

Bit flips in ternary memory are hard to fix:

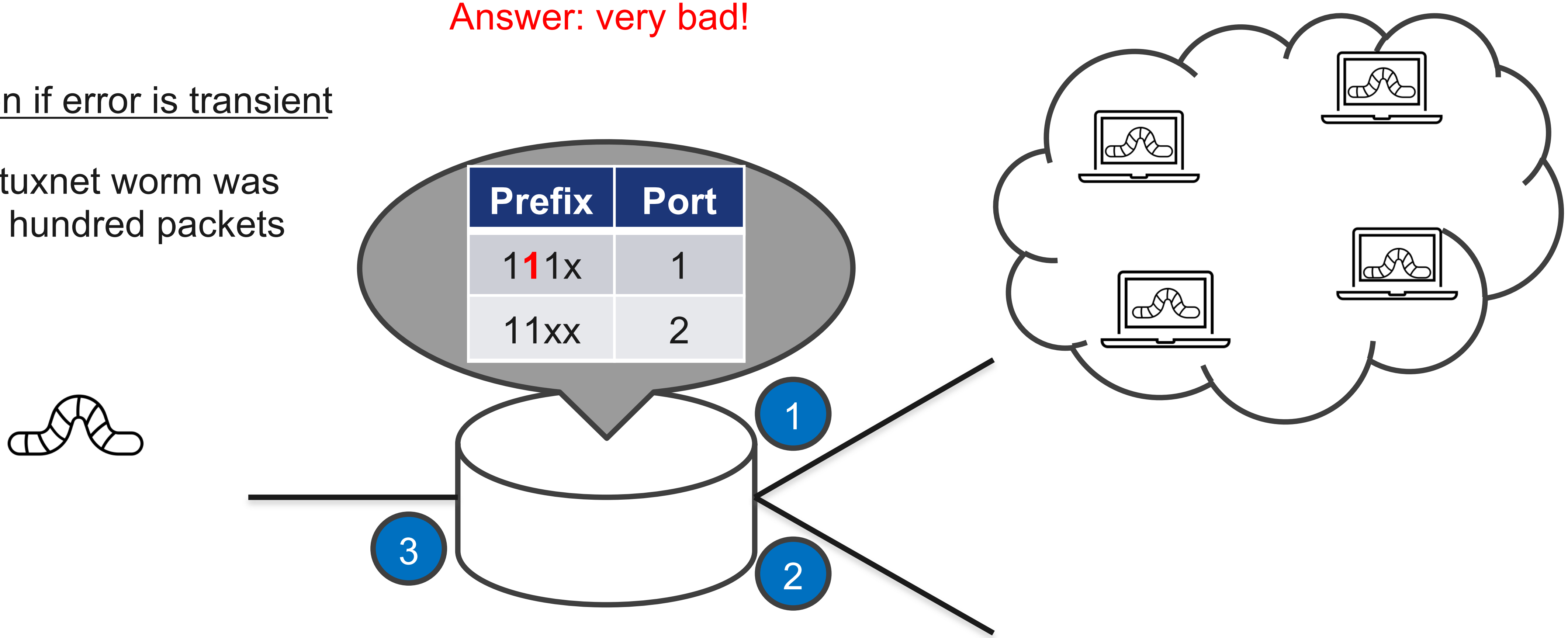
- Cannot use ECC's
- Instead, switches periodically probe tables to check bit flips

How bad can this be?

Answer: very bad!

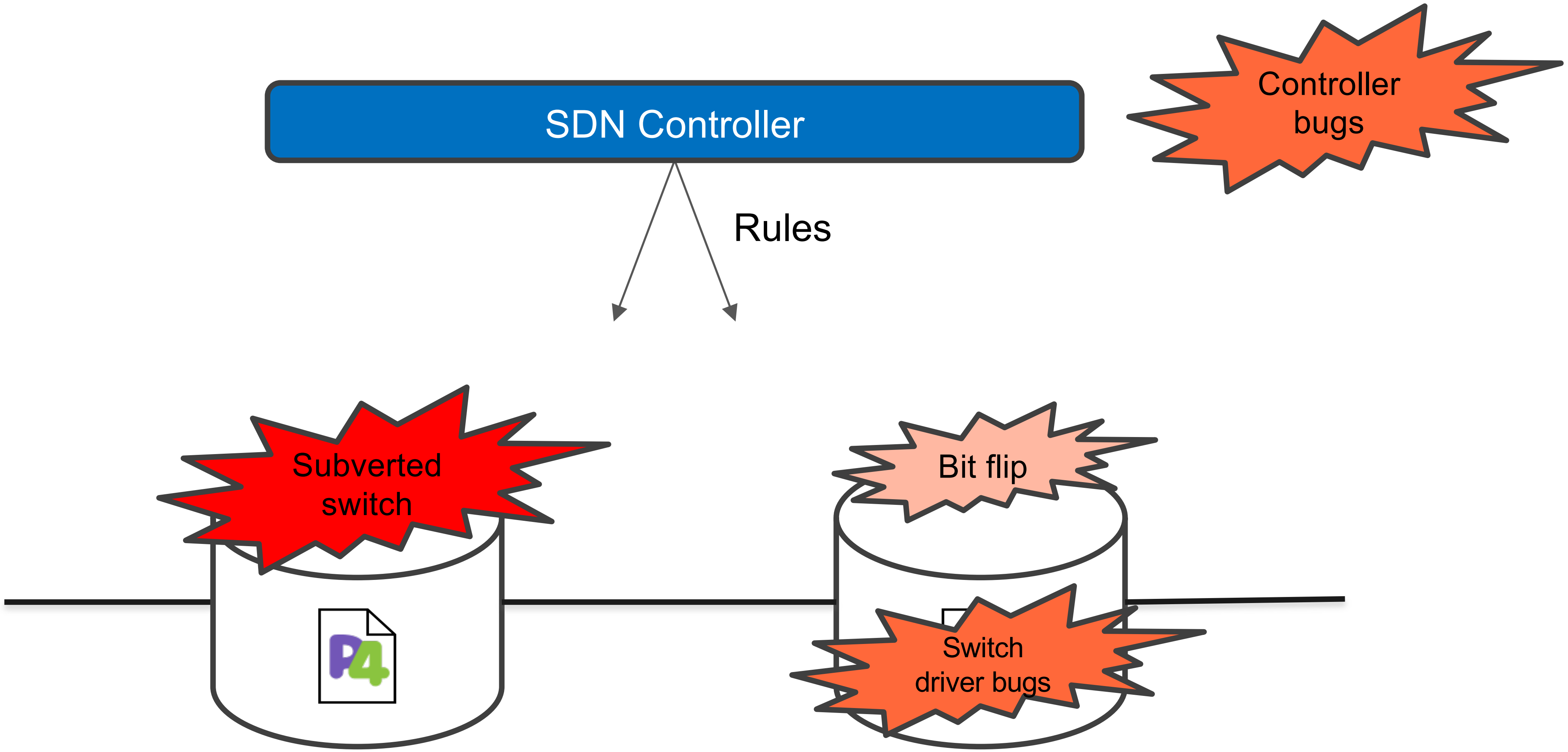
<u>Problematic even if error is transient</u>

Payload of Stuxnet worm was **500KB** ~ few hundred packets



Prefix	Port
111x	1
11xx	2

Other violations of intended behavior



Tools to verify networks

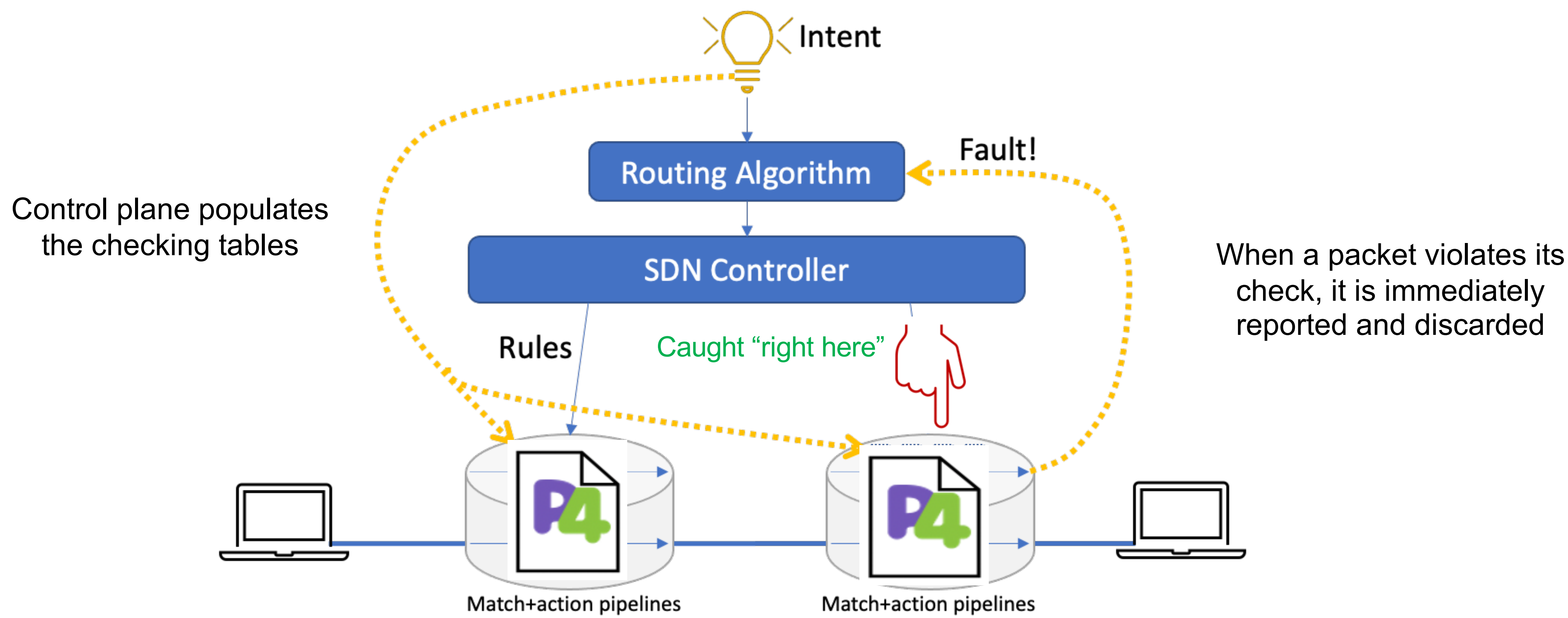
- Ping & Traceroute
  - Rudimentary
  - Probes don't exercise the same paths across probes
- Static Analysis tools (e.g., HSA, Veriflow, etc.)
  - Work against a mathematical of the network forwarding behavior
  - **Cannot verify behavior outside model's assumptions** => e.g., bugs in the switch hardware or switch driver
- Runtime monitoring tools (e.g., INT, postcards, etc.)
  - For every packet, a short summary is sent to a centralized server which validates paths
  - **Not scalable** => even small networks need big beefy servers to validate every packet
  - **Not real-time** => cannot prevent packets from taking incorrect paths

Problem statement

Can a network of switches verify that <u>every packet</u> follows the correct path, by checking, <u>in real time</u>, against a network-wide model of the intended behavior?

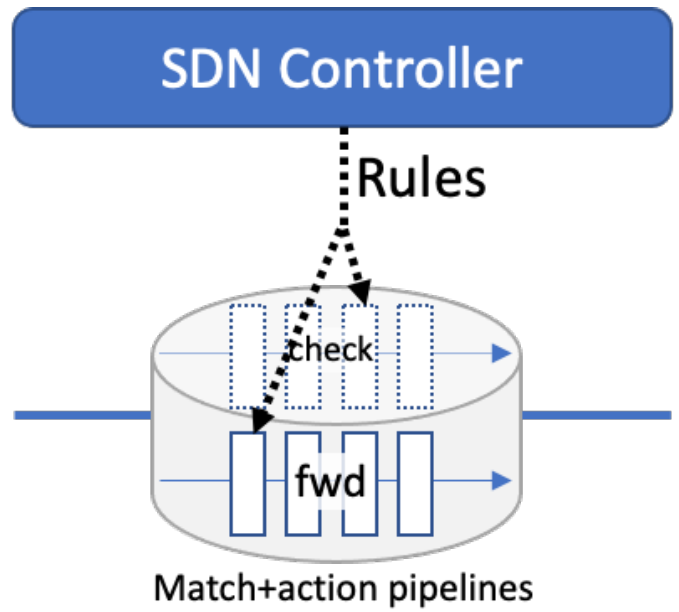
Leverage programmable data planes

- **Idea**: perform the path checks on the switches themselves!
  - Add "checking" match-action tables alongside the usual forwarding tables



Warmup approach

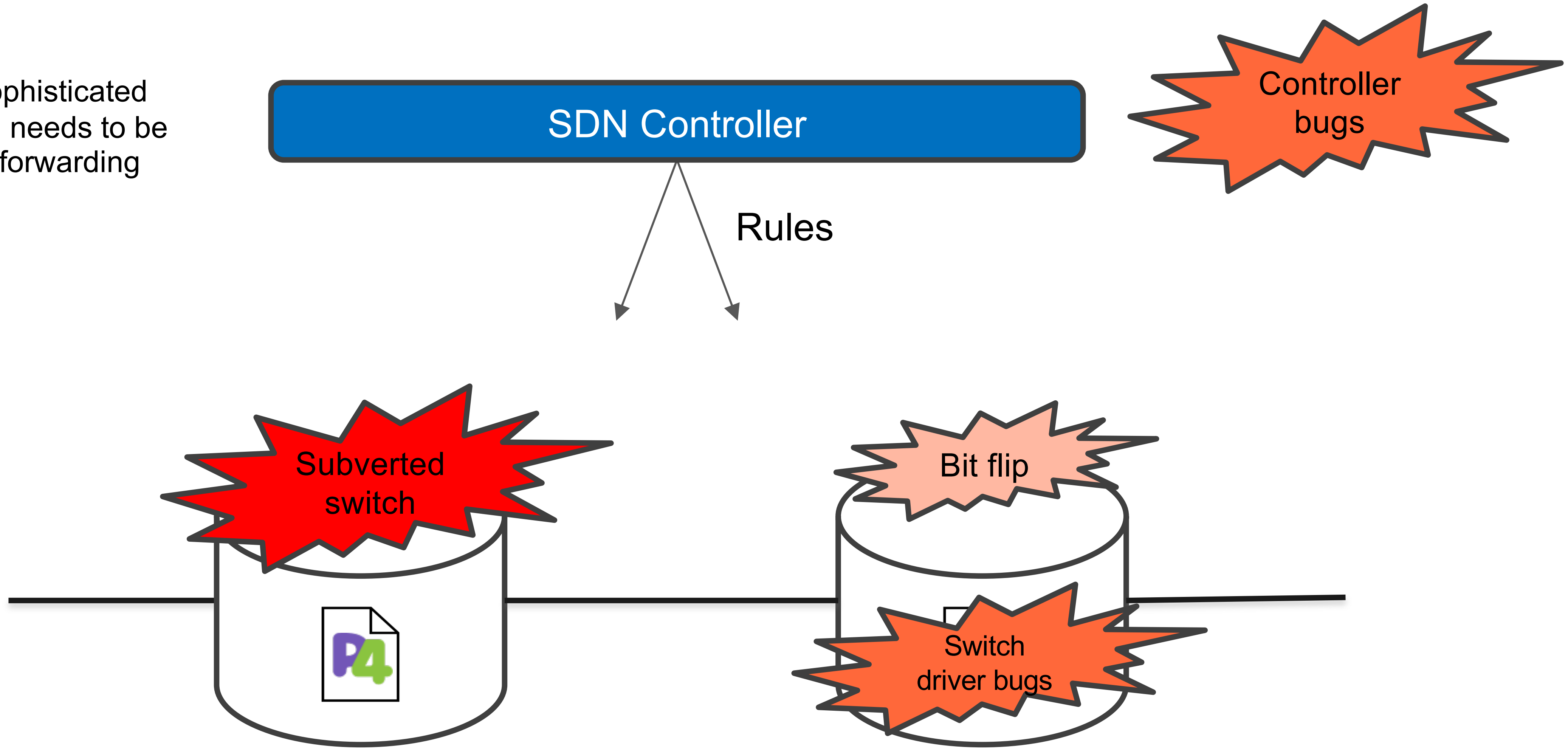
- **Silly idea:**
  - The checking pipeline is a duplicate of the forwarding pipeline



  - *Checking algorithm*: for every packet, run it through both pipelines and see if the outputs match

Warmup approach (bugs detected)

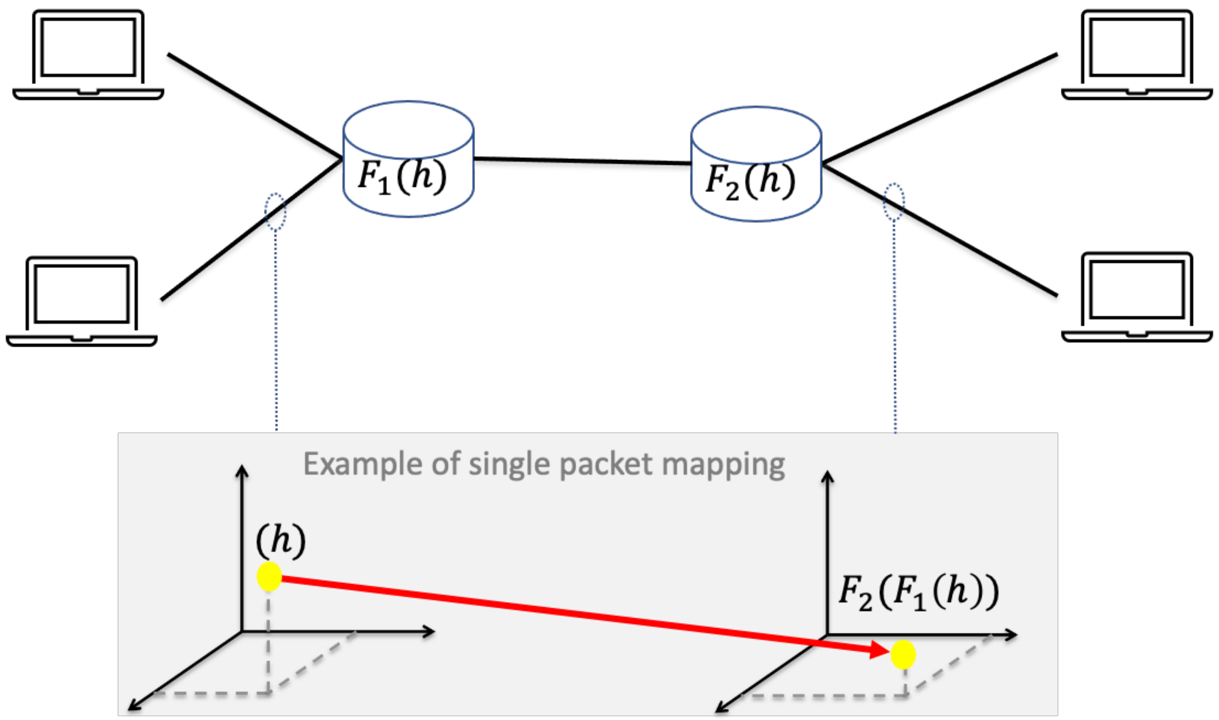
To catch more sophisticated bugs, the checking needs to be **independent** of forwarding



Approach 1: self-validation

- First, we need a model of the intended behavior
  - We use Header Space Analysis (HSA) to construct the model

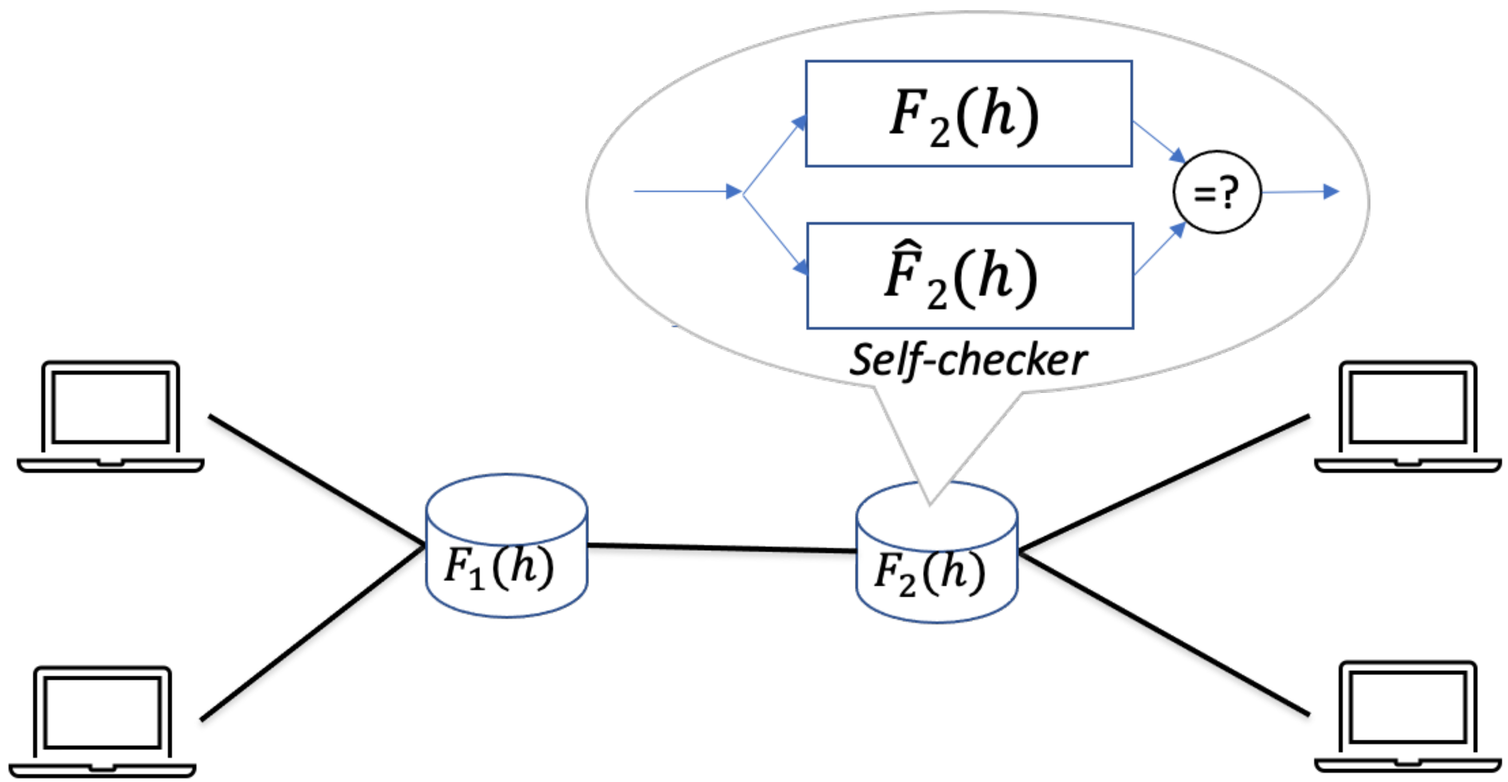
The industry is adopting HSA through the commercial offerings by **Forward Networks**



Each switch is represented as a transfer function F_i

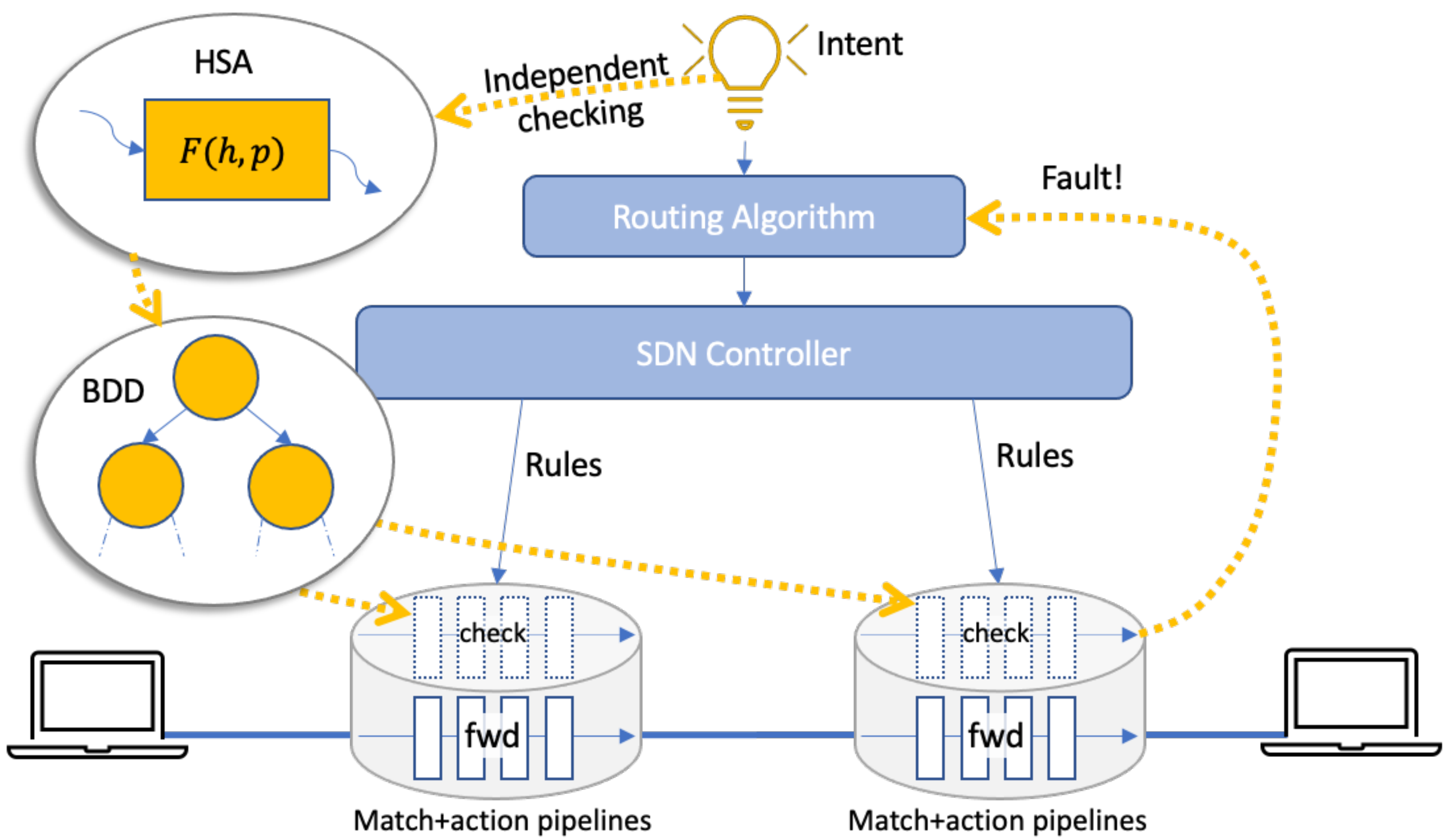
HSA model for self-validation

- Each switch's transfer function is augmented with an independent checking function



HSA model to checking tables

- Convert the HSA model into a Binary Decision Diagrams (BDDs) and turn these into table entries

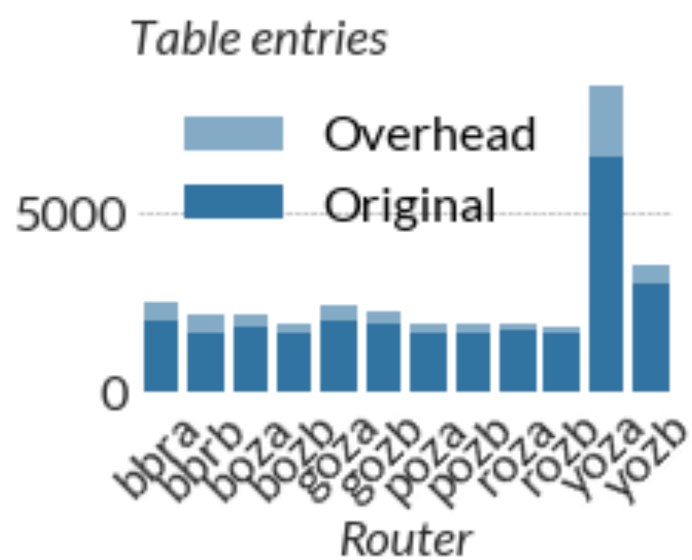


Approach 1: self-validation

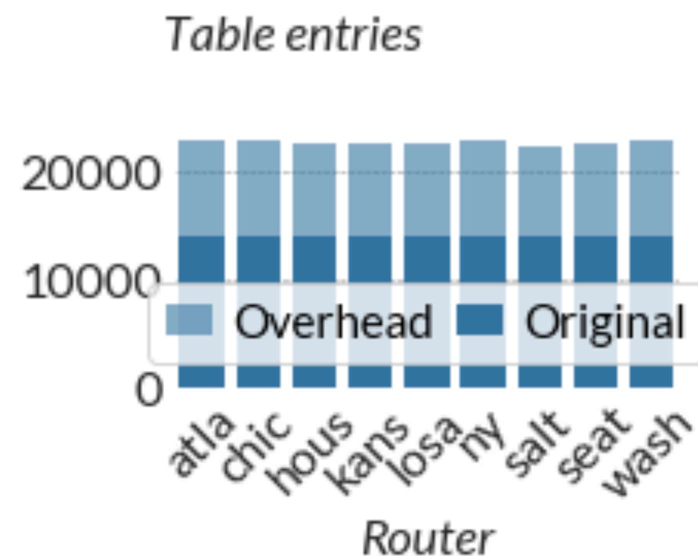
- **Checking overhead**
  - Expressed in terms of the additional table entries needed for checking

Worst-case overhead

Network type	Self checks
Stanford	15.4%
Internet2	61.4%
Azure	100%

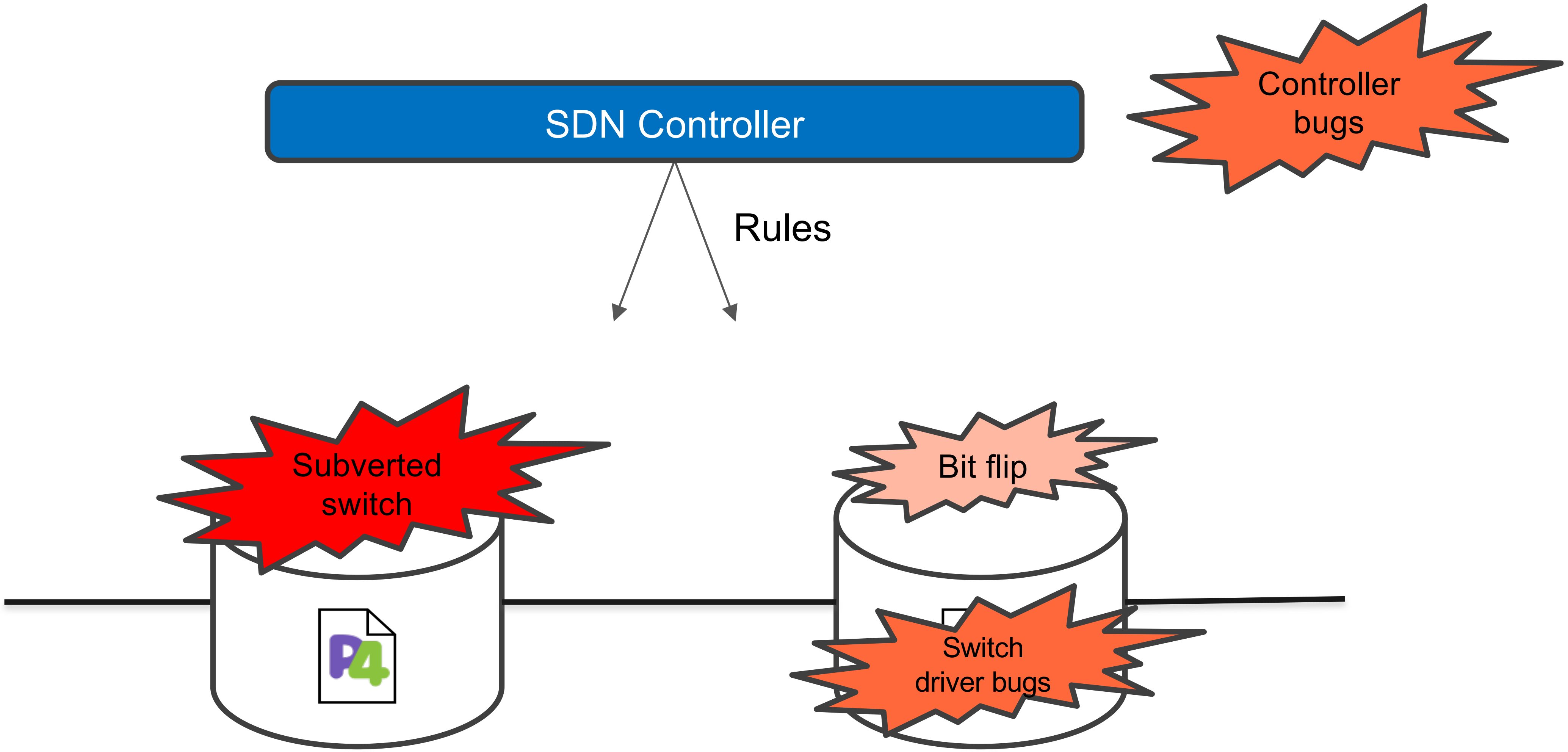


Stanford overheads



Internet2 overheads

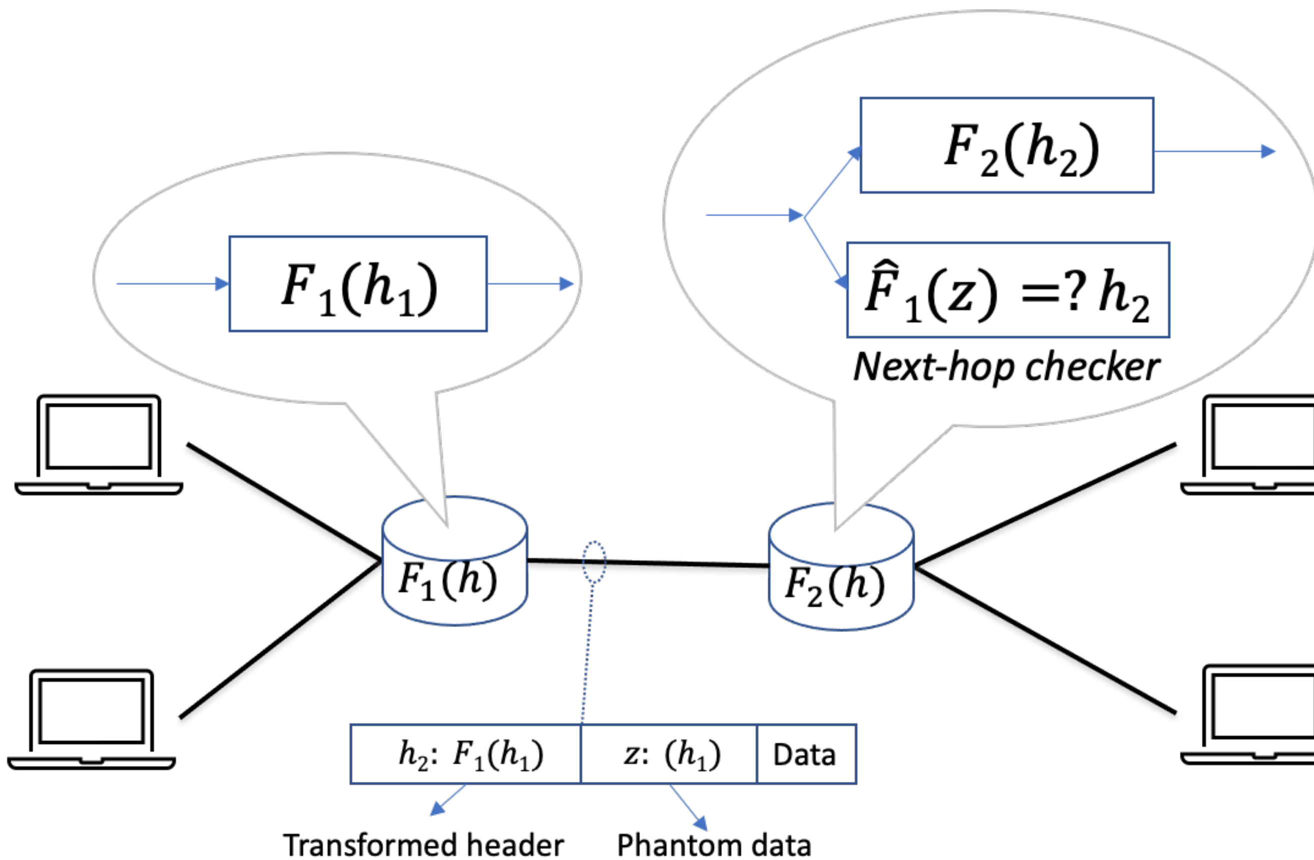
Approach 1: self-validation (bugs detected)



Approach 2: neighbor validation

- **We can move the checks around!**
  - Each switch is checked by its neighbors

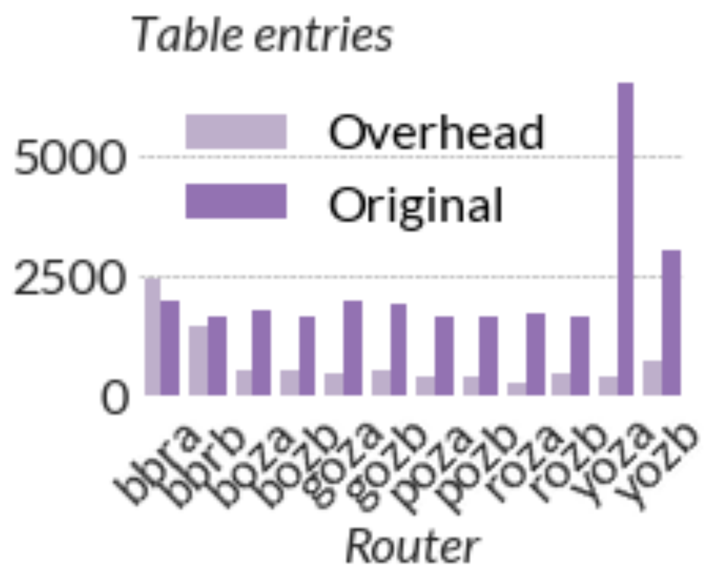
Augmented HSA model for neighbor validation



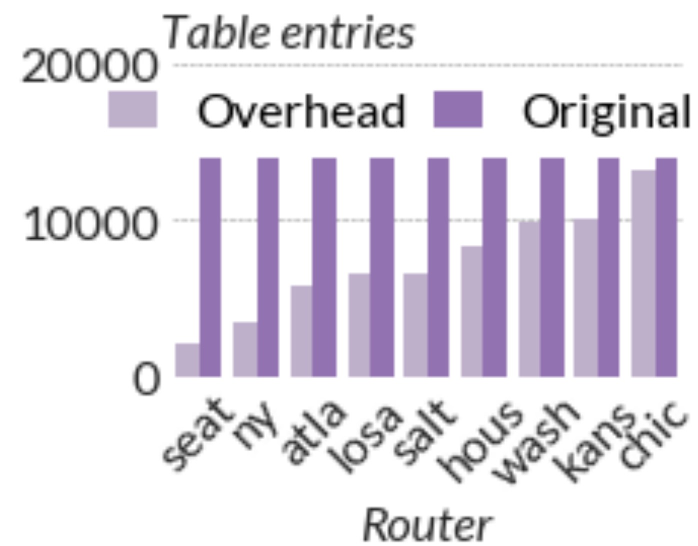
Approach 2: Neighbor validation overhead

Worst-case overhead

Network type	Neighbor checks
Stanford	25.1%
Internet2	94.4%
Azure	180%



Stanford overheads



Internet2 overheads

Neighbor validation doesn't appear expensive! ☺

Next steps

- Incorporate the checking of stateful properties
- Integrate with ONF’s Aether platform