SIGCOMM '12, PLDI '15

Efficient Synthesis of Network Updates

Nate Foster
Cornell University

Jedidiah McClurgh

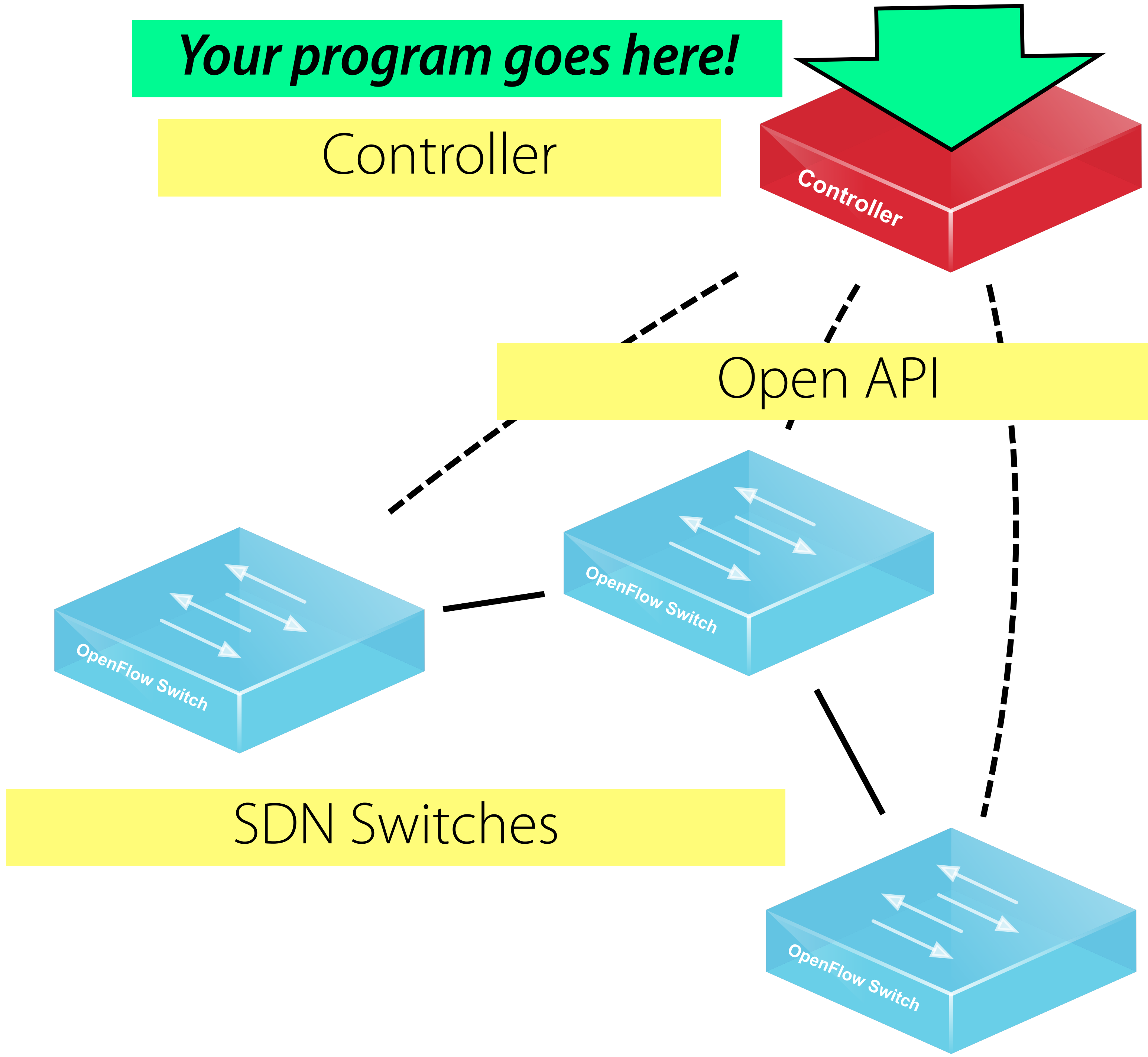
Hossein Hojjat

Pavol Cerny

Todd Warszawski

Andrew Noyes

Software-Defined Networking

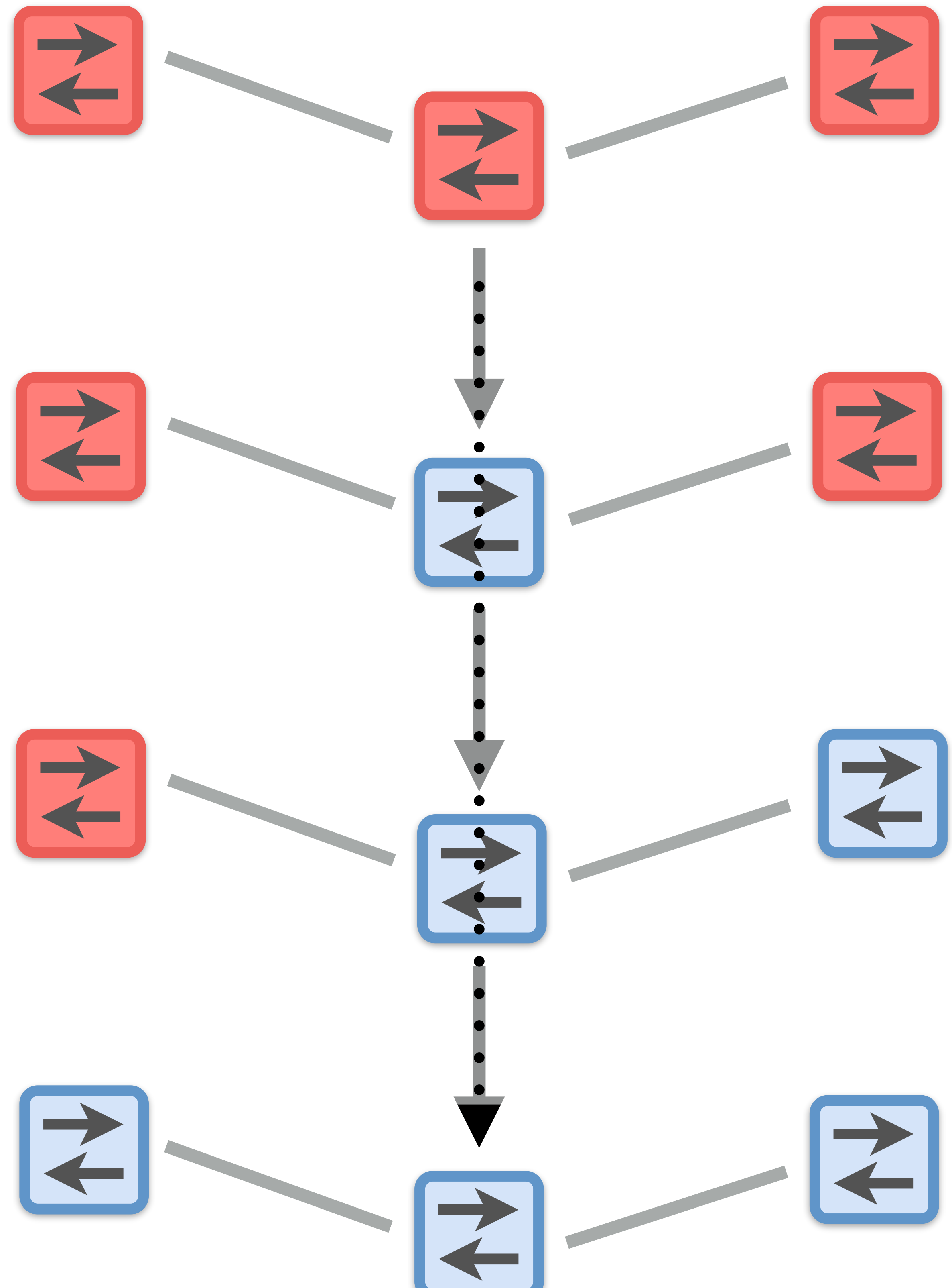


Network Updates

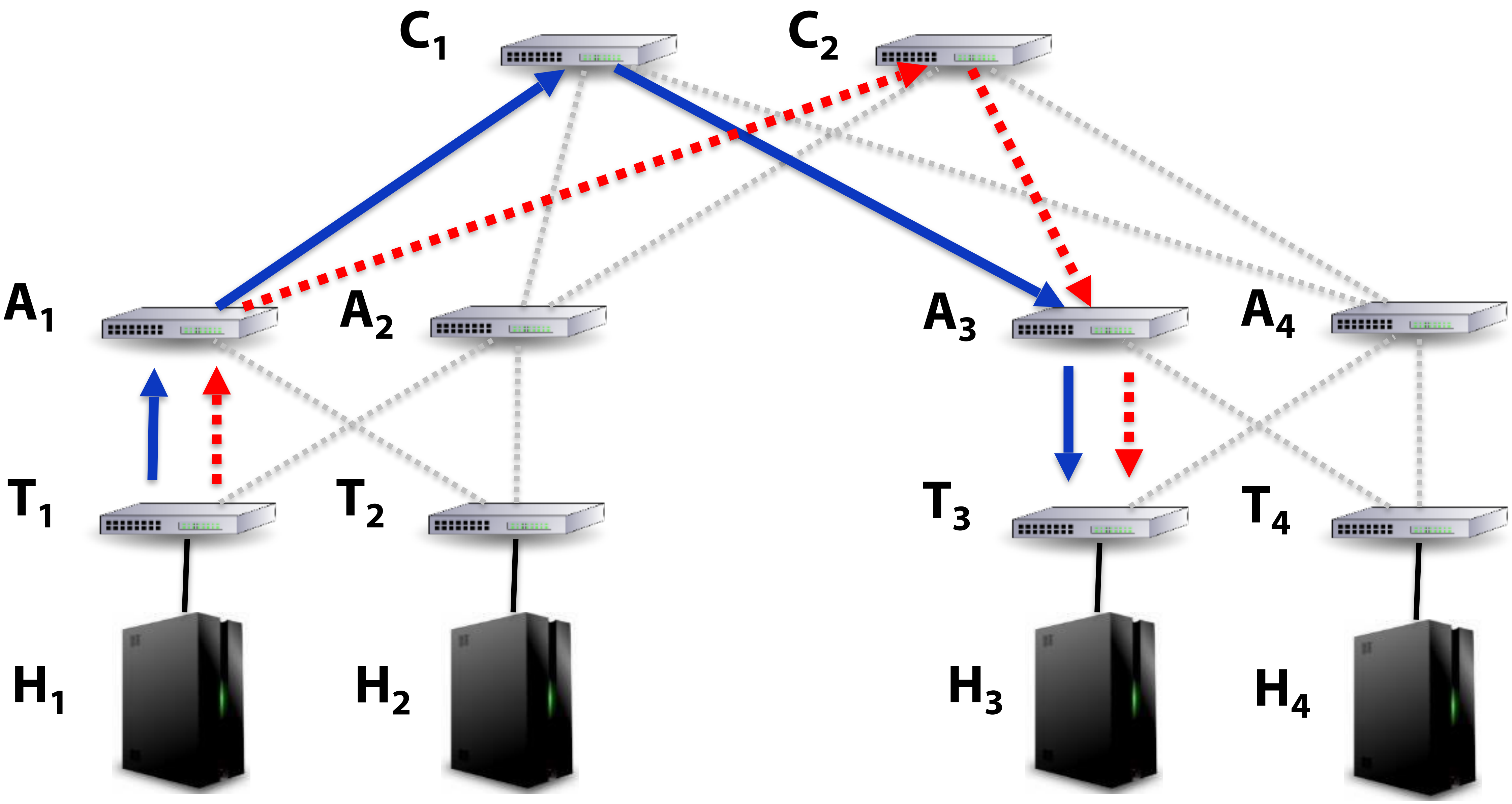
How to transition from one network-wide configuration to another?

It requires stepping through multiple intermediate configurations in general…

…hard to guarantee that important network-wide properties will be preserved

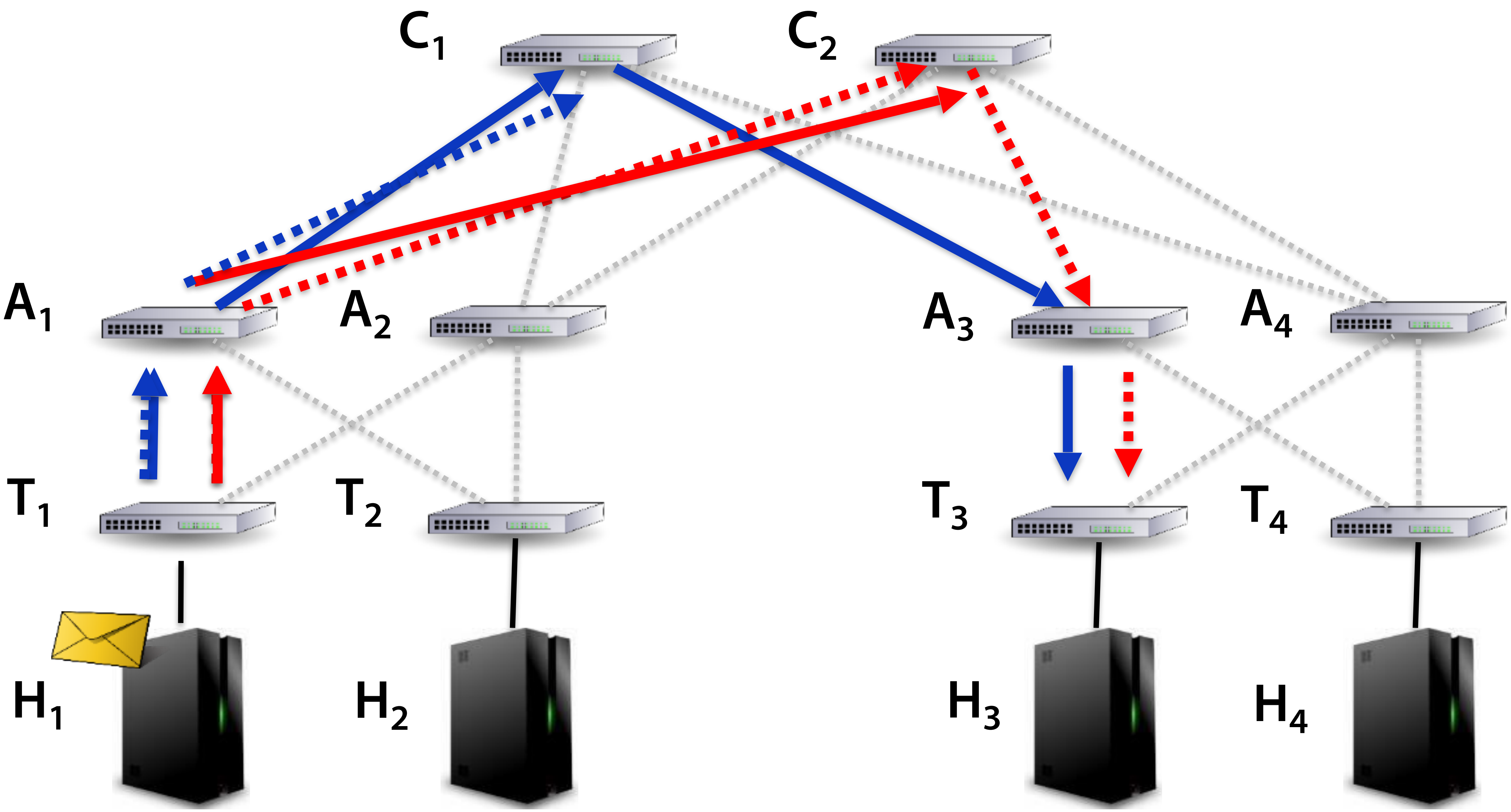


Example: Data Center

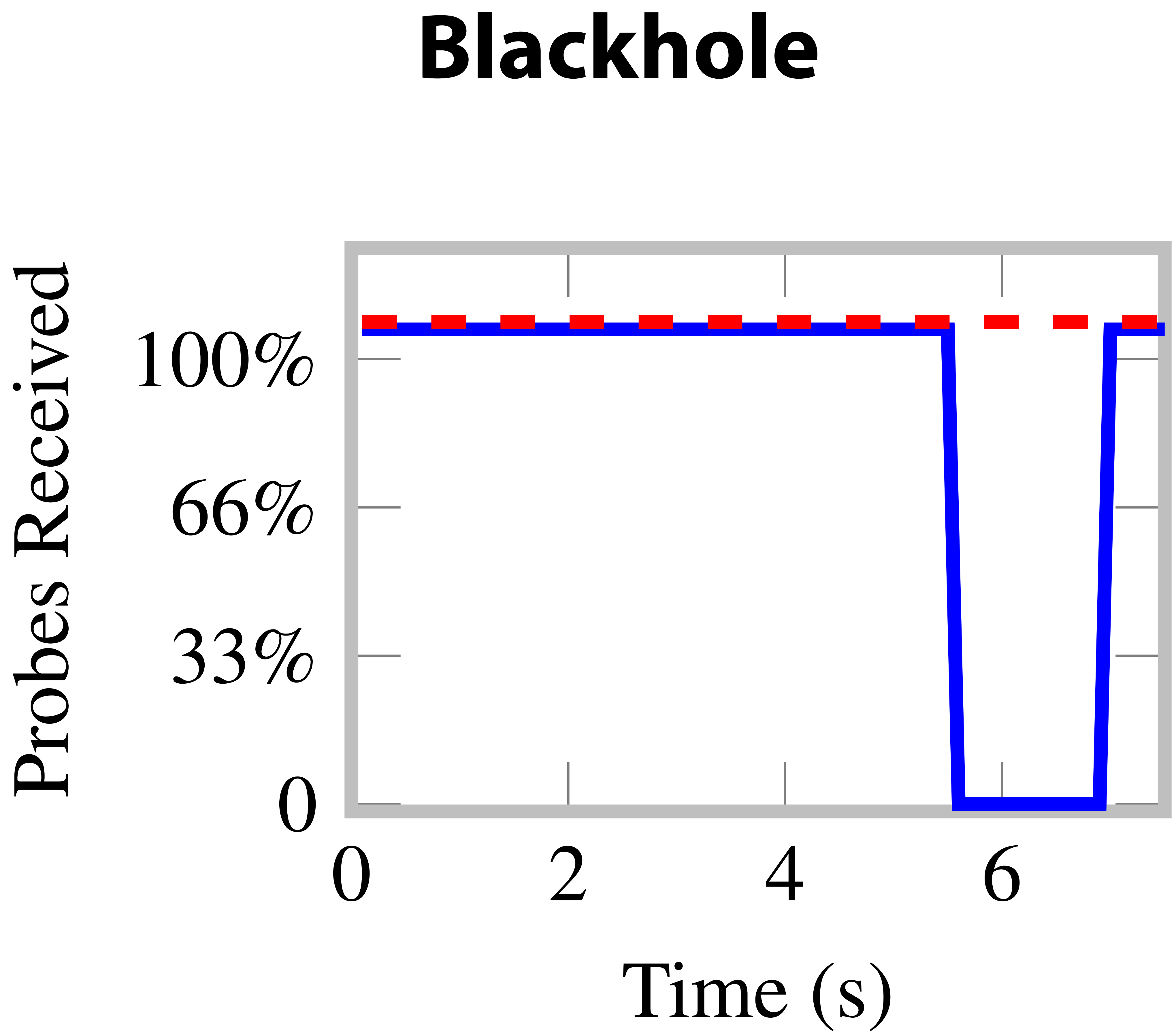


Update: `upd T1; upd A1; upd C2; upd A3`

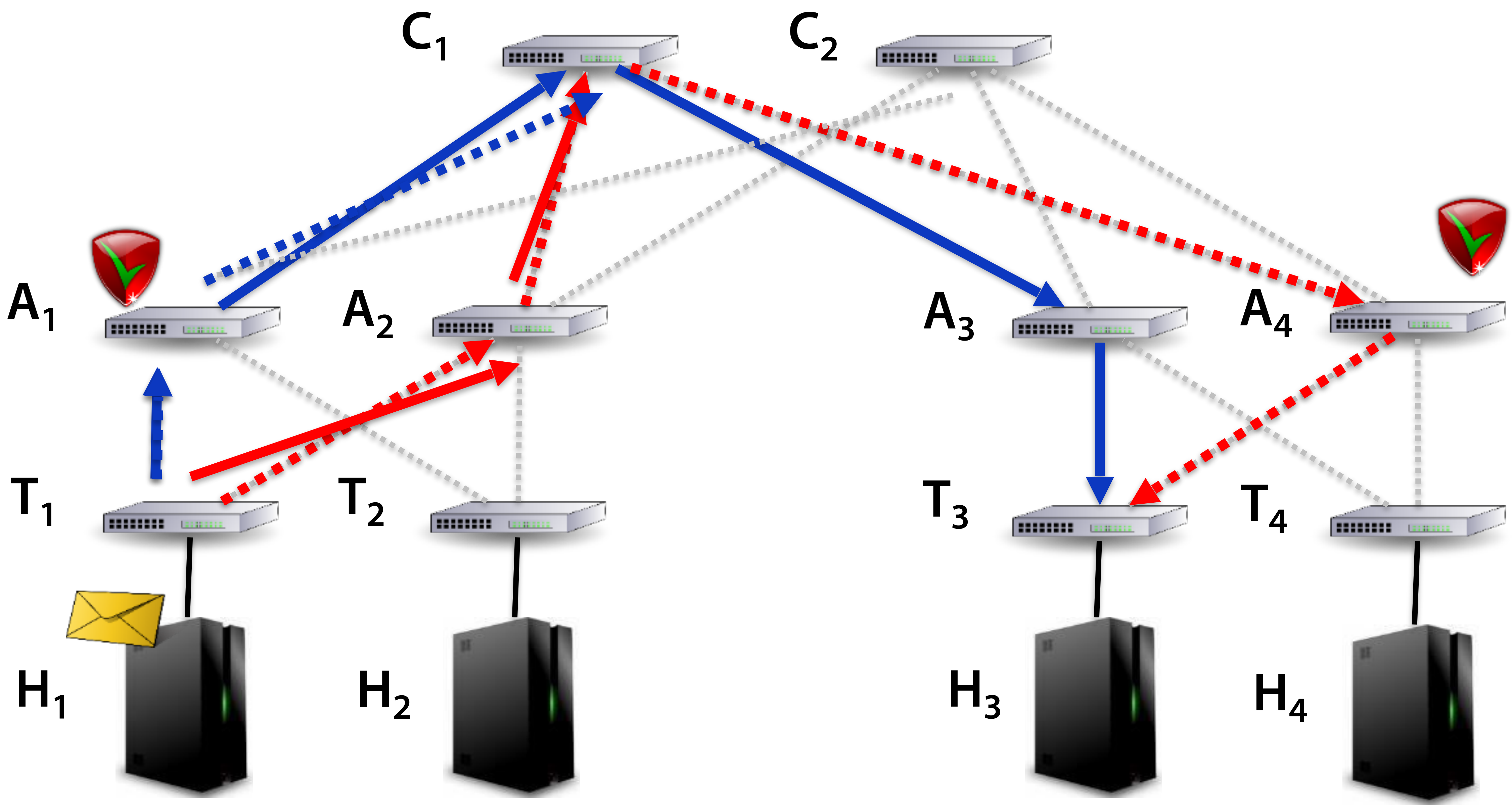
Naive Update



Problem: naive update creates a blackhole!


Blackhole
Probes Received
100%
66%
33%
0
0
2
4
6
Time (s)

Naive Update



Problem: naive update leads to access control violation!

Is This Really a Problem?

At 12:47 AM PDT on April 21st, a network change was performed as part of our normal scaling activities...

During the change, one of the steps is to shift traffic off of one of the redundant routers...

The traffic shift was executed incorrectly and the traffic was routed onto the lower capacity redundant network.

This led to a “re-mirroring storm”...

During this re-mirroring storm, the volume of connection attempts was extremely high and nodes began to fail, resulting in more volumes left needing to re-mirror. This added more requests to the re-mirroring storm...

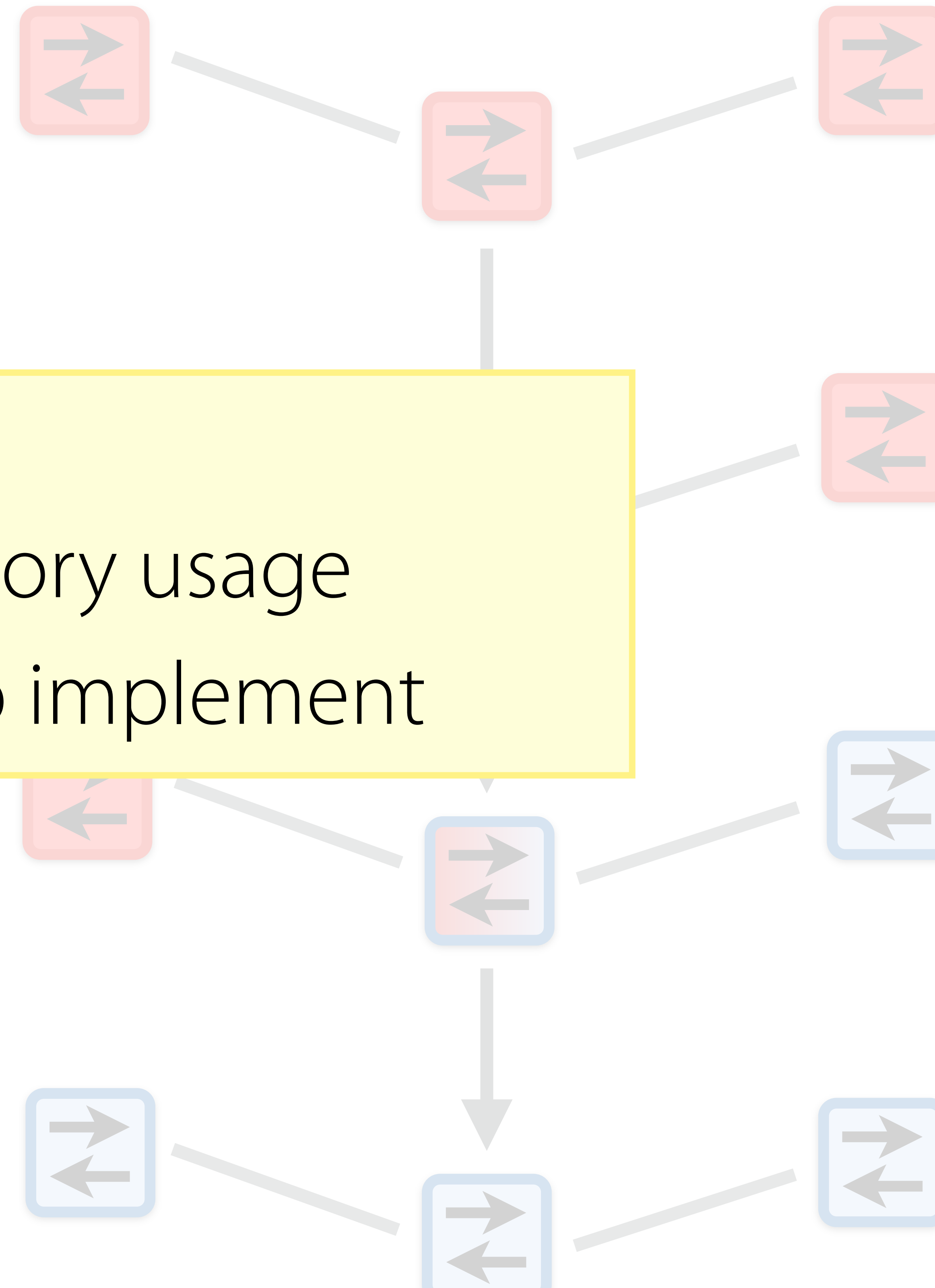
The trigger for this event was a **network configuration change**.

Per-Packet Consistent Updates

Guarantee: every packet (or flow) in the network "sees" a single policy version

Two-Phase

- Tag config
- Install new
- Install new
- Wait for in-flight packets to exit
- Delete old configurations

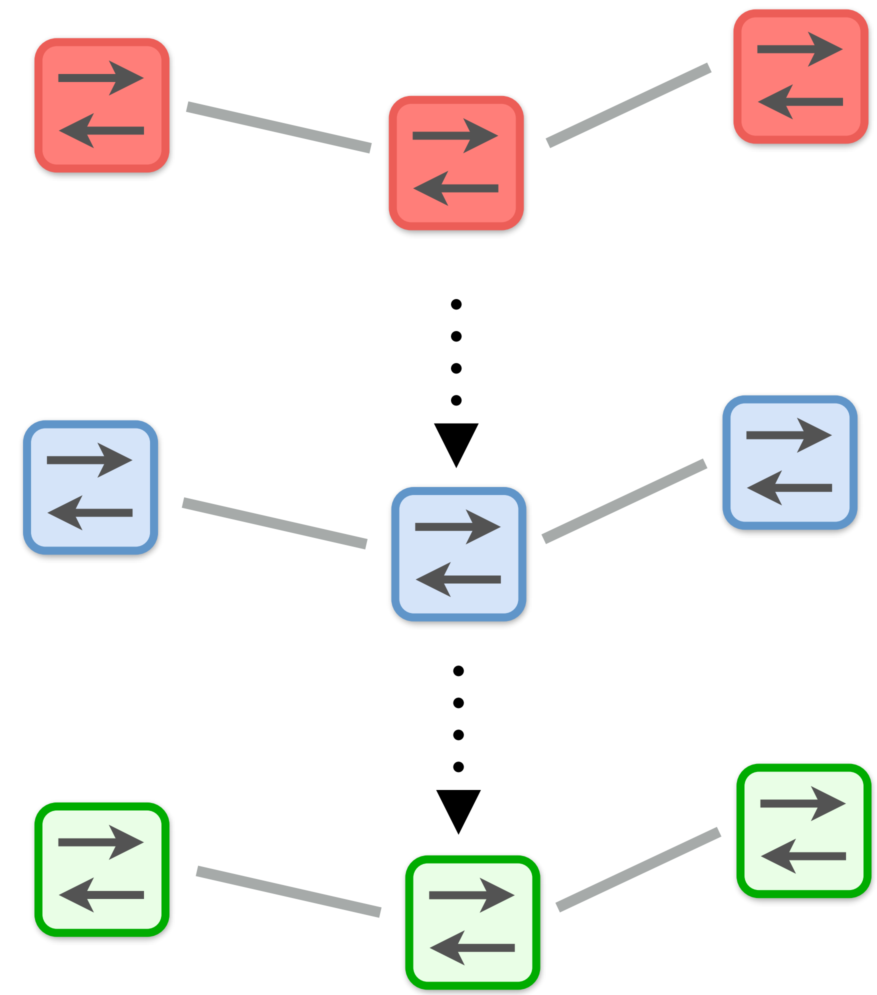


Limitations:

- Doubles peak memory usage
- Updates are slow to implement

Per-Packet Consistent Updates

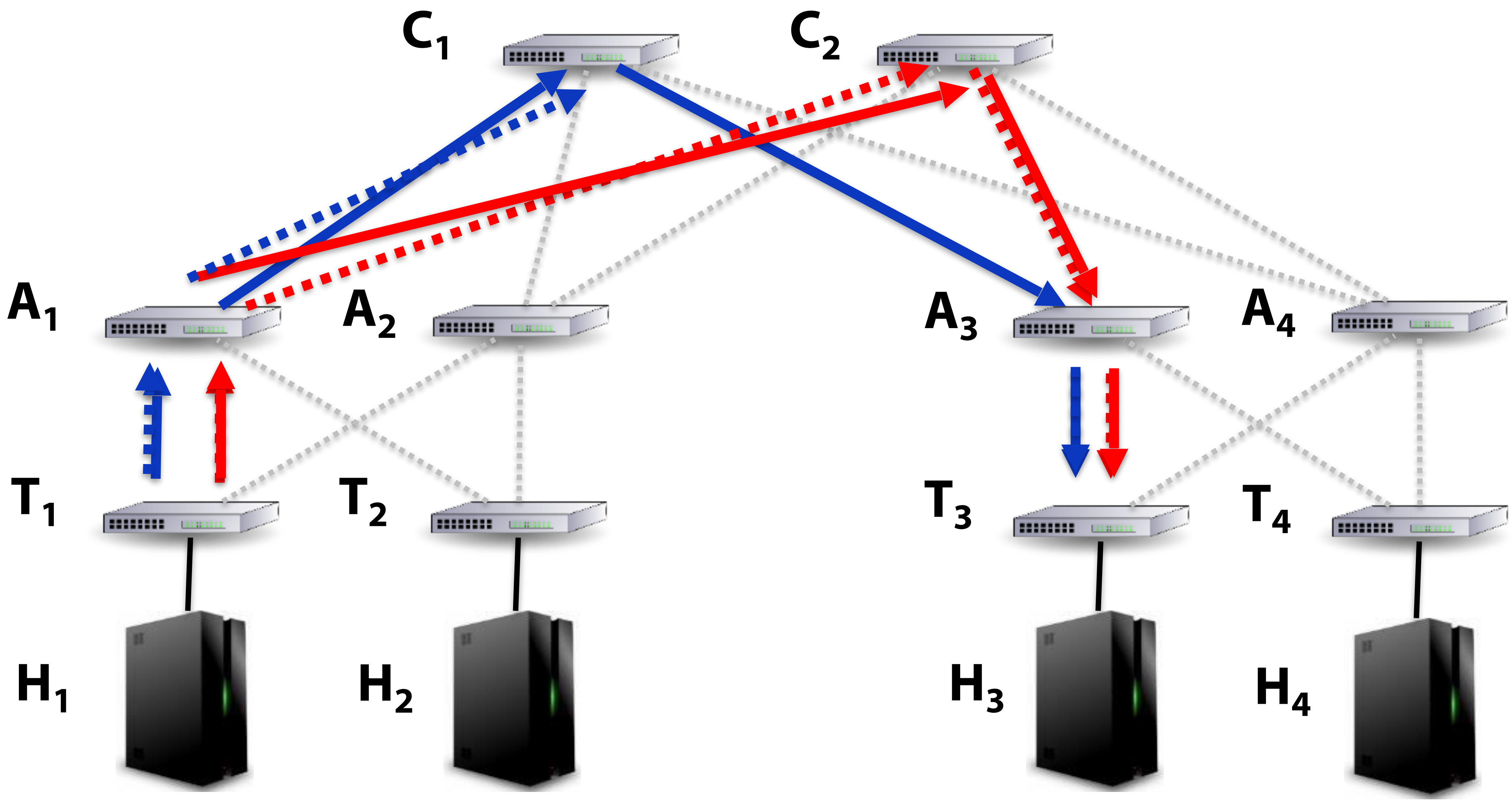
Theorem [SIGCOMM '12]: a network update is per-packet consistent if and only if it preserves all safety properties.



Questions:

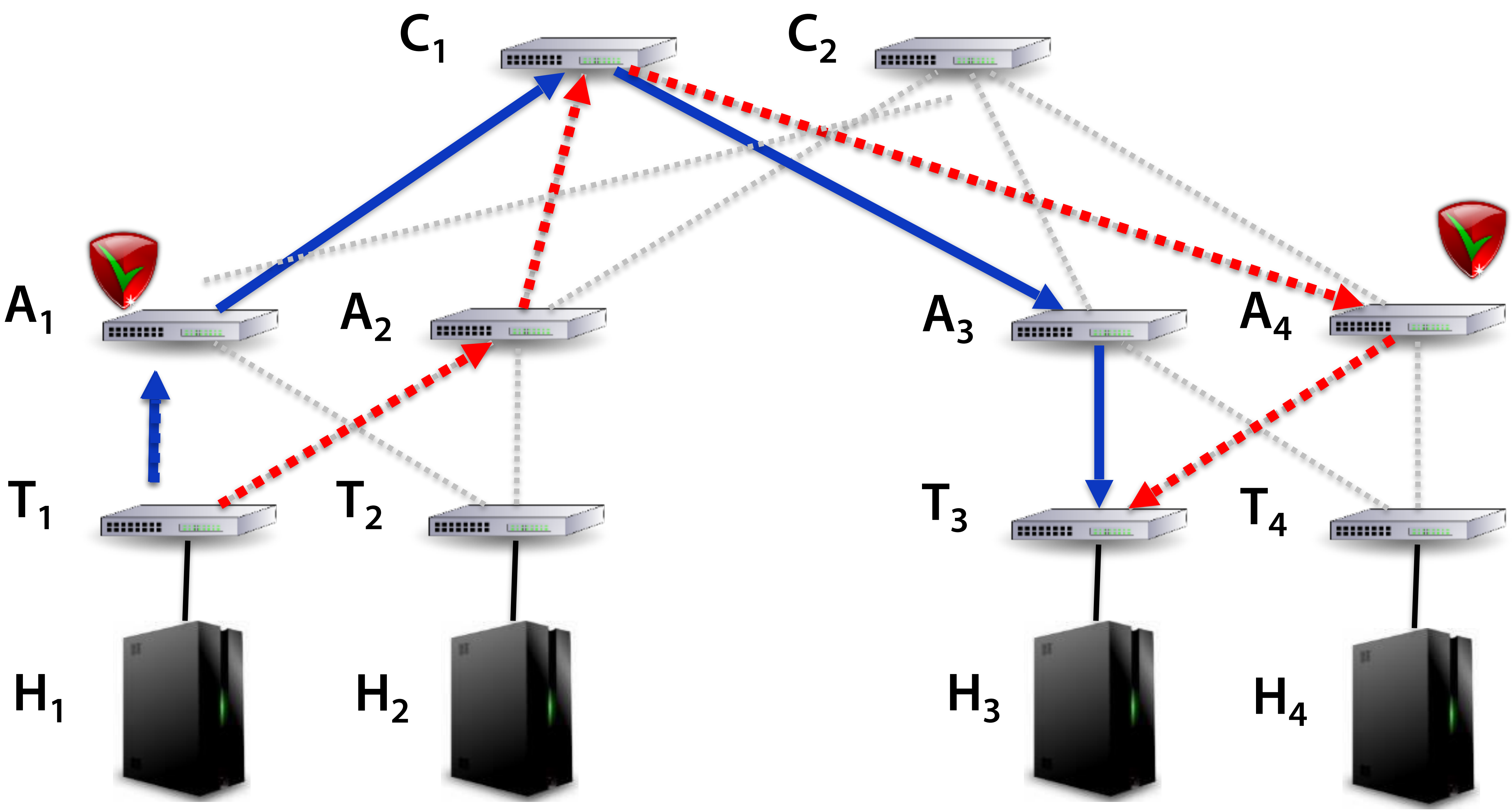
- Can we implement a per-packet consistent update by simply updating switches in the right order?
- If not, can we relax the requirements in a reasonable way to obtain efficient updates?

Example: Data Center



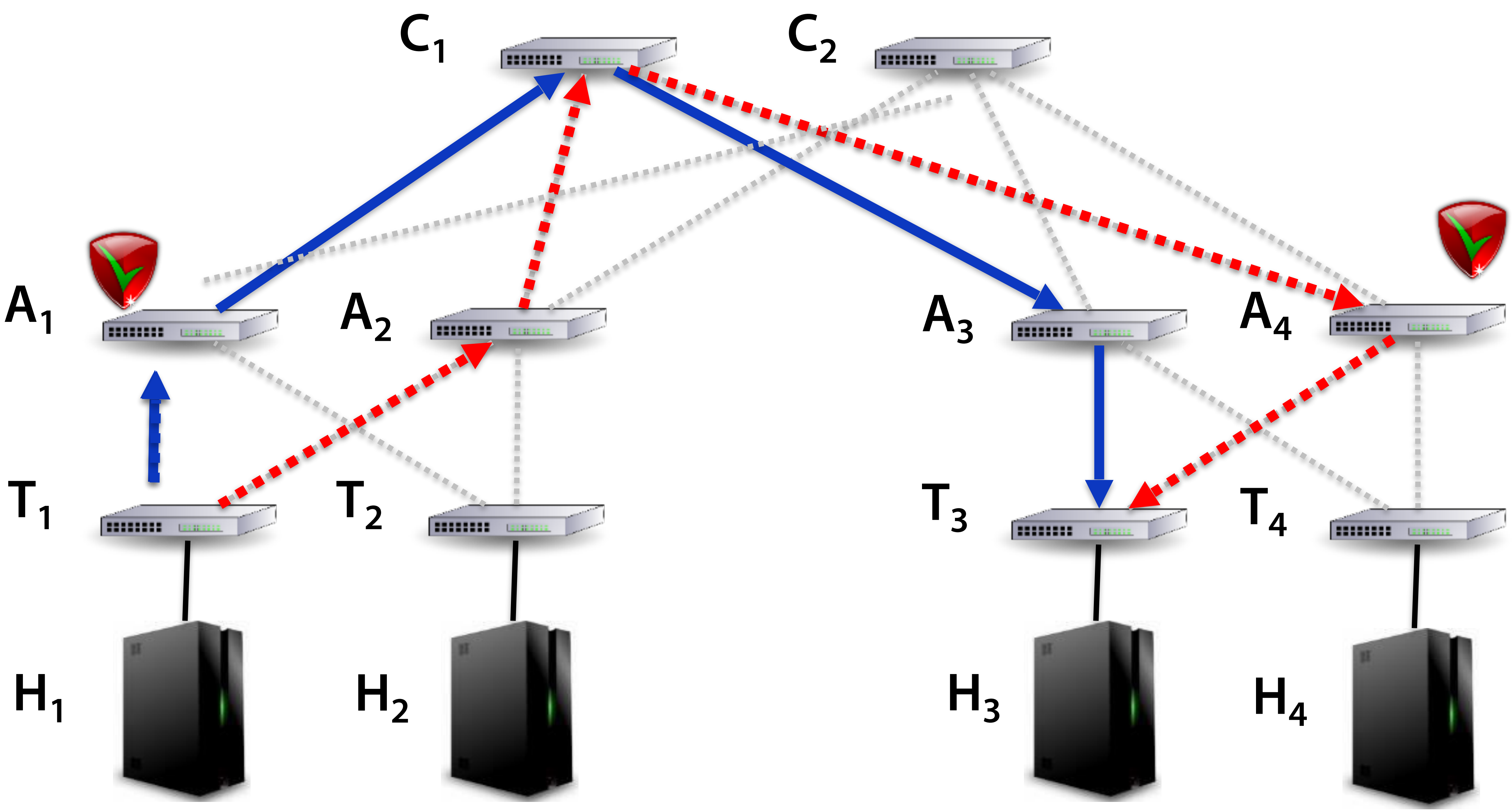
Update: `upd T1; upd C2; upd A3; upd A1` ✔

Naive Update



- **Update:** `upd A2; upd A4; upd T1; upd C1` ✗
- **Update:** `upd A2; upd A4; upd C1; upd T1` ✗
- There is **no update** that ensures per-packet consistency

Relaxing Per-Packet Consistency

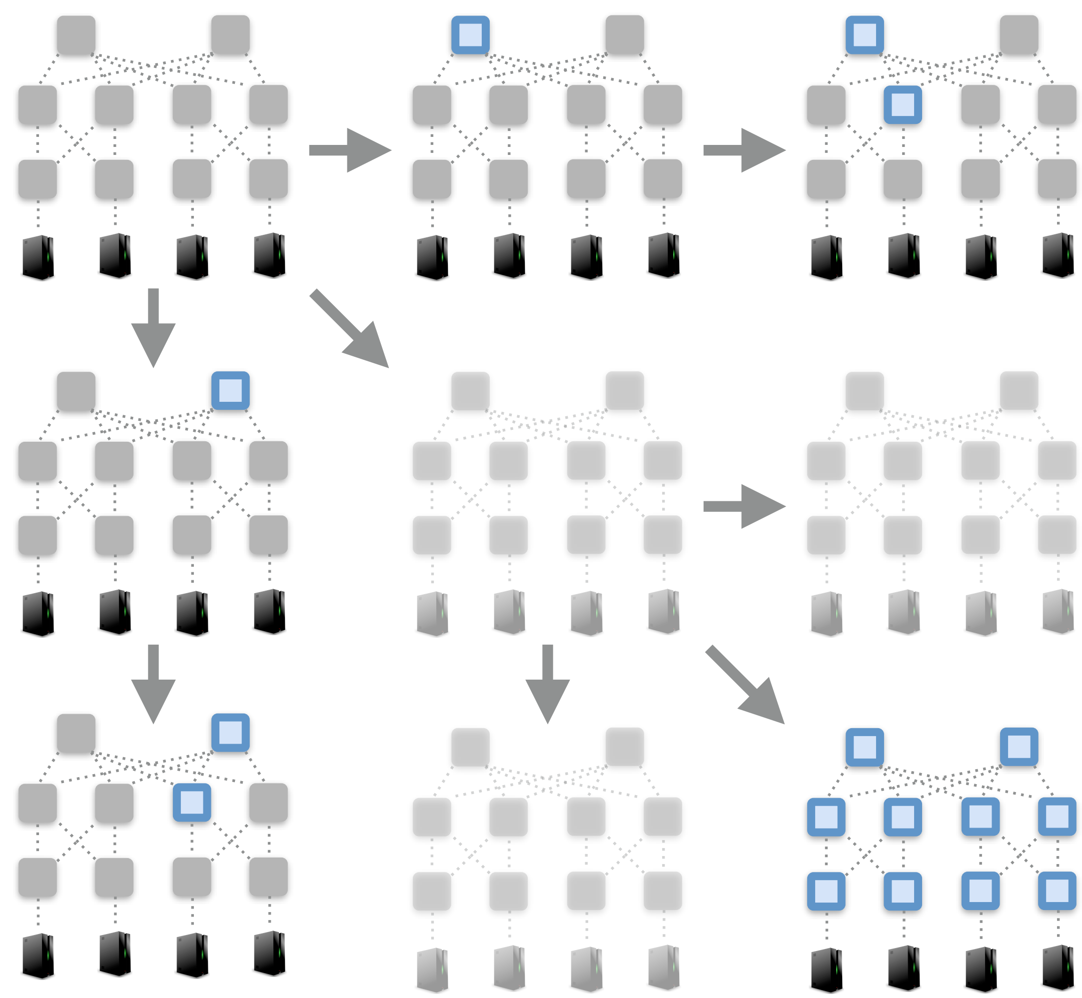


Idea: all packets eventually delivered via A_1 or A_4

- **Update:** `upd A2; upd A4; upd T1; upd C1` ✗
- **Update:** `upd A2; upd A4; upd C1; upd T1` ✓

This Talk

Efficient Synthesis of Network Updates

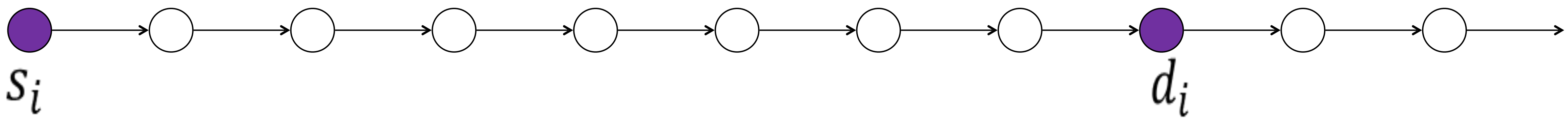


Synthesis for Networks

- Programs are large, but simple and highly structured—e.g., loop free!
- The desired behavior of the network is often clear (at least at an intuitive level)
- Most difficult aspects of network programming stem from limited resources and inherent concurrency

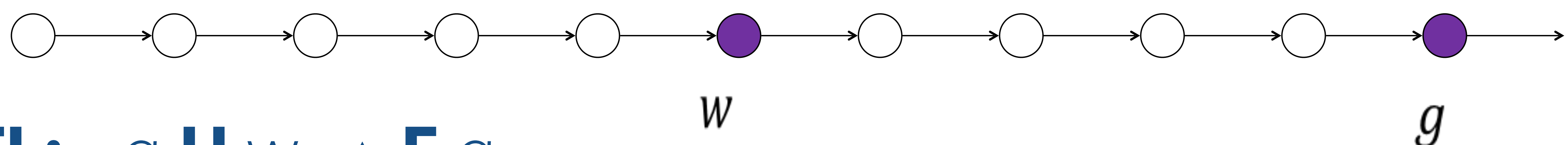
How to Specify Properties?

Reachability: every packet that starts at s_i reaches d_i



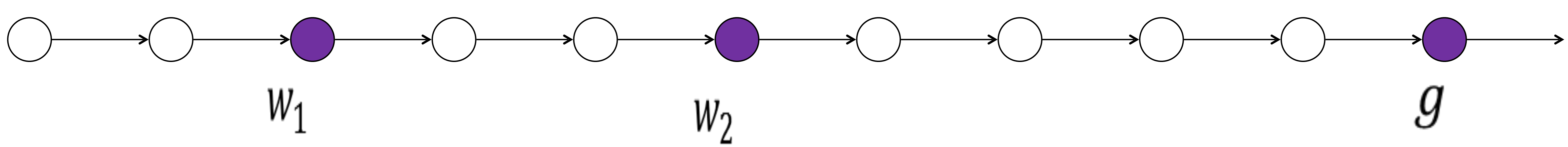
LTL: $\wedge_i \ (s_i \rightarrow \mathbf{F}\ d_i)$

Waypointing: all packets traverse w before exiting



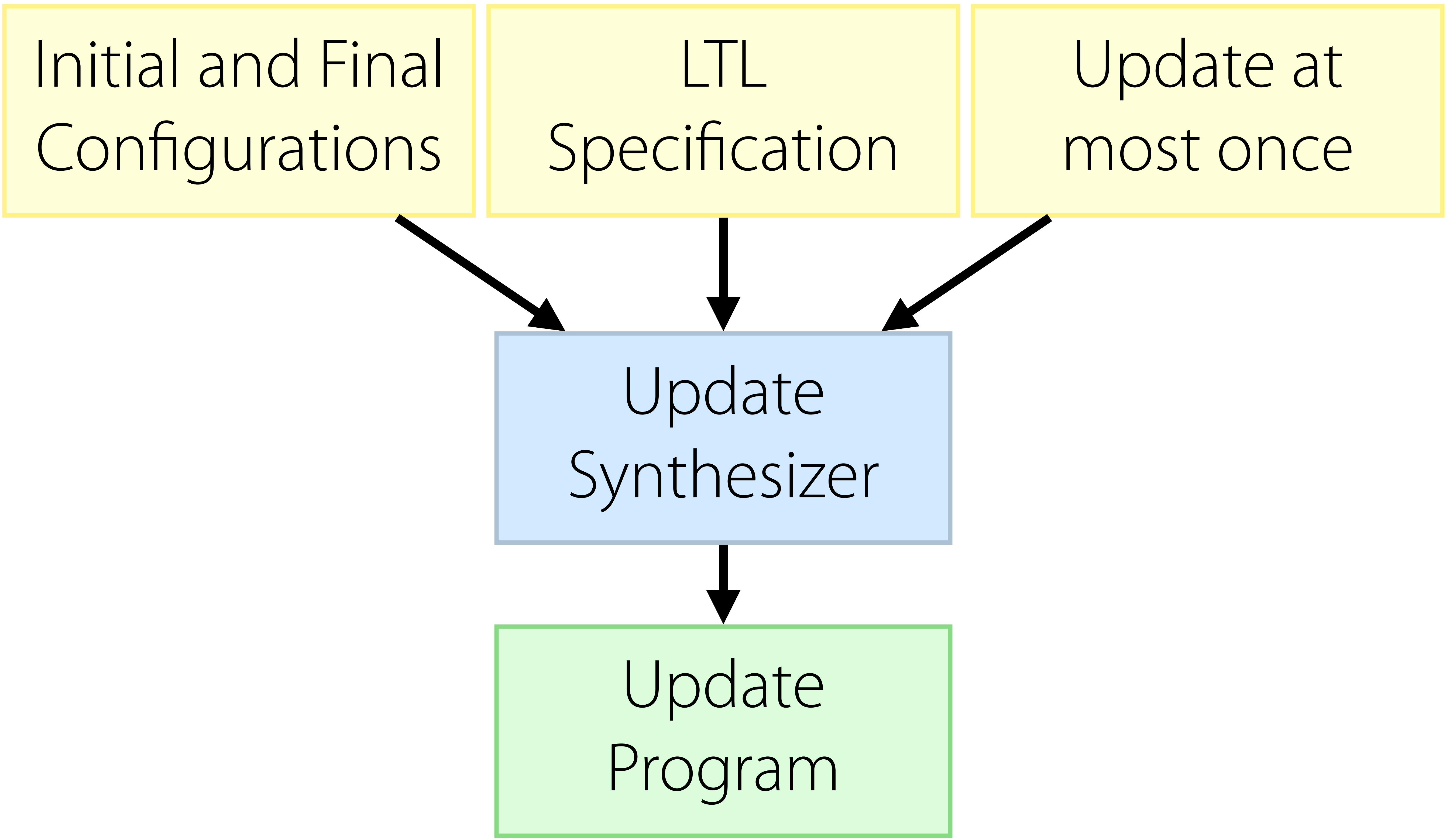
LTL: $\neg g\ \mathbf{U}\ w_2 \wedge \mathbf{F}\ g$

Chaining: all packets traverse w_1 and w_2 before exiting



LTL: $\neg g\ \mathbf{U}\ w_1 \wedge \neg w_1\ \mathbf{U}\ w_2 \wedge \mathbf{F}\ g$

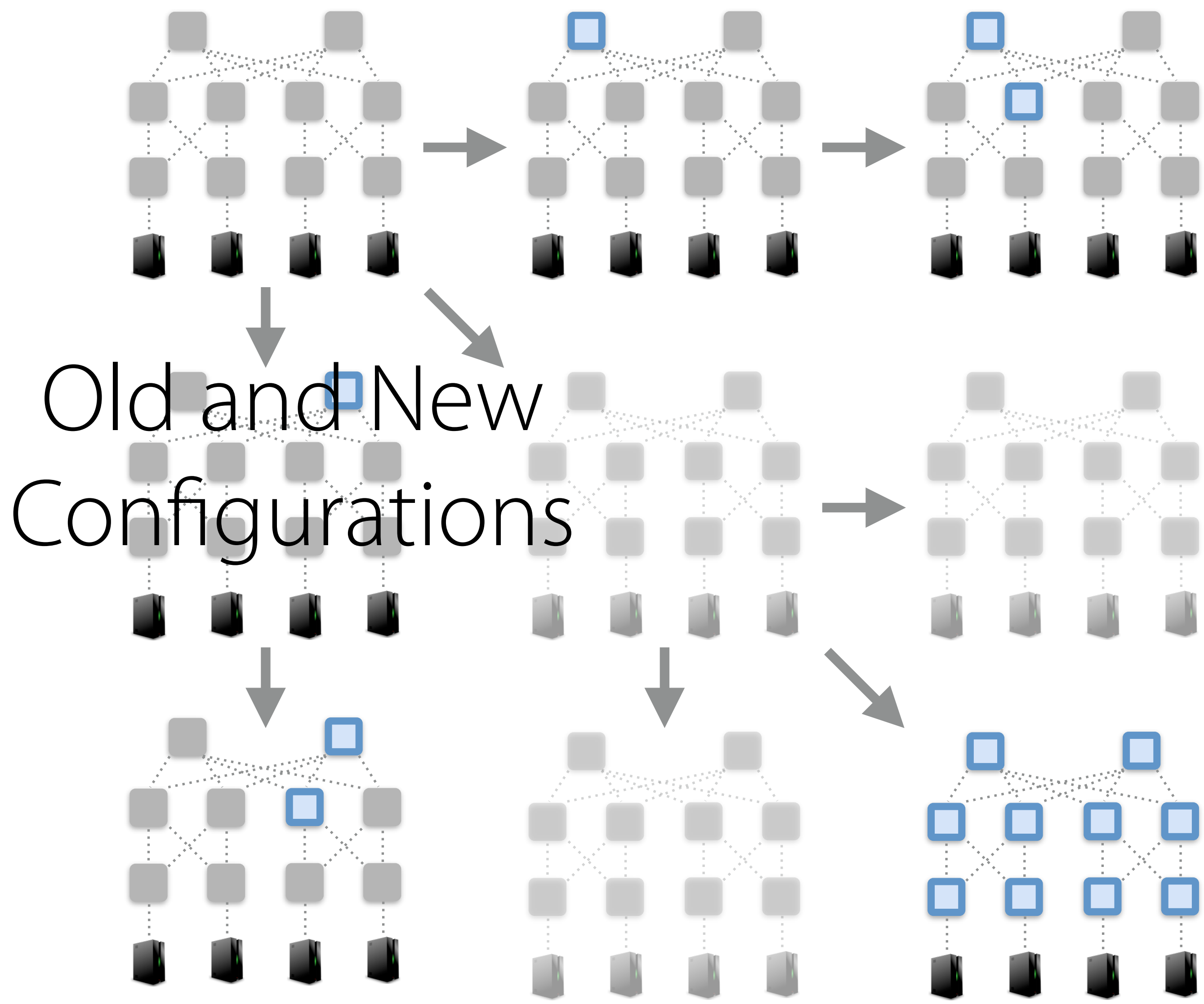
Network Update Synthesis



Synthesis Algorithm

Φ

LTL Specification



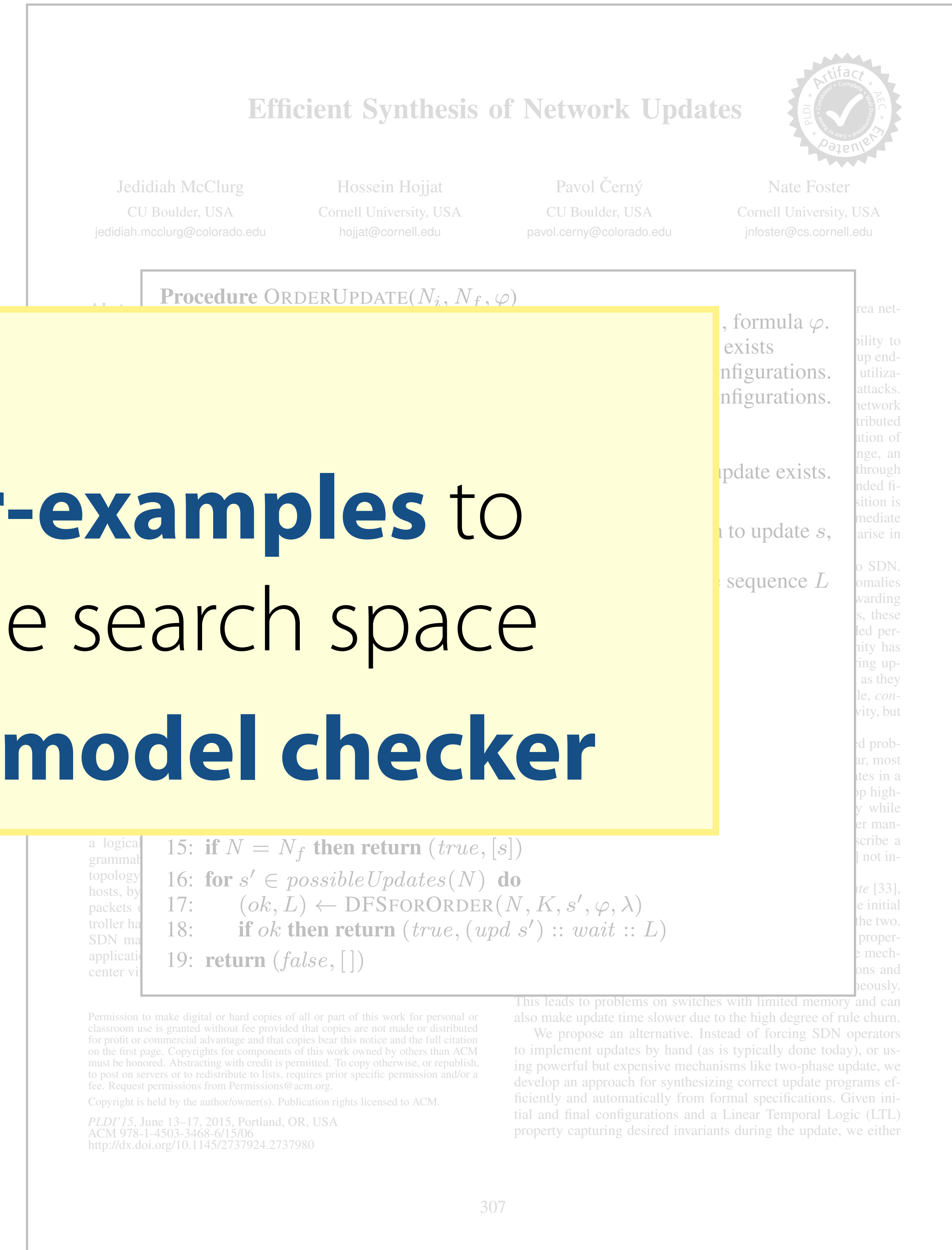
Synthesis Algorithm

Depth-First Search:

- Attempt to update the switches one-by-one
- Backtr… config…

Challe…

- Searc…
- Checking a configuration means solving an LTL model checking problem (PSPACE-complete)!


Efficient Synthesis of Network Updates

Jedidiah McClurg, CU Boulder, USA, jedidiah.mcclurg@colorado.edu
Hossein Hojjat, Cornell University, USA, hojjat@cornell.edu
Pavol Černý, CU Boulder, USA, pavol.cerny@colorado.edu
Nate Foster, Cornell University, USA, jnfoster@cs.cornell.edu

Procedure ORDERUPDATE(N_i, N_f, φ)

15: **if** $N = N_f$ **then return** $(true, [s])$
16: **for** $s' \in possibleUpdates(N)$ **do**
17: $(ok, L) \leftarrow$ DFSFORORDER$(N, K, s', \varphi, \lambda)$
18: **if** ok **then return** $(true, (upd\ s') :: wait :: L)$
19: **return** $(false, [])$

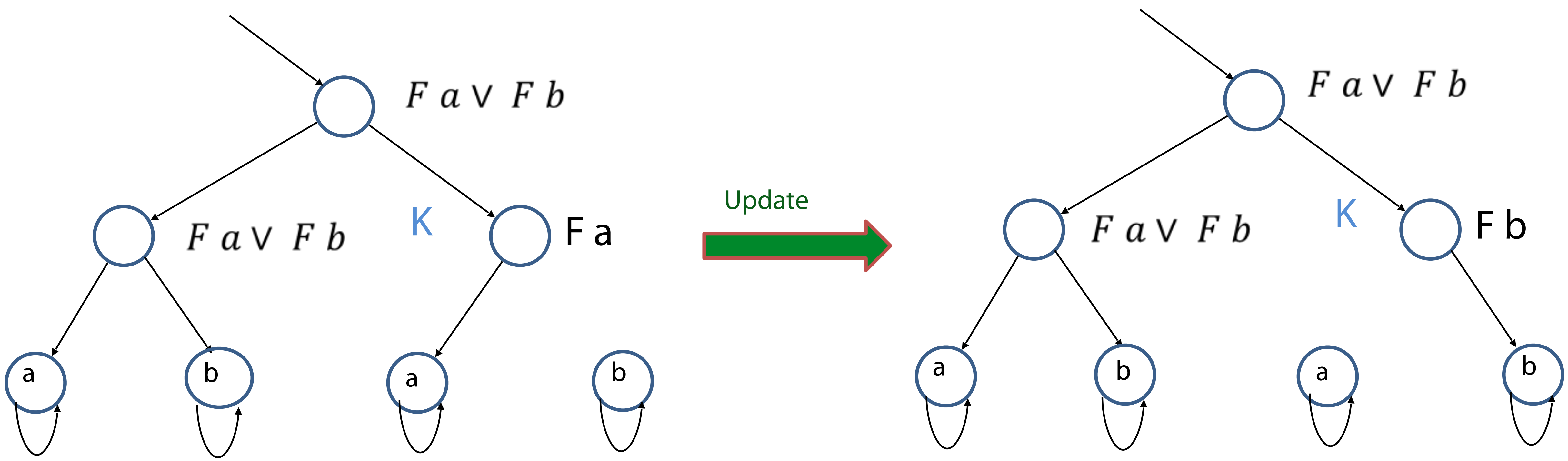
PLDI'15, June 13–17, 2015, Portland, OR, USA
ACM 978-1-4503-3468-6/15/06
http://dx.doi.org/10.1145/2737924.2737980

307

Two main ideas:

- **Learn from counter-examples** to aggressively prune the search space
- Use an **incremental model checker**

Incremental LTL Model Checking

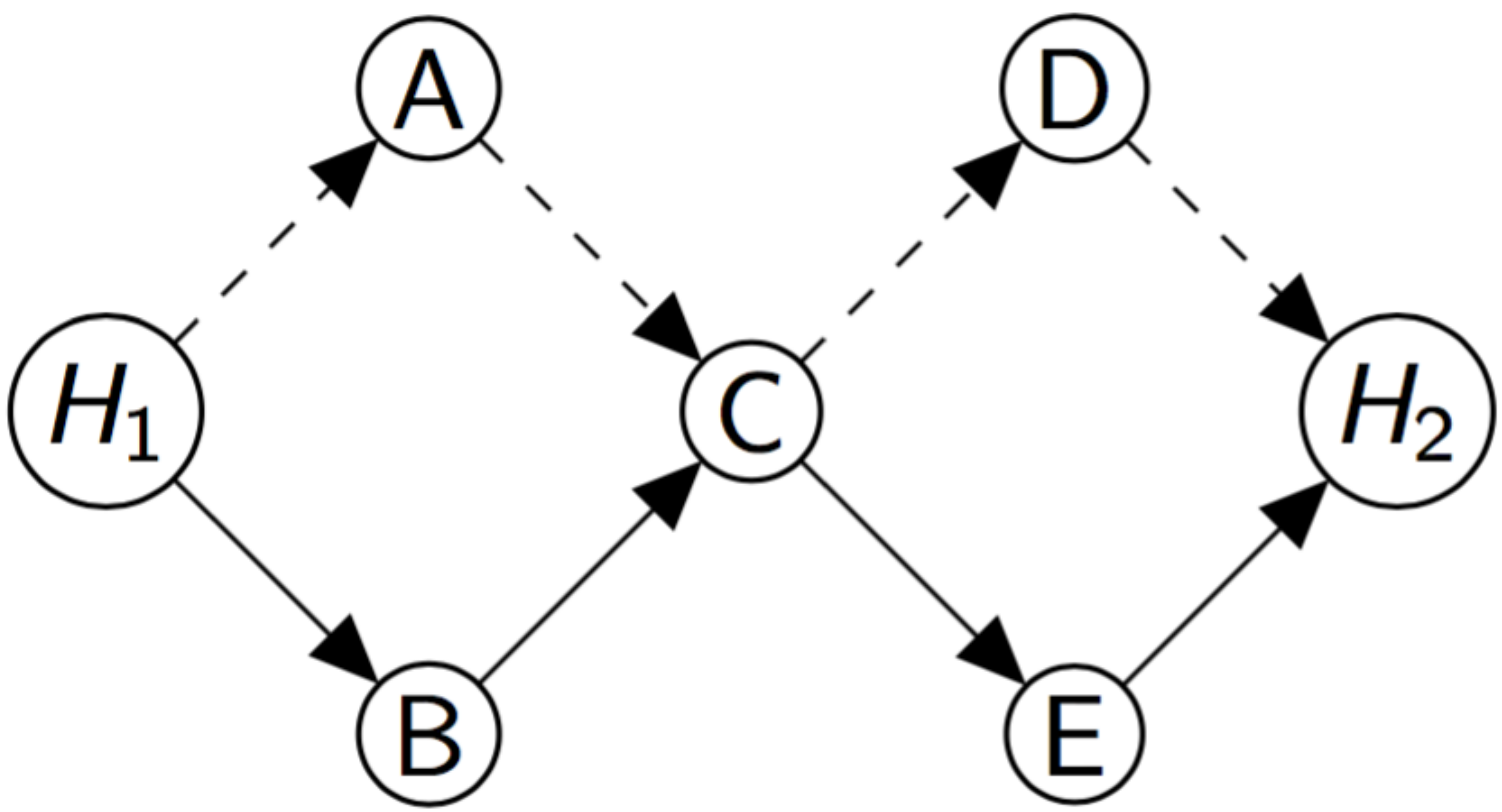


- Networks with loop-free configurations can be molded using DAG-like Kripke structures
- Given a change, can re-label nodes incrementally with a variant of classic Vardi-Wolper model checking

Limitation of Synthesis

For some scenarios there is *no* correct ordering we can use, assuming *at most once* updates

Example: "double diamond" [DISC '16]

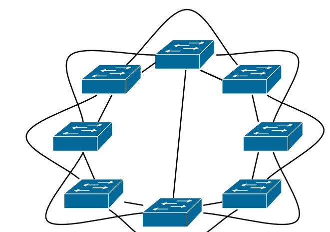


Our implementation reverts to a two-phase update...

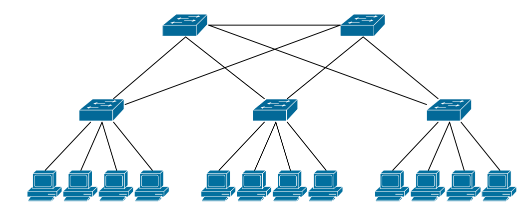
Evaluation

Questions:

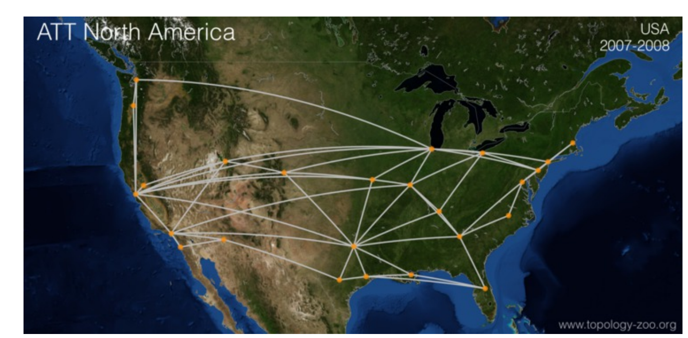
- Scalability of approach:
  - Topology
  - Complexity of specifications
  - Total space explored
- Impact of optimizations:
  - Pruning search space
  - Incremental model checking



Small-world



Fattree

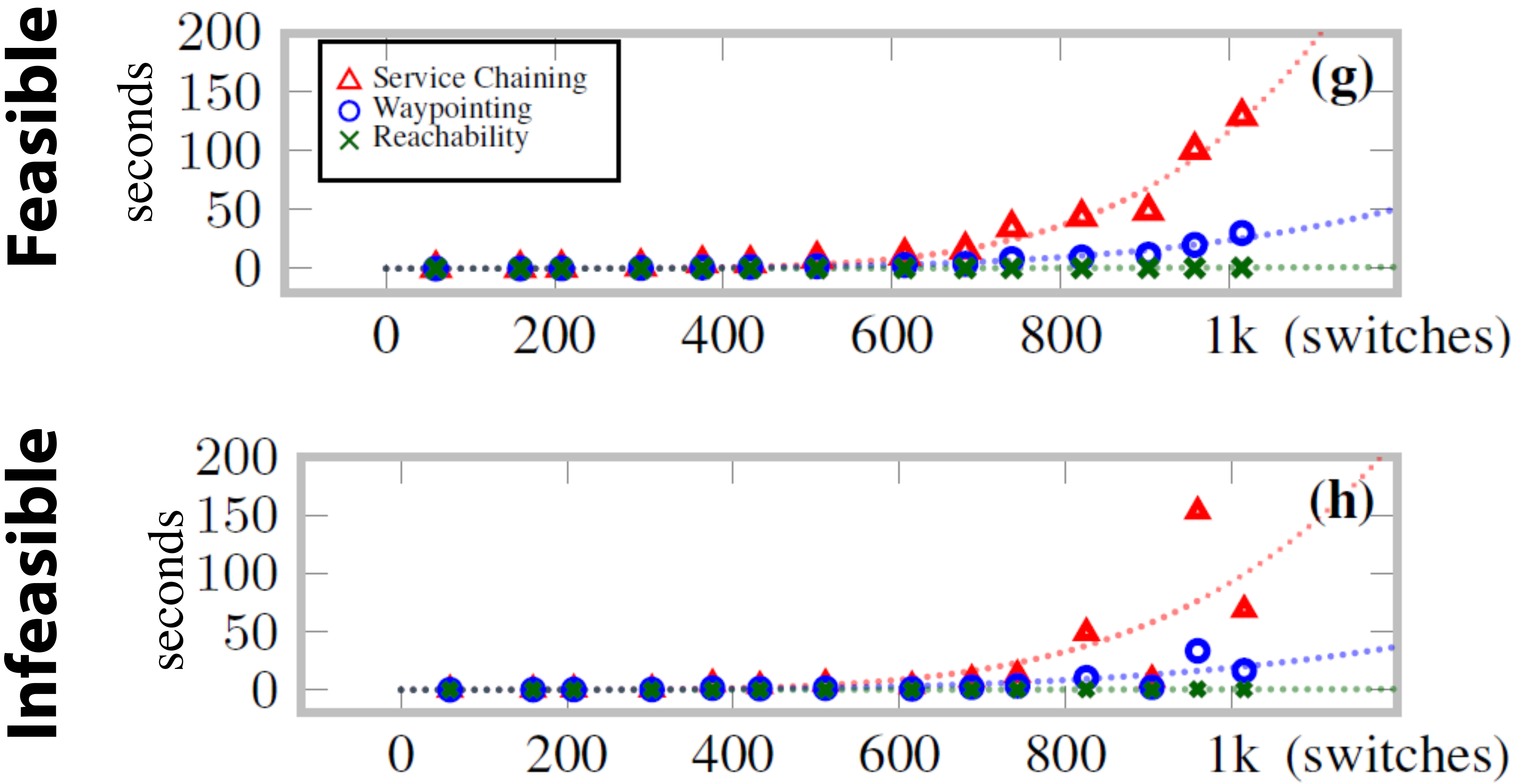


Topology Zoo

Methodology:

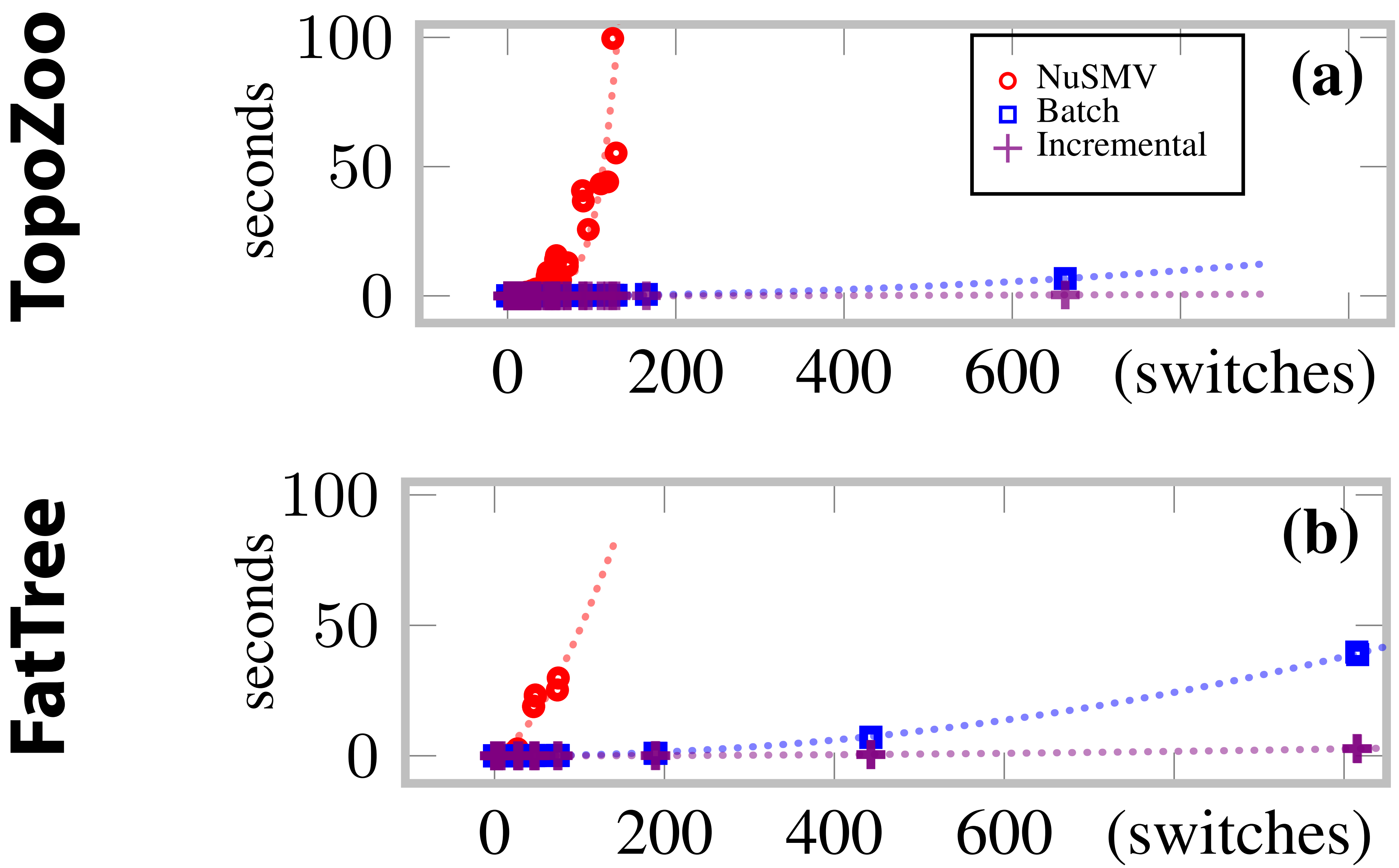
- Real-world topologies (Small World, FatTrees, TopoZoo)
- Synthetic configurations (e.g., shortest-path forwarding)
- Standard properties (reachability, waypointing, etc.)

Scalability



- **Configurations:** "diamond" / "double diamond"
- **Specifications:** reachability, waypointing, chaining

Impact of Optimizations



- **Configurations:** shortest-path forwarding
- **LTL Specification:** all-pairs reachability

Reading

- Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. **Abstractions for Network Update**. In ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), August 2012.
- Jedidiah McClurg, Hossein Hojjat, Pavol Cerny, and Nate Foster. **Efficient Synthesis of Network Updates**. In ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), June 2015.
- Pavol Černý, Nate Foster, Nilesh Jagnik, Jedidiah McClurg. **Optimal Consistent Network Updates in Polynomial Time.** In International Symposium on Distributed Computing (DISC), July 2016.