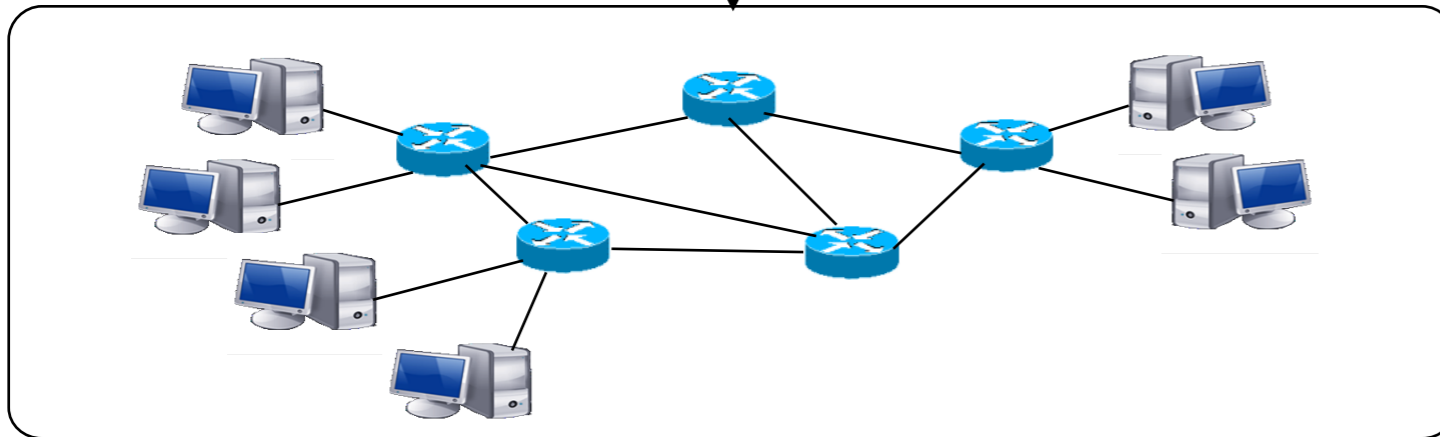
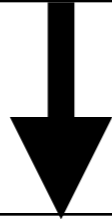
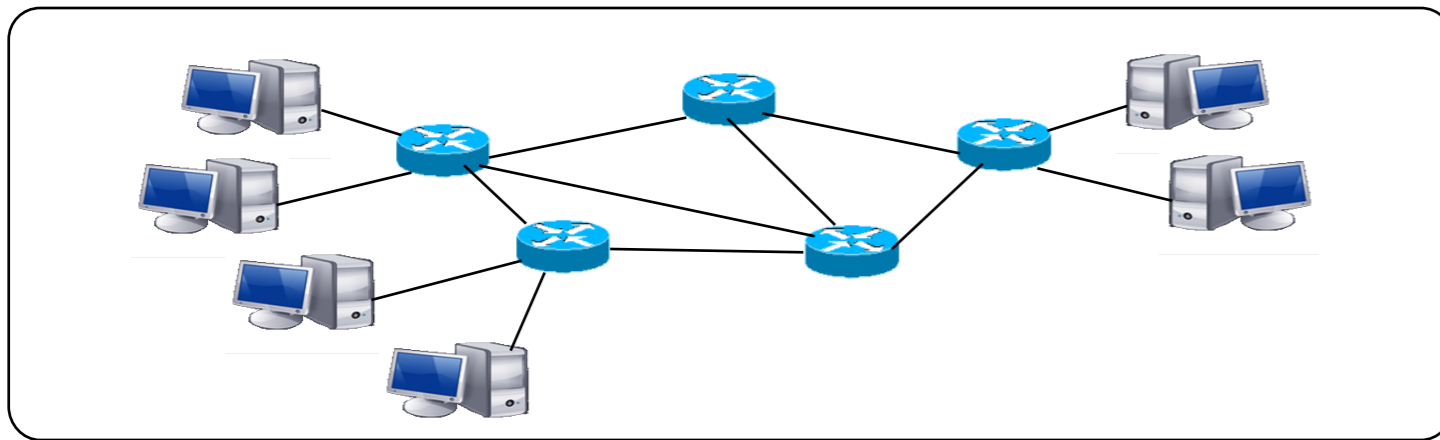


Enforcing Customizable Consistency Properties in Software-Defined Networks

Wenxuan Zhou, Dong Jin, Jason Croft,
Matthew Caesar, Brighten Godfrey

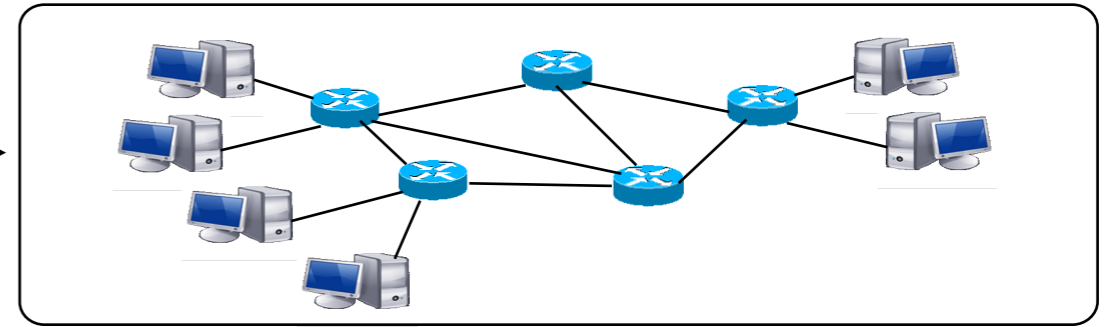
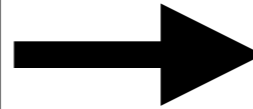
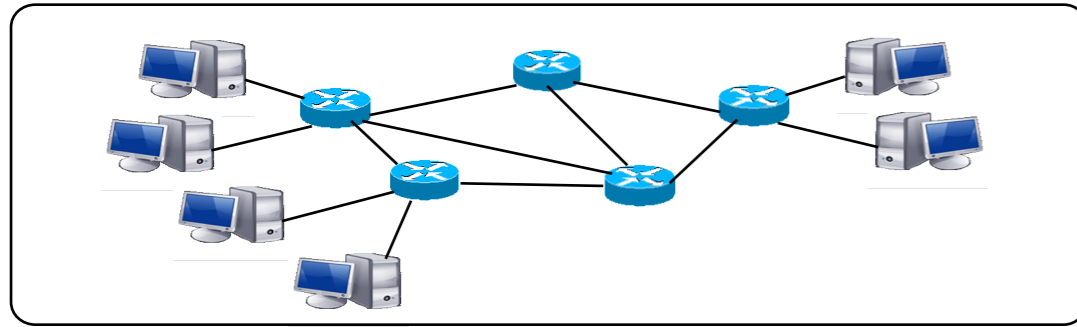


Network changes

- control applications,
- changes in traffic load,
- system upgrades,
- ...

Keeping network correct consistently over time.

-- *Network Consistency*



1. Correctness at every step
2. Customizable properties
3. With efficient update installation

What is Correctness?

- firewall traversal,
- access control,
- balanced load,
- loop freedom,
- ...

Problem Statement

1. Consistency at every step
2. Customizable consistency properties
3. Efficient updates installation

Is it possible to efficiently ensures
customizable correctness properties
as the network evolves?

ΣΣ ΕΠΕ ΠΕΓΜΟΛΚ ΕΛΟ|ΛΕΣ;

Prior Work

VeriFlow: Verifying Network-Wide Invariants in Real Time

Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, P. Brighten Godfrey
University of Illinois at Urbana-Champaign
{ahurshid, xuanzou2, wzhou10, caesar, pbg}@illinois.edu

Abstract

Networks are complex and prone to bugs. Existing tools that check network configurations often use the data plane state to verify correctness. However, this approach is not scalable. In this paper, we propose VeriFlow, a system that verifies network-wide invariants in real time. VeriFlow uses a symbolic approach to verify network-wide invariants. It must be able to control an operator's actions. VeriFlow must be able to control an operator's actions. VeriFlow must be able to control an operator's actions.

1 Introduction

Packet forwarding in modern networks is a complex process, involving codependent functions running on hundreds of thousands of devices, such as routers, switches, and firewalls from different vendors. As a result, a substantial amount of effort is required to ensure networks' correctness, security and fault tolerance. However, faults in the network state arise commonly in practice, including loops, suboptimal routing, black holes and access control violations that make services unavailable or prone to attacks (e.g., DDoS attacks). Software-Defined Networking (SDN) promises to ease the development of network applications through logically-centralized network programmability via an open interface to the data plane, but bugs are likely to remain problematic since the

complexity of software will increase. Moreover, SDN allows multiple applications or even multiple users to program the same physical network simultaneously, potentially introducing new bugs. Symbolic verification of all for large networks is infeasible. I must be able to control an operator's actions. VeriFlow must be able to control an operator's actions. VeriFlow must be able to control an operator's actions.

This paper studies the question, *Is it possible to check network-wide correctness in real time as the network evolves?* If we can check each change to forwarding behavior before it takes effect, we can raise alarms immediately, and even prevent bugs by blocking changes that violate important invariants. For example, we could prohibit changes that violate access control policies or cause forwarding loops.

However, existing techniques for checking networks are inadequate for this purpose as they operate on timescales of seconds to hours [10, 17, 19]. Deploying updates for processing can harm consistency of network state, and increase reaction time of protocols with real-time requirements such as routing and fast failover; and processing a continuous stream of updates in a large network can be prohibitively expensive. The average run time of reachability tests in [17] is 13 seconds, and it takes a few hundred seconds to perform reachability checks in Asterix [19].

Dynamic Scheduling of Network Updates

Xin Jin¹, Hongqiang Harry Liu², Rohan Gandhi³, Srikanth Kandula⁴,
Ratul Mahajan¹, Ming Zhang², Jennifer Rexford³, Roger Wattenhofer⁴
¹Microsoft Research, ²Princeton University, ³Yale University, ⁴Purdue University, ⁵ETH Zurich

Abstract—We present Dionysus, a system for fast, consistent network updates in software-defined networks. Dionysus encodes as a graph the consistency-related dependencies among updates at individual switches, and it then dynamically schedules these updates based on runtime differences in the update speeds of different switches. This dynamic scheduling is the key to its speed: prior update methods are slow because they pre-determine a schedule, which does not adapt to runtime conditions. Tested experiments and data-driven simulations show that Dionysus improves the median update speed by 55-80% in both wide area and data center networks compared to prior methods.

Categories and Subject Descriptors: C.2.1 (Computer-Communication Networks): Network Architecture and Design—Network Communications; C.2.3 (Computer-Communication Networks): Network Operations

Keywords: Software-Defined

1. Introduction

Many researchers have worked on network updates. This approach to route computation [1]; plans [2, 3]; reduce one path [5, 6, 7, 8, 9]; being by frequently updating periodically or based on triggers such as failures. This state consists of a set of rules that determine how switches forward packets. A common challenge faced in all centrally-controlled networks is consistently and quickly updating the data plane. Consistency implies that certain properties should not be violated during network updates, for instance, packets should not loop (loop freedom) and traffic arriving at a link should not exceed its capacity (congestion freedom). Consistency requirements impose dependencies on the order in which rules can be updated at switches. For instance, for congestion freedom, a rule update that brings a new flow to a link must occur after an update that removes an existing flow if the link cannot support both flows simultaneously. Not obeying update ordering requirements can lead to inconsistencies such as loops, blackholes, and congestion.

Current methods for consistent network updates are slow because they are based on static ordering of rule updates [9, 10, 11, 12]. They pre-compute an order in which rules must be updated, and this order does not adapt to runtime differences in the time it takes for individual switches to apply updates. These differences inevitably arise because of disparities in switches' hardware and CPU load and the variabilities in the time it takes the centralized controller to make remote procedure calls (RPC) to switches. In BA, a centrally-controlled wide area network, the time of the 99th percentile to the median delay to change a rule at a switch was found to be over five (5 versus 1 second) [8]. Further, some switches can "struggle" taking substantially more time than average (e.g., 10-100x) to apply an update. Current methods can stall in the face of straggling switches.

The speed of network updates is important because it determines the agility of the control loop. If the network is being updated in response to a failure, slower updates imply a longer period during which the network is in a sub-optimal state.

Further, many systems load, both in the wide area and in data centers, at their effectiveness in networks. For example, traffic engineering (e.g., network updates. It is not multiple valid rule sets [1]) dynamically set of switches can lead to many consistency properties, including all the ones that have been explored by prior work [9, 10, 11, 12, 13].

We face two main challenges in practically realizing our approach. The first is devising a compact way to represent multiple valid orderings of rule updates; there can be exponentially many such orderings. We address this challenge using a dependency graph in which nodes correspond to rule updates and network resources, such as link bandwidth and switch rule memory capacity, and directed edges denote dependencies among rule updates and network resources. Scheduling updates in any order, while respecting dependencies, guarantees consistent updates.

The second challenge is scheduling updates based on dynamic behavior of switches. This problem is NP-complete in the general case, and making matters worse, the dependency graph can also have cycles. To schedule efficiently, we develop greedy heuristics based on preferring critical paths and strongly connected components in the dependency graph [14].

We instantiate our approach in a system called Dionysus and evaluate it using experiments on a modern-sized testbed and large-scale simulations. Our simulations are based on topology and traffic data from two real networks, one wide-area network and one data center network. We show that Dionysus improves the median network update speed by 55-80%. We also show that its faster updates lower congestion and packet loss by over 40%.

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC), Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, Roger Wattenhofer (ETH)

Abstract—We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-optimizing the network's data plane to match current traffic demand. But done simplistically, these re-optimizations can also cause severe, transient congestion because different switches may receive updates at different

in some cases, services send traffic whenever they want and however much they want. As a result, the network cycles through periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-subscribed on average. Observe that network usage does not have to be this way if we can exploit the characteristics of inter-DC traffic. Some inter-DC services are

Fixed Consistency Property

reach end users [15]. It is an expensive resource, with associated annual cost of 100s of millions of dollars, so it provides 100x of Gbps to Tbps of capacity over long distances. However, providers are unable to fully leverage this investment today. Inter-DC WANs have extremely poor efficiency; the average utilization of even the busiest links is 40-60%. One culprit is the lack of coordination among the services that use the network. Having coarse, static limits

sensitive traffic. The underlying problem is that the updates are not atomic as they require changes to multiple switches. Even if the before and after states are not congested, congestion can occur during updates if traffic that a link is supposed to carry after the update arrives before the traffic that is supposed to leave has left. The extent and duration of such congestion is worse when the network is busier and has larger RTTs (which lead to greater temporal disparity in the application of updates). Both these conditions hold

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org. SIGCOMM '12, August 13-17, 2012, Helsinki, Finland. Copyright 2012 ACM 978-1-4503-2036-6/12/08...\$15.00.

In some networks, fault tolerance is another reason for low utilization; the network is provisioned such that there is ample capacity even after (contingent) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing technologies can tag and protect such traffic in the face of failures [12].

Abstractions for Network Update

Mark Reiblat¹, Nate Foster², Jennifer Rexford³, Cole Schlesinger⁴, David Walker⁵
¹Cornell, ²Cornell, ³Princeton, ⁴Princeton, ⁵Princeton

ABSTRACT

Configuration changes are a common source of instability in networks, leading to outages, performance degradation, and security vulnerabilities. Even when the initial and final configurations are correct, the update process itself often steps through intermediate configurations that are inconsistent.

Networks exist in a constant state of flux. Operators frequently modify routing tables, adjust link weights, and change access control lists to perform tasks from planned maintenance, to traffic engineering, to patching security vulnerabilities, to migrating virtual machines in a datacenter. But even when updates are planned well in advance, they are difficult to implement correctly, and can result in network outages, or the death

guaranteed to between configurations, per-packet for implement APIs like OpenFlow, and properties. W several updates consistent updating the correct describe the res effectiveness.

is for network manage of progress and faithfully is for network manage of progress and faithfully

Categories

C.2.1 (Computer-Communication Networks)

General Terms

Design, Languages, Theory

Keywords

Consistency, planned change, software-defined networking, OpenFlow, network programming languages, Forecasts.

1. INTRODUCTION

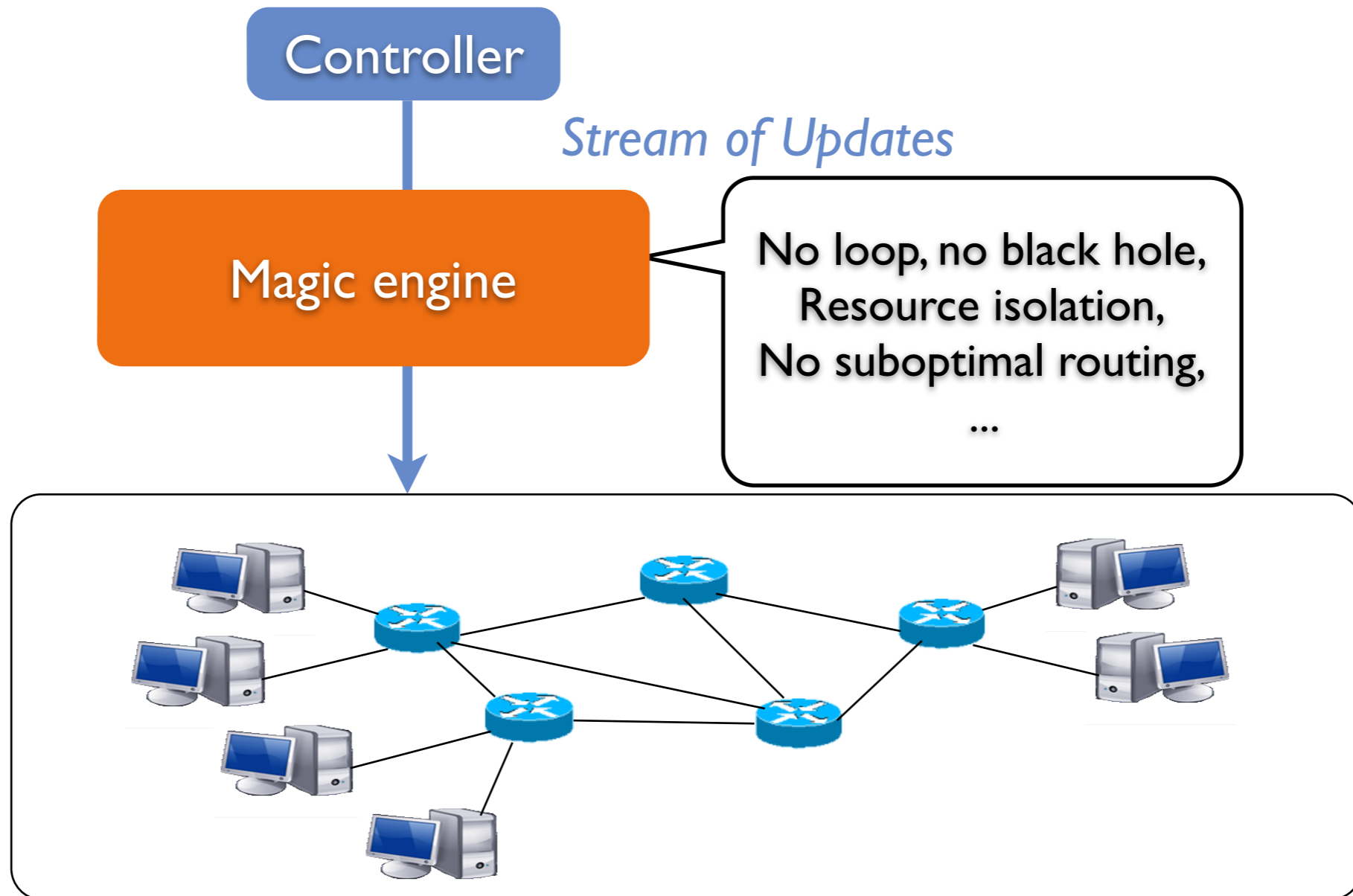
"Nothing endures but change." —Heracleitus

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyright for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SIGCOMM '12, August 13-17, 2012, Helsinki, Finland. Copyright 2012 ACM 978-1-4503-2036-6/12/08...\$15.00.

Programmers can use the interface to build robust applications on top of a reliable foundation. The mechanisms, while possibly complex, would be implemented once by experts, tested and optimized, and used over and over, much like register allocation or garbage collection in a high-level programming language.

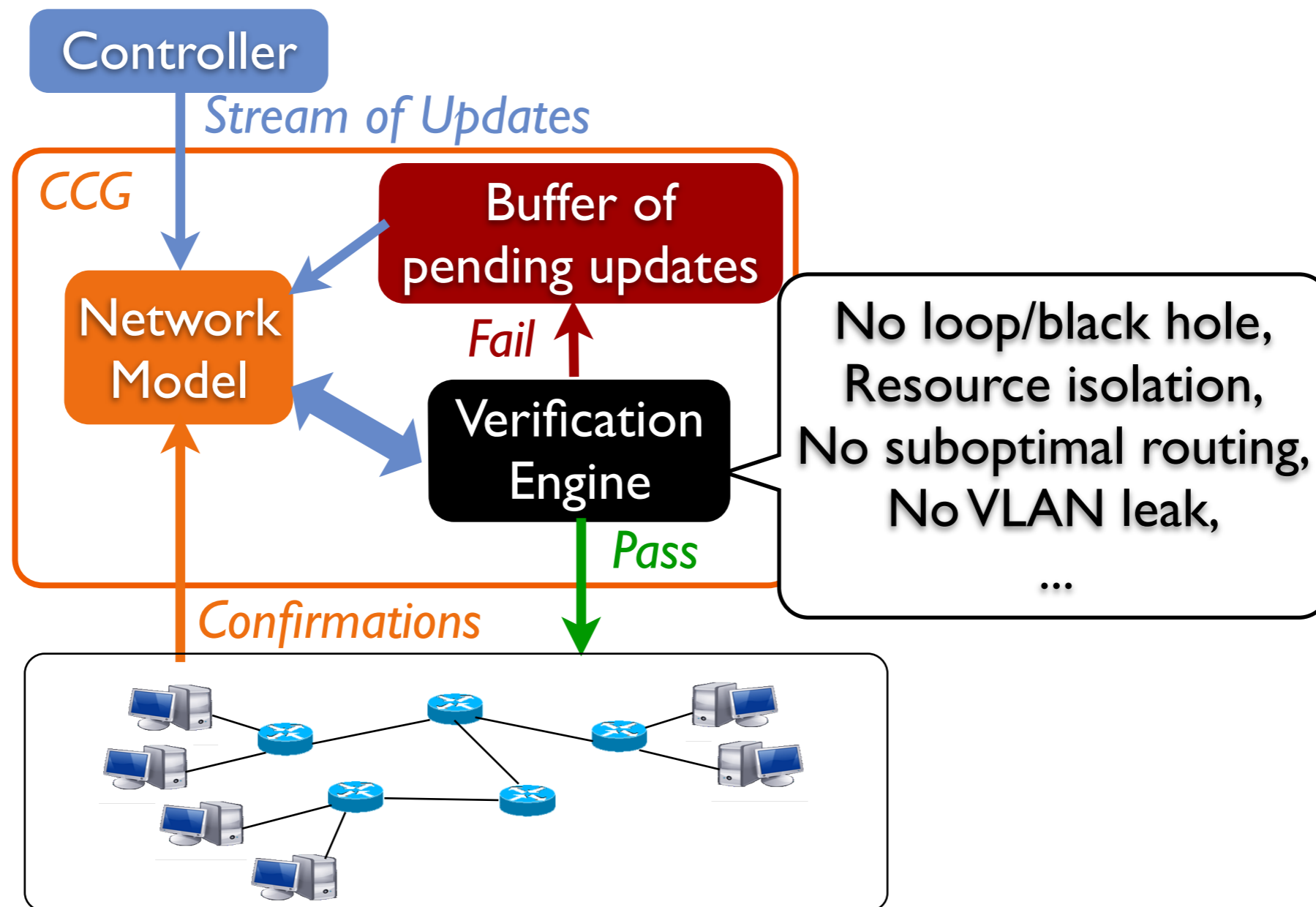
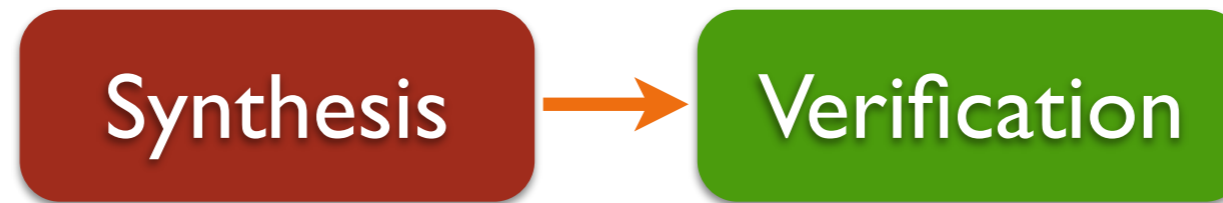
Software-defined networks. The emergence of Software-Defined Networks (SDN) presents a tremendous opportunity for developing general abstractions for managing network updates. In an SDN, a program running on a logically-centralized controller manages the network directly by configuring the packet-handling mechanisms in the underlying switches. For example, the OpenFlow API allows a controller to install rules that each specify a pattern that matches on bits in the packet header, actions performed on matching packets (such as drop, forward, or divert to the controller), a priority (to distinguish between overlapping patterns), and timeouts to allow the switch to remove stale rules [10]. Hence, whereas today network operators have (at best) indirect control over the distributed implementations of routing, access control, and load balancing, SDN platforms like OpenFlow provide programmers with direct control over the processing of packets in the network. However, despite the conceptual appeal of centralized control,

Ideally given arbitrary invariants, a sequence with minimized overhead is produced



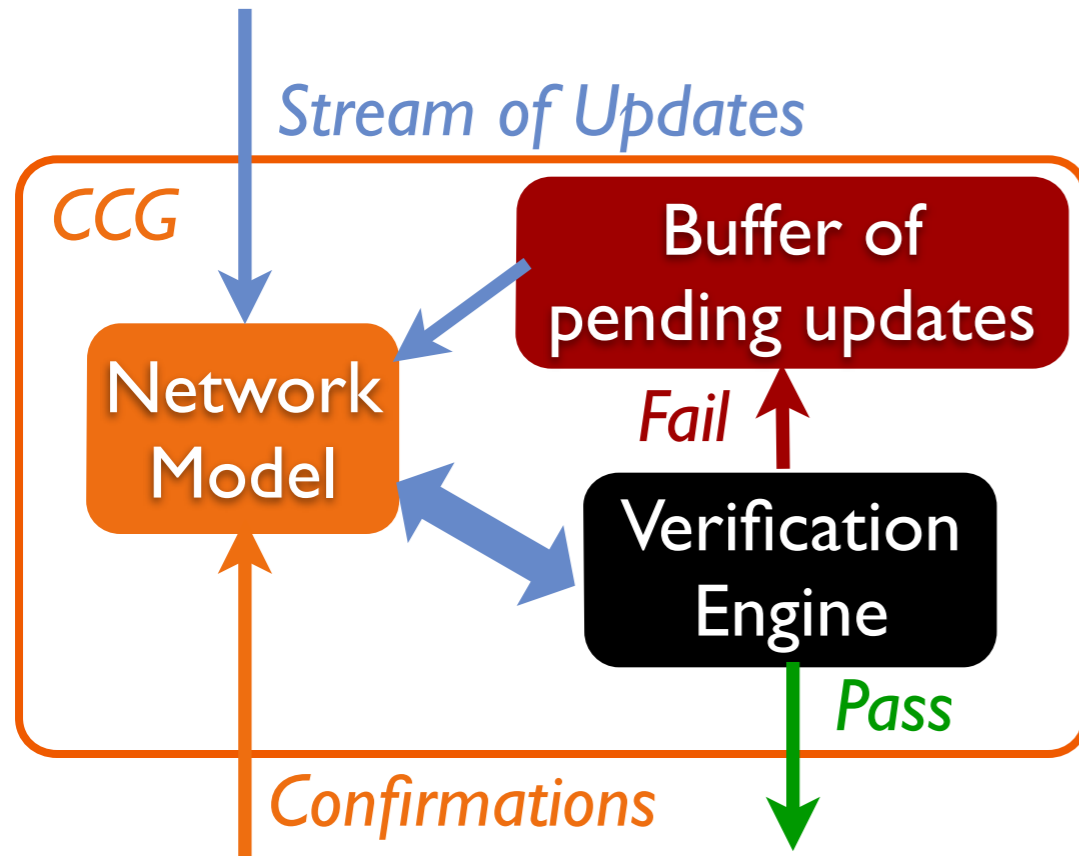
Our design: Customizable Consistency Generator

Key insight:



Our design: Customizable Consistency Generator

Challenges:

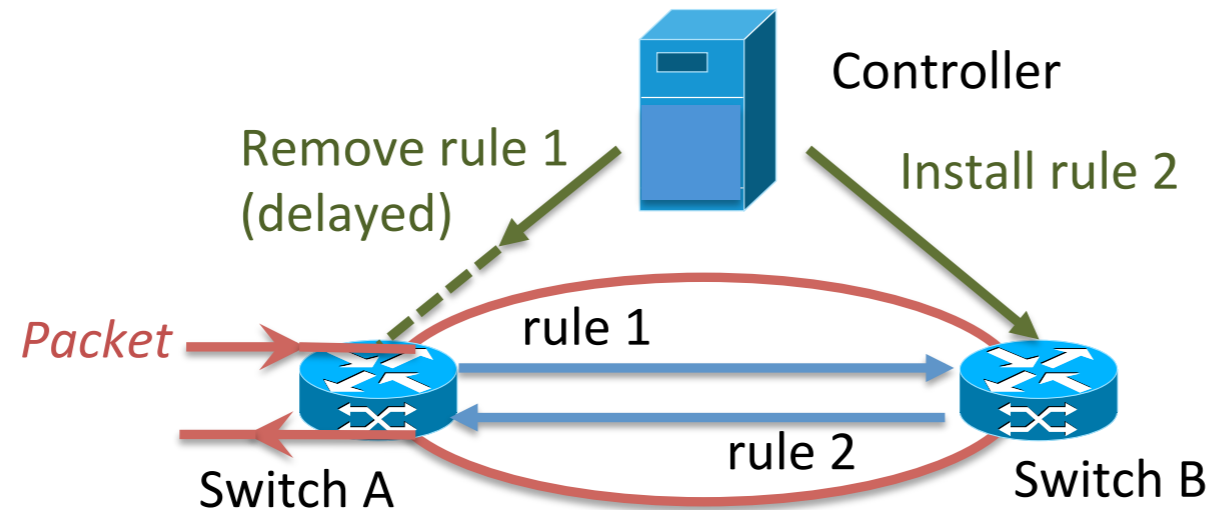


- Greedy algorithm may get stuck
 - * identify the scope of cases that guarantees no deadlock
 - * For other cases, a more heavyweight update technique as a fallback, triggered rarely in practice
- Distributed nature of networks (uncertainty)
 - * compact *uncertain forwarding graph*
 - * verification optimization

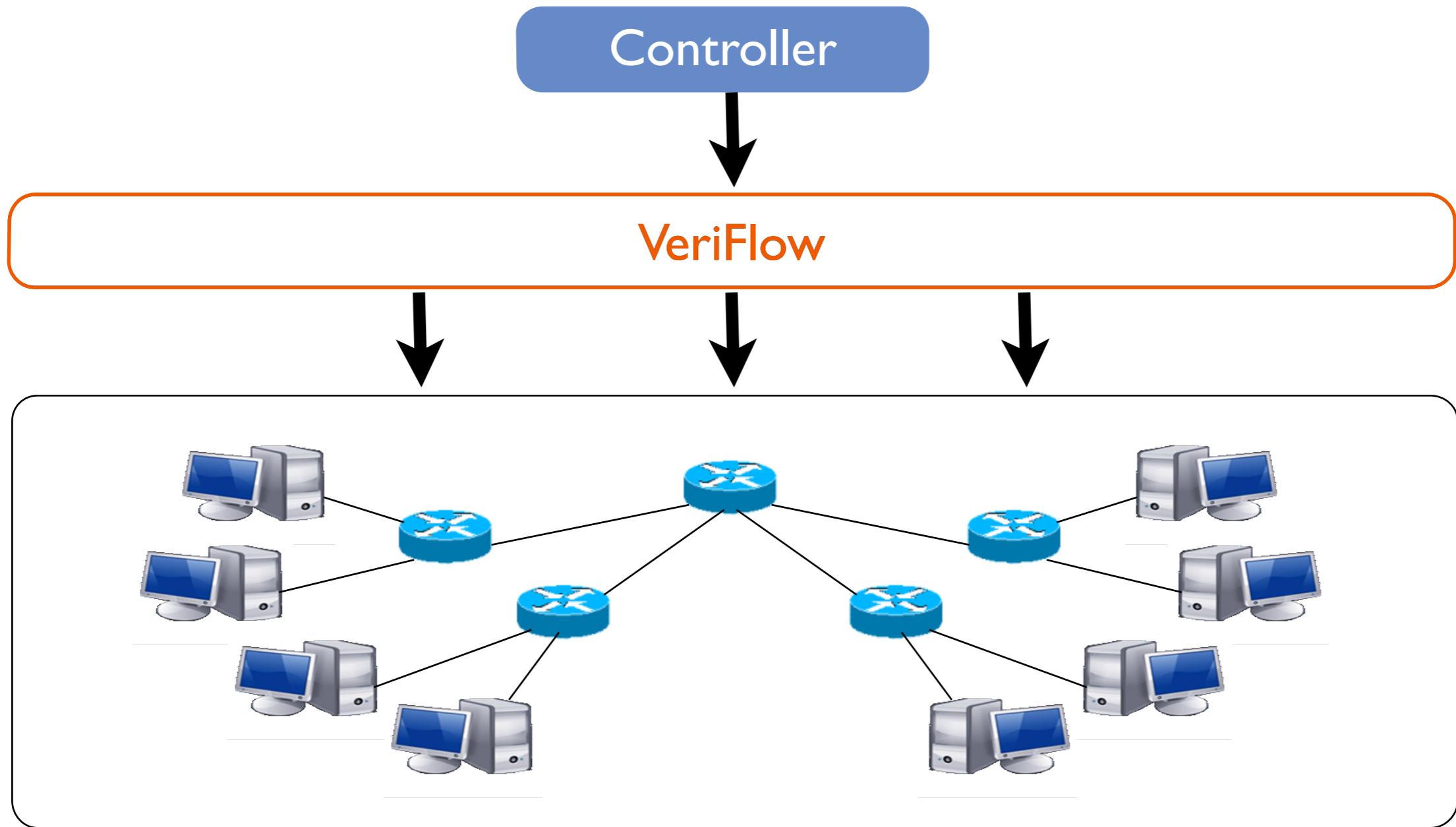
Network Uncertainty

The “uncertainty” of an observation point tasked with instilling updates in knowing the current network state.

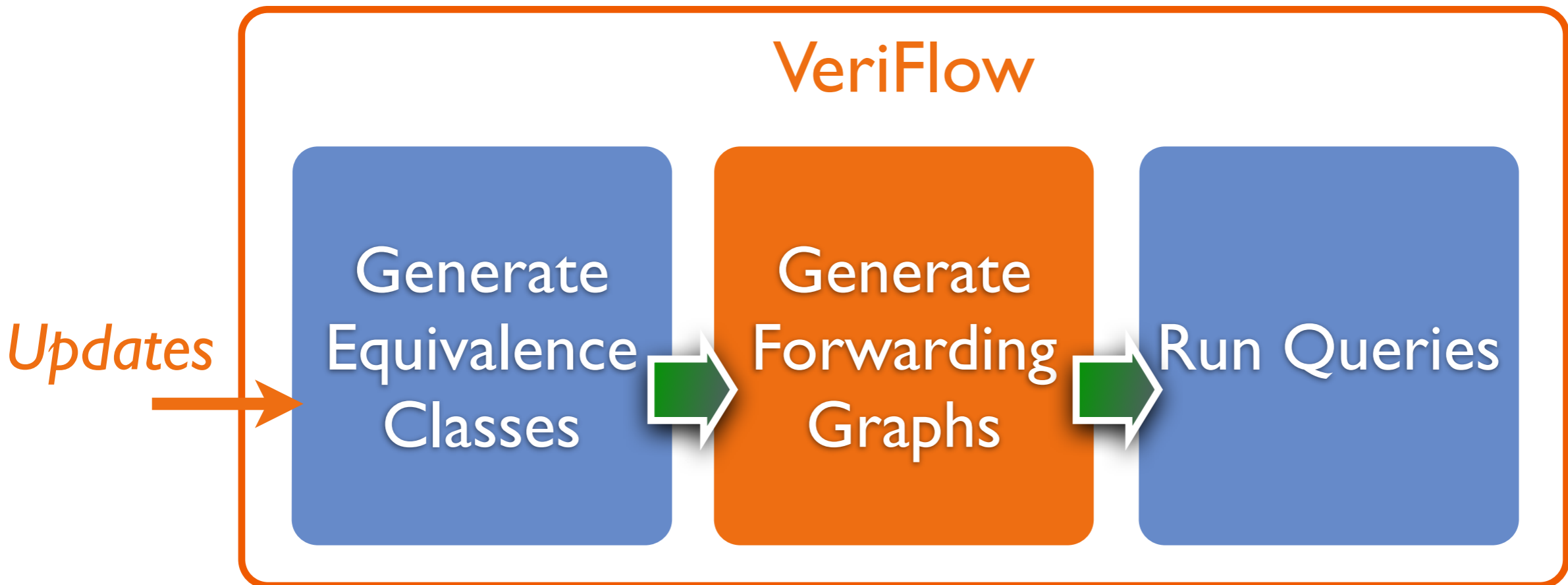
May deviate network behavior away from desired properties.



Uncertainty-aware Modeling Basis: VeriFlow

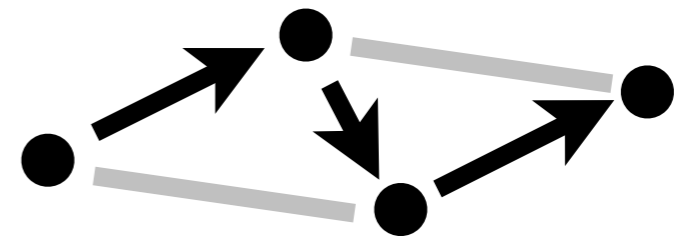


Uncertainty-aware Modeling Basis: VeriFlow



Equivalence class: Packets experiencing the same forwarding actions throughout the network.

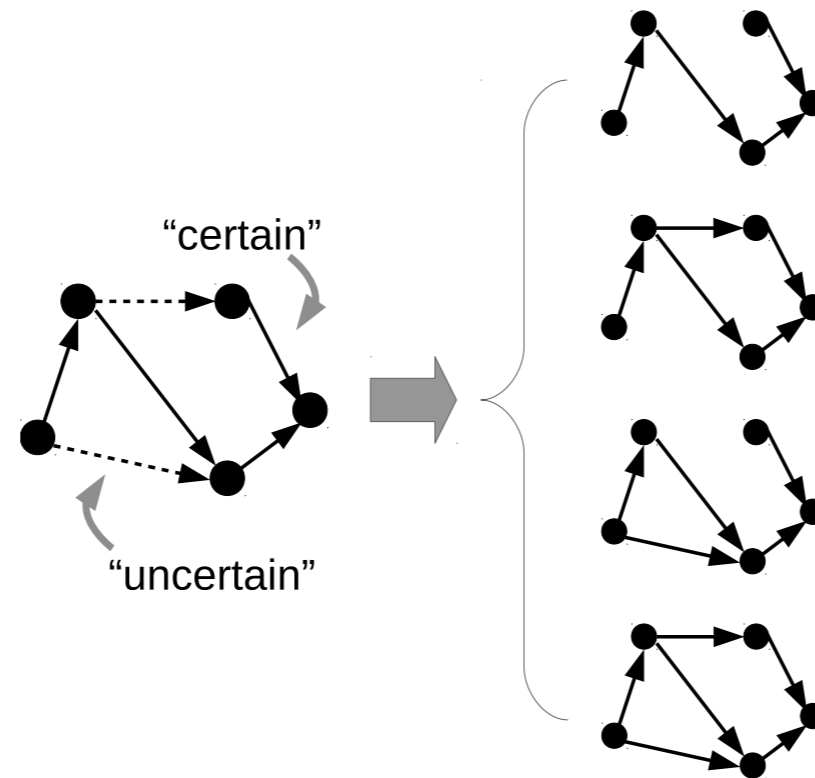
Forwarding graphs:



Uncertainty-aware Modeling

Naively, represent every possible network state $O(2^n)$

Uncertain graph: represent all possible combinations



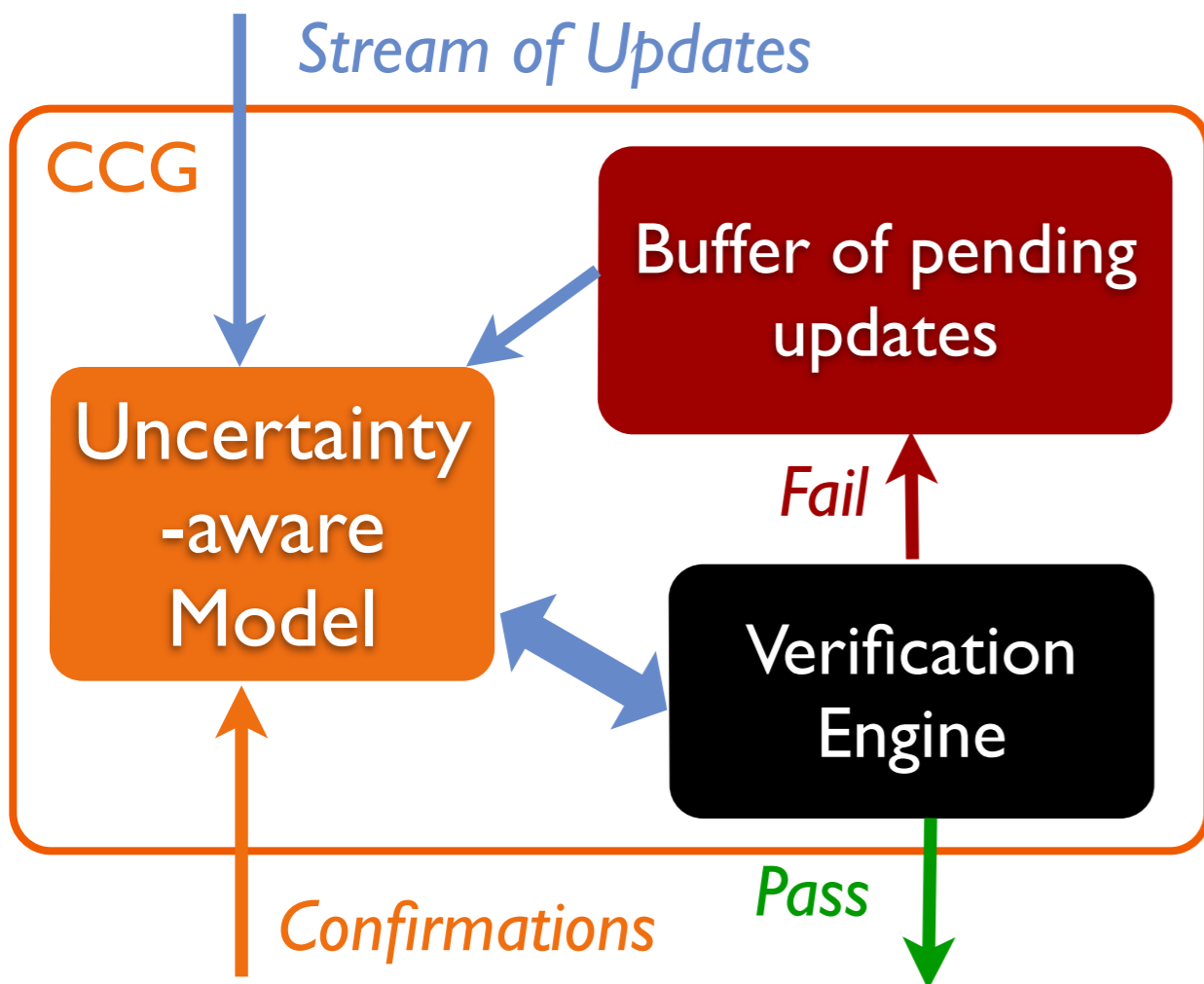
The model captures packets' view of the network, assuming controller initiates changes.

When to change “uncertain” to “certain”?

How to verify the network under “uncertainty”?

Consistency under Uncertainty

Enforcing consistency with max parallelism heuristically

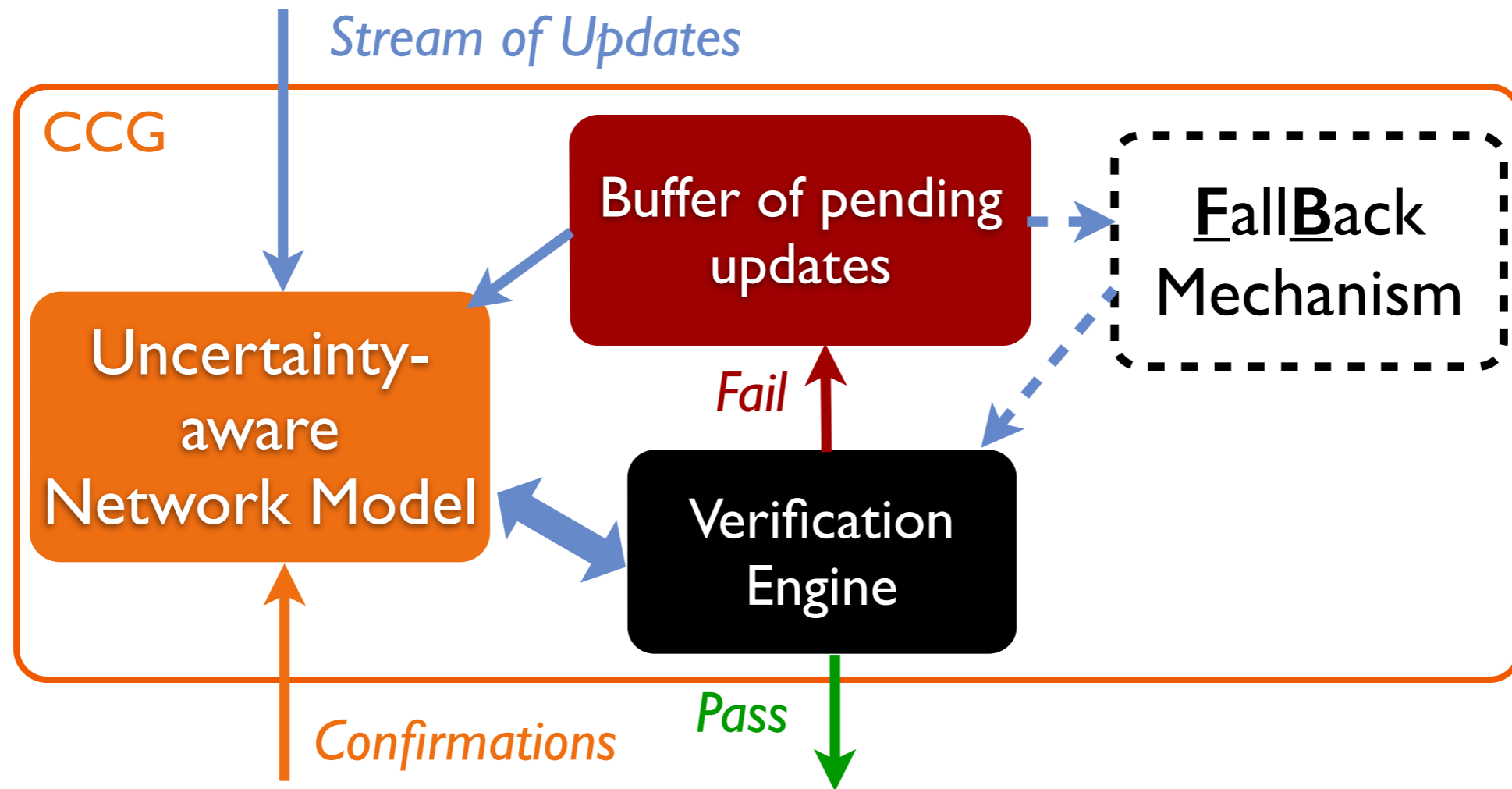


Waypoint Properties: flows are required to traverse a set of waypoints

- connectivity,
- waypointing,
- access control,
- service chaining, ...

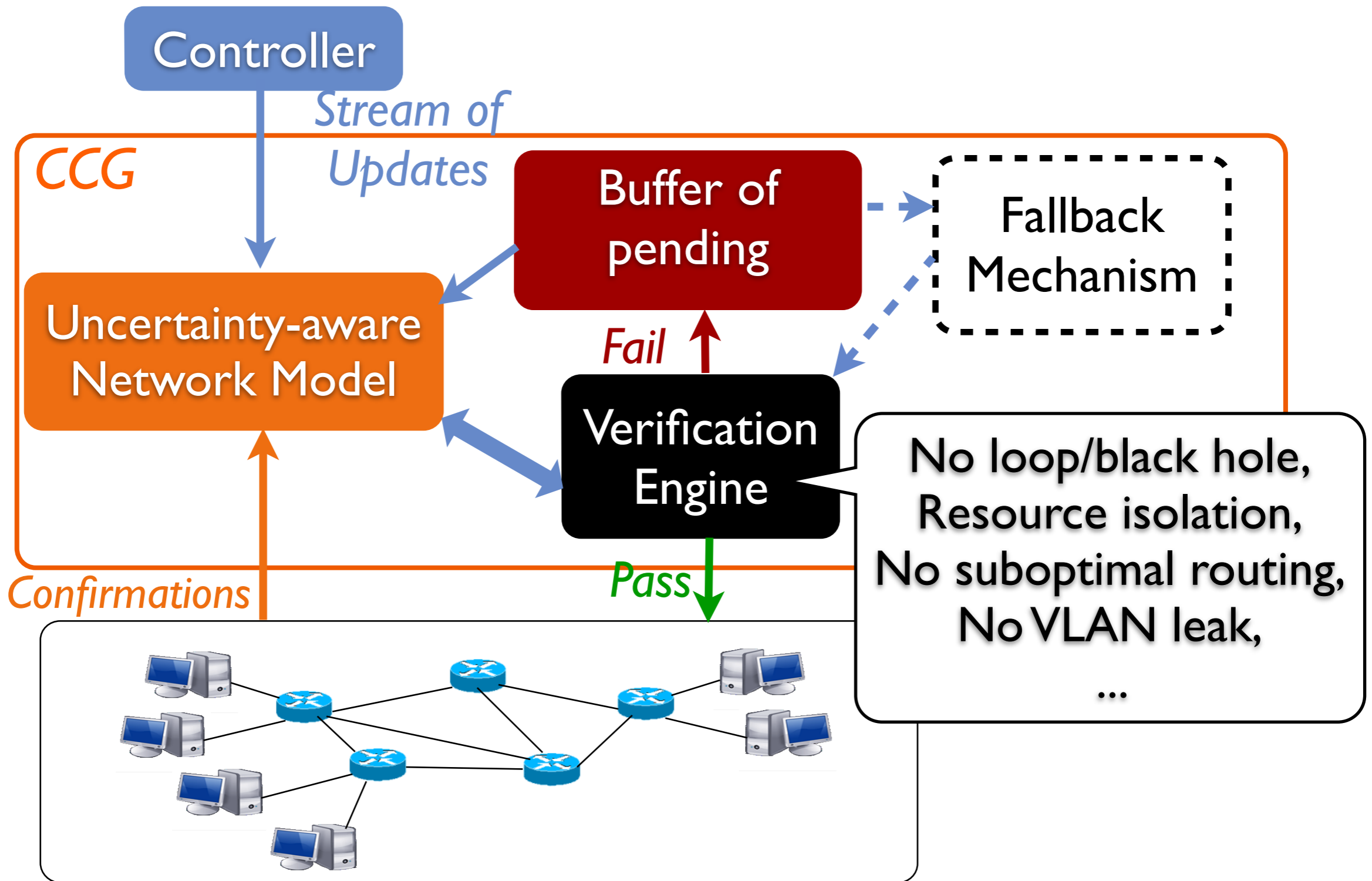
Theorem: *Segment independent* properties is guaranteed by the heuristic.

Consistency under Uncertainty



Even with FB triggered, CCG achieves better efficiency than using FB alone.

System Structure



Evaluation

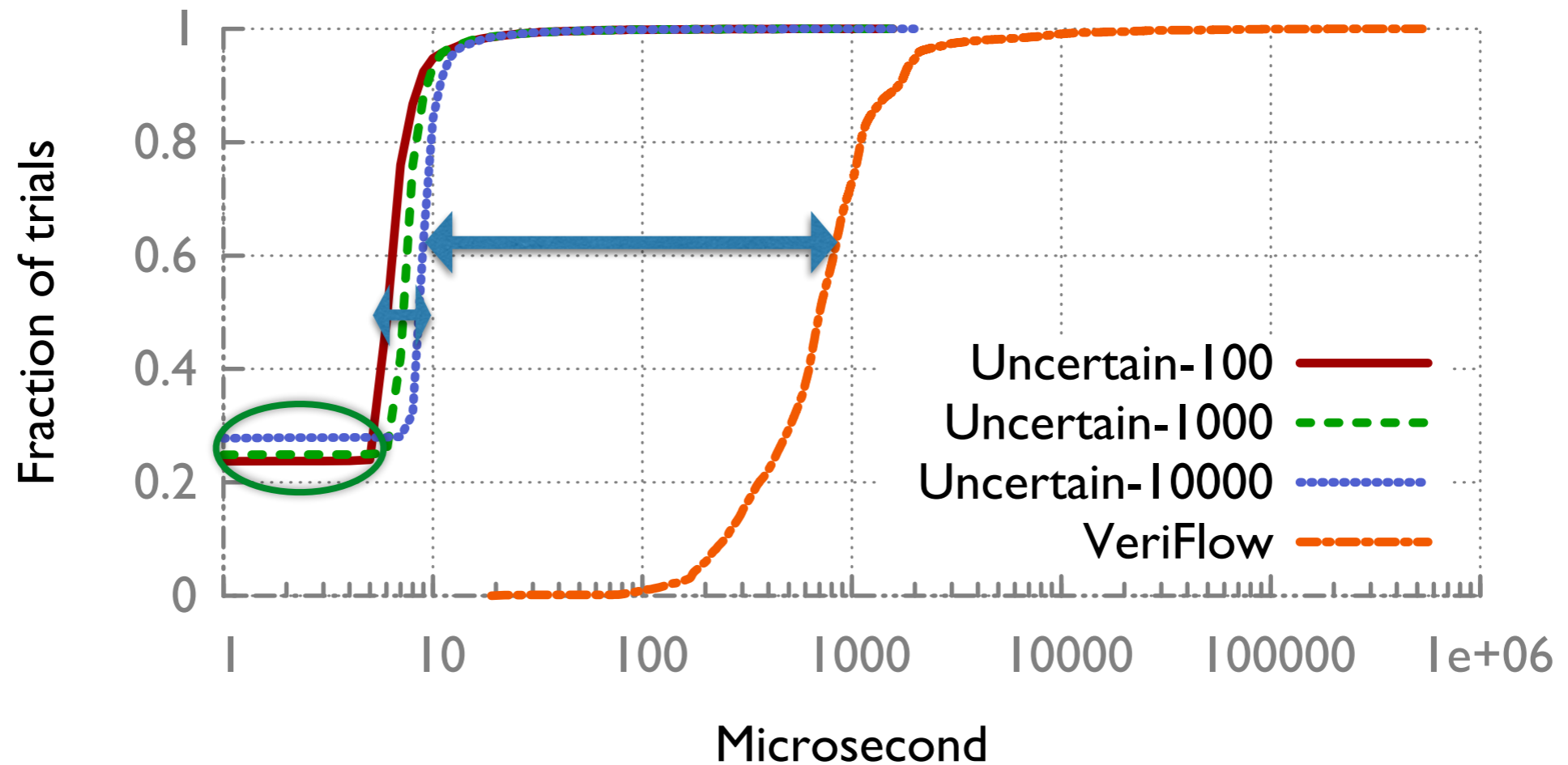
Can CCG verify network invariants in real time?

Can CCG achieve performance gain during network transitions with its algorithm for maximizing the parallelism of applying updates?

- Segment-independent Policies
- Non-segment-independent Policies

- Emulations
- Testbed experiments

Speed Analysis



15X less memory overhead (540MB vs. 9GB)

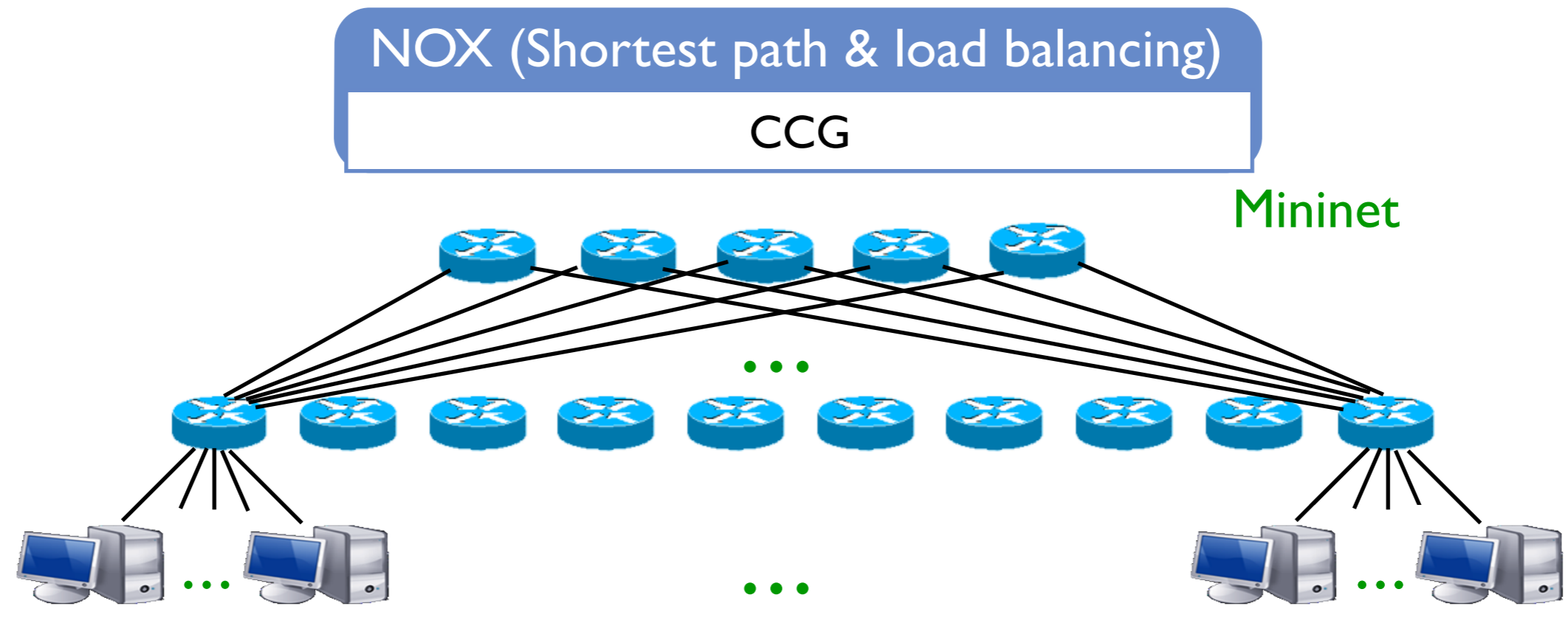
Simulated network: BGP RIBs and update trace from RouteViews injected into 172-router AS 1755 topology, checking reachability invariant

Emulation: Segment-independent Policies

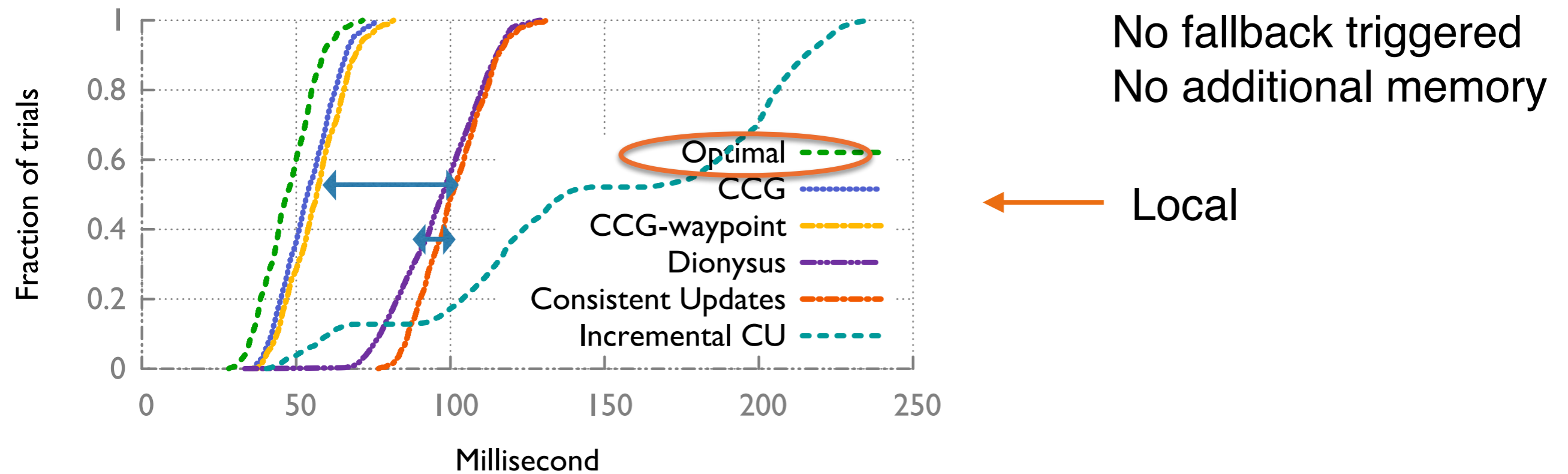
Controller-switch delay = network delay + processing delay

- Local (4ms)
- Wide area (100ms)

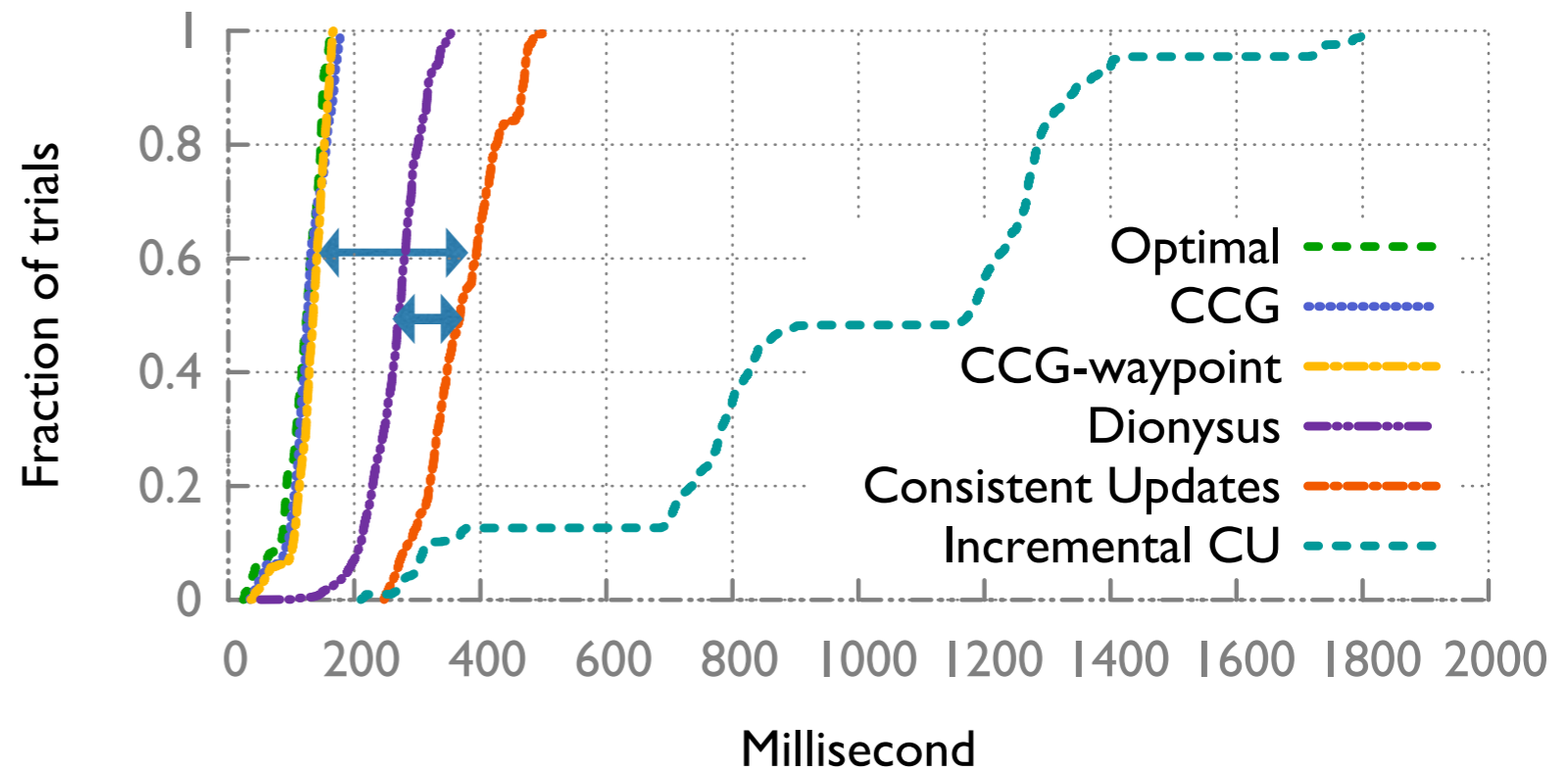
Measure: path completion time



Emulation: Segment-independent Policies



Wide area →



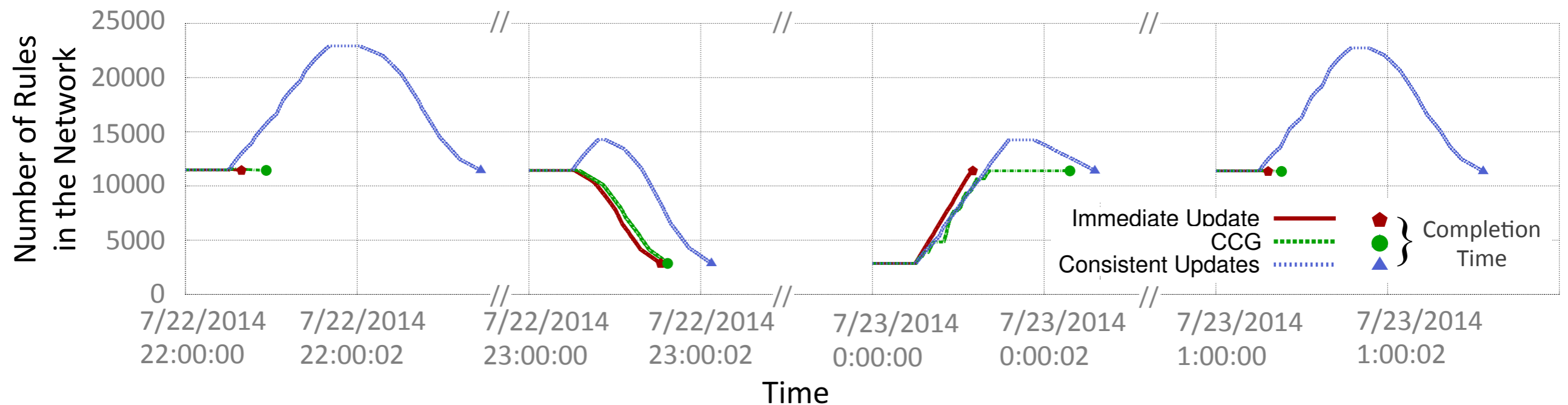
Emulation: Non-segment-independent Policies

Traces from an enterprise network with 200+ layer-3 devices.

One day, one snapshot per hour, 24 transitions, 4ms delay.

- New rules were added first, then old rules deleted.

Rules overlapped with longest prefix match, not segment-independent.



Fallbacks happened rarely.

Overhead close to Immediate Update, with no transient connectivity violations.

Conclusion

Uncertainty problem with network control

Uncertainty-aware network model

GCC, a system that

- enforces customizable network consistency properties with
- heuristically optimized efficiency.

Ongoing work:

- Study the generality of *segment independency*
- Test with more data traces, and compare against the original implementation of Dionysus
- Handle changes initiated from the network.