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,
- …
Network changes
• control applications,
• changes in traffic load,
• system upgrades,
• …
Network changes

- control applications,
- changes in traffic load,
- system upgrades,
- …
Old correct behavior

Next correct behavior
Old correct behavior

Next correct behavior
Old correct behavior

Next correct behavior
Old correct behavior

Next correct behavior
Old correct behavior

Next correct behavior
Old correct behavior

Next correct behavior
Towards a solution

1. Correctness at every step
Towards a solution

1. Correctness at every step
Towards a solution

1. Correctness at every step
2. With efficient update installation
Towards a solution

1. Correctness at every step
2. With efficient update installation

What is Correctness?
- firewall traversal,
- access control,
- balanced load,
- loop freedom,
- …
Towards a solution

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

What is Correctness?
- firewall traversal,
- access control,
- balanced load,
- loop freedom,
- …
Problem Statement

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

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

Is it possible to efficiently ensure customizable correctness properties as the network evolves?
Prior Work

Consistent Updates
Prior Work

Consistent Updates

Any property <= “packet coherence”
Prior Work

Consistent Updates

Any property $\leq$ “packet coherence"

Sacrifices efficiency
Prior Work

Dionysus
Prior Work

Dionysus

Dynamic scheduling
Prior Work

Consistent Updates

Dionysus

Dynamic scheduling
Prior Work

Dionysus

Dynamic scheduling

Customizes consistency for efficiency
Prior Work

Dionysus

Dynamic scheduling

Customizes consistency for efficiency

Specific algorithms for different properties
Ideally given an **arbitrary** set of properties, a sequence with **minimized** update overhead is produced.

Controller

*Stream of Updates*
Ideally given an **arbitrary** set of properties, a sequence with **minimized** update overhead is produced.

Controller

Stream of Updates

No loop, no black hole, Resource isolation, No suboptimal routing, ...

![Diagram showing network with stream of updates](image-url)
Ideally given an **arbitrary** set of properties, a sequence with **minimized** update overhead is produced.
Our design: **Customizable Consistency Generator**

- Controller
- Stream of Updates
- CCG
- Confirmations
Our design: **Customizable Consistency Generator**

Key insight:
Our design: **Customizable Consistency Generator**

Key insight: **Synthesis**

Controller \( \rightarrow \text{Stream of Updates} \)

\( CCG \)

\( \rightarrow \text{Confirmations} \)
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis**
- **Verification**

**Controller**

- Stream of Updates

**CCG**

- Confirmations
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** → **Verification**

- **Controller**
  - Stream of Updates

- **CCG**
  - Confirmations
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** → **Verification**

- **Controller**

- **CCG**

- **Stream of Updates**

- **Verifier**
  - **Verification Engine**
  - **Network Model**

- **Confirmations**
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** → **Verification**

- **Controller**

- **CCG**

- **Stream of Updates**

- **Verifier**
  - Verification Engine
  - Network Model

- **Confirmations**

- No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...

- Synthesis Verification
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** ➔ **Verification**

- **Controller** ➔ **Stream of Updates** ➔ **CCG**
  - **Update queue** ➔ **Safe?** ➔ **Verifier**
    - **Verification Engine** ➔ **Network Model**
      - **Confirmations** ➔ **No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...**

- **Controller** ➔ **No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...**
Our design: **Customizable Consistency Generator**

Key insight:

**Synthesis** → **Verification**

- Controller
- Stream of Updates

**CCG**

- Update queue
- Safe?
- Yes

**Verifier**

- Verification Engine
- Network Model
- Confirmations

- No loop/black hole
- Resource isolation
- No suboptimal routing
- No VLAN leak
- ...

Diagram of network connections and components.
Our design: **Customizable Consistency Generator**

Key insight:

![Diagram of Synthesis and Verification processes with CCG, Verification Engine, Network Model, and Confirmations]

- **Controller**
  - Stream of Updates
  - CCG
  - Update queue
  - Safe?
    - Yes
    - No
  - Verification Engine
  - Network Model
  - Confirmations

- **Verification**
  - No loop/black hole
  - Resource isolation
  - No suboptimal routing
  - No VLAN leak
  - ...

---

Synthesis ➔ Verification
Our design: **Customizable Consistency Generator**

Key insight:

- **Synthesis** → **Verification**

- **Controller**
  - Stream of Updates
  - **CCG**
  - Safe?
  - No

- **Update queue**
  - Yes

- **Verifier**
  - Verification Engine
  - Network Model
  - Confirmations

- **No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...**
Our design: **Customizable Consistency Generator**
Our design: **Customizable Consistency Generator**

![Diagram](image)

- **Controller**
- **Stream of Updates**
- **CCG**
- **Update queue**
- **Verifier**
  - **Verification Engine**
  - **Network Model**
- **Safe?**
  - No
  - Yes
- **Confirmations**

A should reach B

A → C → D → E → B
A → F → G → H
Our design: **Customizable Consistency Generator**

**Controller**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

**CCG**

- **Update queue**
- **Verifier**
  - **Verification Engine**
  - **Network Model**

**Stream of Updates**

1. 2 1 3 4

**Safe?**

- Yes
- No

**A should reach B**

**Confirmation**

1. C
2. D
3. E
4. F
5. G
6. H
7. B

A should reach B
Our design: **Customizable Consistency Generator**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

A should reach B
Our design: **Customizable Consistency Generator**

- **Controller**
  - 1. mod A->C to A->F
  - 2. add F->G
  - 3. add G->H
  - 4. add H->B

- **Update queue**
- **Safe?**
  - Yes
  - No

- **Verifier**
  - **Verification Engine**
  - **Network Model**

- **Stream of Updates**

- **CCG**

- **A should reach B**

- **Confirmations**

Diagram:
- A -> B
- C -> D
- E -> F
- G -> H
- A -> C
- F -> G
- G -> H
- H -> B
Our design: **Customizable Consistency Generator**

**Controller**
- **Update queue**
- **Safe?**
  - Yes
  - No

**Verifier**
- **Verification Engine**
- **Network Model**

**Stream of Updates**
1. mod A→C to A→F
2. add F→G
3. add G→H
4. add H→B

**A should reach B**

**Confirmation**

Diagram:
- **A** to **C**
- **A** to **D**
- **A** to **F**
- **C** to **D**
- **D** to **E**
- **E** to **B**
- **F** to **G**
- **G** to **H**
- **H** to **B**

Diagram shows a network model with nodes A, B, C, D, E, F, G, and H, connected with arrows indicating the direction of updates.
Our design: **Customizable Consistency Generator**

**CCG**
- Customizable Consistency Generator
- Stream of Updates
- Controller
- Update queue
- Safe?
- Verification Engine
- Network Model
- Confirmations

**A should reach B**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B
Our design: **Customizable Consistency Generator**

**Controller**
- Stream of Updates
  - 1. mod A→C to A→F
  - 2. add F→G
  - 3. add G→H
  - 4. add H→B

**CCG**
- Update queue
- Verification Engine
  - Network Model
  - A should reach B
- Confirmations

**Diagram**
- A → C → D → E → B
- F → G → H
- Safe? (Yes/No)

**Legend**
- Red for **Safe?**
- Blue for **Controller**
- CCG for **Customizable Consistency Generator**
Our design: **Customizable Consistency Generator**

- **Controller**
  - Stream of Updates
  - 1. mod A->C to A->F
  - 2. add F->G
  - 3. add G->H
  - 4. add H->B

**CCG**
- Update queue

**Verifier**
- Verification Engine
- Network Model

**Confirmation**
- A should reach B

Network Model:
- A
- B
- C
- D
- E
- F
- G
- H

Controller:
- Safe?
  - Yes
  - No
Our design: **Customizable Consistency Generator**

1. mod A→C to A→F
2. add F→G
3. add G→H
4. add H→B

Controller

Update queue

Verifier

Verification Engine

Network Model

A should reach B

Stream of Updates

CCG

Safe?

No

Yes

Confirmations

A
B
C
D
E
F
G
H
Our design: **Customizable Consistency Generator**

### Stream of Updates

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

### Verification Process

- **Controller**
- **CCG**
- **Update queue**
  - No → **Safe?**
    - Yes
    - No → **Verifier**
      - **Verification Engine**
      - **Network Model**

**A should reach B**

**Confirmations**

**Diagram**:

- A
- B
- C
- D
- E
- F
- G
- H
Our design: **Customizable Consistency Generator**

**Controller**

- **CCG**
- **Update queue**
- **Safe?**
- **Verifications Engine**
- **Network Model**
- **Confirmations**

**Stream of Updates**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

A should reach B

Graph:

- A -> C
- C -> D
- D -> E
- E -> B
- F -> G
- G -> H
- H -> B
Our design: **Customizable Consistency Generator**

![Diagram of Consistency Generator]

- **Controller**
  - Stream of Updates
  - CCG
  - Update queue
  - Safe?
    - No
    - Confirmations
    - 1. mod A->C to A->F
    - 2. add F->G
    - 3. add G->H
    - 4. add H->B
  - Yes

- **Verifier**
  - Verification Engine
  - Network Model

A should reach B
Our design: **Customizable Consistency Generator**

- **Controller**
  - Stream of Updates
  - CCG
  - Update queue
  - Safe?
  - No
  - A should reach B
  - Yes
  - Confirmations

- **Verifier**
  - Verification Engine
  - Network Model

- **A should reach B**

- **Update Queue Updates**
  1. mod A->C to A->F
  2. add F->G
  3. add G->H
  4. add H->B
Our design: **Customizable Consistency Generator**

1. **Controller**
   - **Stream of Updates**

2. **CCG**
   - **Update queue**

3. **Verifier**
   - **Verification Engine**
   - **Network Model**

4. **Confirmation**

A should reach B

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mod A-&gt;C to A-&gt;F</td>
</tr>
<tr>
<td>2</td>
<td>add F-&gt;G</td>
</tr>
<tr>
<td>3</td>
<td>add G-&gt;H</td>
</tr>
<tr>
<td>4</td>
<td>add H-&gt;B</td>
</tr>
</tbody>
</table>

Diagram:

- A
- B
- C
- D
- E
- F
- G
- H

Connections:
- A -> C
- C -> D
- D -> E
- E -> B
- F -> G
- G -> H
- H -> F
Our design: **Customizable Consistency Generator**

1. **Controller**
   - **CCG**
   - **Update queue**
   - **Verifier**
     - **Verification Engine**
     - **Network Model**

2. **Stream of Updates**
   - **Safe?**
     - **Yes**
     - **No**

3. **Confirmations**

**A should reach B**

Legend:
- **mod A->C to A->F**
- **add F->G**
- **add G->H**
- **add H->B**

Diagram:
- Node **A**
- Node **B**
- Node **C**
- Node **D**
- Node **E**
- Node **F**
- Node **G**
- Node **H**

**Steps**:
1. **mod A->C to A->F**
2. **add F->G**
3. **add G->H**
4. **add H->B**
Our design: **Customizable Consistency Generator**

---

- **Controller**
- **Update queue**
- **Verifier**
  - **Verification Engine**
  - **Network Model**

**Stream of Updates**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

---

**A should reach B**

<table>
<thead>
<tr>
<th>Step</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mod A-&gt;C to A-&gt;F</td>
</tr>
<tr>
<td>2</td>
<td>add F-&gt;G</td>
</tr>
<tr>
<td>3</td>
<td>add G-&gt;H</td>
</tr>
<tr>
<td>4</td>
<td>add H-&gt;B</td>
</tr>
</tbody>
</table>

---

- **CCG**
- **Safe?**
  - Yes
  - No

---

**Diagram:**

- A
- B
- C
- D
- E
- F
- G
- H
Our design: **Customizable Consistency Generator**

1. **Controller**
   - Stream of Updates
     1. Safe?
     2. Yes
     3. No

   - **Update queue**

   - **Verifier**
     - **Verification Engine**
     - **Network Model**

   - **Confirmations**

   - **CCG**

   - **A should reach B**

   - **Modifications**:
     1. mod A->C to A->F
     2. add F->G
     3. add G->H
     4. add H->B

   - **Steps**:
     2. Add connection from C to F.
     3. Add connection from G to H.
     4. Add connection from H to B.
Our design: **Customizable Consistency Generator**

- **Controller**
  - **Update queue**
  - **Safe?**
    - **Yes**
    - **No**

- **Verifier**
  - **Verification Engine**
  - **Network Model**

- **Stream of Updates**
  - 1. mod A->C to A->F
  - 2. add F->G
  - 3. add G->H
  - 4. add H->B

- **Path**
  - A to C
  - C to D
  - D to E
  - E to F
  - F to G
  - G to H
  - H to B

- **A should reach B**
Our design: **Customizable Consistency Generator**

Controller

1. **CCG**
2. Stream of Updates

1. Update queue
2. Safe?
3. Yes
4. No

Verifier

Verification Engine

Network Model

Confirmations

A should reach B

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

Diagram:

A -> C -> D -> E -> B

F -> G -> H
Our design: Customizable Consistency Generator

![Diagram of the Consistency Generator](image)

- **Controller**
- **CCG**
  - **Update queue**
  - **Safe?**
  - **No**
  - **Yes**

**Verifier**
- **Verification Engine**
- **Network Model**
- **Confirmations**

- **A should reach B**

**Stream of Updates**
- 1. mod A->C to A->F
- 2. add F->G
- 3. add G->H
- 4. add H->B
Our design: **Customizable Consistency Generator**

- **CCG (Consistency Generator)**
  - Stream of Updates
  - Update queue
  - Controller
    - Safe?
      - Yes
      - No

- **Verifier**
  - Verification Engine
  - Network Model
    - Confirmations

- **Network Diagram**
  - A -> C to A -> F
  - add F -> G
  - add G -> H
  - add H -> B

- A should reach B
Our design: **Customizable Consistency Generator**

1. **Controller**
   - Stream of Updates
   - **CCG**
     - Update queue
     - **Verifier**
       - Verification Engine
       - Network Model
     - **Safe?**
       - Yes
       - No

2. **A should reach B**

- **Confirmations**
  - 1. mod A->C to A->F
  - 2. add F->G
  - 3. add G->H
  - 4. add H->B

Diagram:
- A -> C -> D
- C -> E
- E -> B
- F -> G
- G -> H
- H -> E

Controller decides:
- **Safe?**
  - Yes
  - No
Our design: **Customizable Consistency Generator**

- **Controller**
  - Stream of Updates
  - **Update queue**
  - **Safe?**
    - Yes
    - No

- **Verifier**
  - Verification Engine
  - Network Model
  - Confirmations

1. mod A→C to A→F
2. add F→G
3. add G→H
4. add H→B

A should reach B

Diagram:

- Nodes: A, B, C, D, E, F, G, H

- CCG:
  - 1. mod A→C to A→F
  - 2. add F→G
  - 3. add G→H
  - 4. add H→B
Our design: **Customizable Consistency Generator**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

A should reach B
Our design: **Customizable Consistency Generator**

1. **Controller**
   - **Update queue**
   - **Safe?**

2. **CCG**
   - Stream of Updates
   - **Verifier**
     - Verification Engine
     - Network Model
     - Confirmations

3. **A should reach B**

4. **Confirmation**
   - mod A->C to A->F
   - add F->G
   - add G->H
   - add H->B

Diagram:
- Nodes: A, B, C, D, E, F, G, H
Our design: **Customizable Consistency Generator**

Controller → Update queue → **Verifier**

**C**ustomizable **C**onsistency **G**enerator

- Stream of Updates
- Verification Engine
- Network Model
- Confirmations

1. mod A→C to A→F
2. add F→G
3. add G→H
4. add H→B

A should reach B

---

- A
- B
- C
- D
- E
- F
- G
- H

Controller

Update queue

Safe?

Yes

No
Our design: **Customizable Consistency Generator**

**Controller**

- Stream of Updates
- **CCG**
  - Update queue
  - **Verifier**
    - Verification Engine
    - **Network Model**
      - **Confirmations**

- **Safe?**
  - Yes
  - No

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B

A should reach B

Diagram: Network with nodes A, B, C, D, E, F, G, H, showing a stream of updates and the verification process.
Our design: **Customizable Consistency Generator**

1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B
Our design: **Customizable Consistency Generator**

```
1. mod A->C to A->F
2. add F->G
3. add G->H
4. add H->B
```

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

A should reach B

A

C

D

E

F

G

H

B

Yes

No

Safe?
Our design: **Customizable Consistency Generator**

Enforcing consistency with heuristically maximized parallelism.
Challenges

CCG

Stream of Updates

Update queue

Verifier

Verification Engine

Network Model

Confirmation

Safe?

No

Yes
Challenges

1. Distributed nature of networks (uncertainty)
Challenges

1. Distributed nature of networks (uncertainty)
2. Greedy algorithm may get stuck
Network Uncertainty

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

Remove rule 1 (delayed)
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.
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.
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.
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.

Possible network states:
**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.
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.

Possible network states:
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.
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

Network states as forwarding graphs
Uncertainty-aware Modeling

Network states as forwarding graphs

Naively, represent every possible network state $O(2^n)$
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph

When to change “uncertain” to “certain”?
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph

When to change “uncertain” to “certain”?
Uncertainty-aware Modeling

Network states as forwarding graphs

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

Uncertain graph: Collapse all possible states onto one graph

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

When to change “uncertain” to “certain”?
Verification under Uncertainty

Verification: Traversal on forwarding graphs
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

• No certain outgoing link → a possible black-hole
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

- No certain outgoing link a possible black-hole

By traversing the graph once,
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

- No certain outgoing link → a possible black-hole

By traversing the graph once,

- perform a simultaneous traversal of all possibilities, and thus
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

- No certain outgoing link a possible black-hole

By traversing the graph once,

- perform a simultaneous traversal of all possibilities, and thus
- reason about the network state correctly in the presence of uncertainty
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

• No certain outgoing link \(\Rightarrow\) a possible black-hole

By traversing the graph once,

• perform a simultaneous traversal of all possibilities, and thus
• reason about the network state correctly in the presence of uncertainty

Implementation Optimizations
Verification under Uncertainty

Verification: Traversal on forwarding graphs

Take uncertainty into account:

- No certain outgoing link \(\rightarrow\) a possible black-hole

By traversing the graph once,

- perform a simultaneous traversal of all possibilities, and thus
- reason about the network state correctly in the presence of uncertainty

Implementation Optimizations
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

No

Safe?

Yes

Confirmations
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

No

Safe?

Yes

Verifier

Verification Engine

Network Model

Confirmations

A should reach E via Firewall F

A should reach E via Firewall F
Consistency under Uncertainty — Problem

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

Controller

Stream of Updates

Verifier

Verification Engine

Network Model

A should reach E via Firewall F
Consistency under Uncertainty — Problem

Controller

CCG

Update queue

Verifier

Verification Engine

Network Model

A should reach E via Firewall F

A should reach E via Firewall F

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E
Consistency under Uncertainty — Problem

CCG

Stream of Updates

Verifier

Verification Engine

Network Model

Controller

Update queue

No

Safe?

Yes

Confirmations

A should reach E via Firewall F

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A
B
F
D
E
Consistency under Uncertainty — Problem

Stream of Updates:
- 1: mod A->D to A->B
- 2: add D->F
- 3: mod F->D to F->B
- 4: add B->E

A should reach E via Firewall F

CCG

Controller

Update queue

Verifier

Verification
Engine

Network Model

Confirmation:

A

B

F

D

E

Safe?

No

Yes

A should reach E via Firewall F
Consistency under Uncertainty — Problem

CCG

Controller

Stream of Updates

Update queue

Verifier

Verification Engine

Network Model

A should reach E via Firewall F

A

F

E

D

B

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

Confirmations

Yes

No
Consistency under Uncertainty — Problem

Controller

CCG

Update queue

Verifier

Verification Engine

Network Model

Stream of Updates

A should reach E via Firewall F

1. mod A→D to A→B
2. add D→F
3. mod F→D to F→B
4. add B→E

A

B

F

D

E

Consensus under Uncertainty — Problem
Consistency under Uncertainty — Problem

CCG

Update queue

Verifier

Verification Engine

Network Model

Controller

Safe?

No

Yes

Stream of Updates

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A should reach E via Firewall F
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

No

Safe?

Yes

Confirmations

1 mod A->D to A->B
2 add D->F
3 mod F->D to F->B
4 add B->E

A should reach E via Firewall F

A should reach E via Firewall F
Consistency under Uncertainty — Problem

Stream of Updates

Controller

CCG

Update queue

Verifier

Verification Engine

Network Model

No

Safe?

Yes

Confirmations

mod A→D to A→B

add D→F

mod F→D to F→B

add B→E

A should reach E via Firewall F

A should reach E via Firewall F

A

B

D

F

E
Consistency under Uncertainty — Problem

Stream of Updates

Controller

CCG

Update queue

Verifier

Verification Engine

Network Model

Confirmations

Controller

1. mod A→D to A→B
2. add D→F
3. mod F→D to F→B
4. add B→E

A should reach E via Firewall F

A should reach E via Firewall F
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

Safe?

Confimations

A should reach E via Firewall F

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A should reach E via Firewall F
Consistency under Uncertainty — Problem

Controller

Update queue

Safe?

Verifier

Verification Engine

Network Model

Stream of Updates

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A should reach E via Firewall F

CCG

No

Yes

Confirmations
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

Yes

Confirmsions

1  mod A->D to A->B
2  add D->F
3  mod F->D to F->B
4  add B->E

A should reach E via Firewall F

A should reach E via Firewall F
Consistency under Uncertainty — Problem

**Controller**

- **CCG**
  - **Update queue**
    - **Safe?**
      - **Yes**
      - **Verifier**
        - **Verification Engine**
          - **Network Model**
            - **Confirmations**
              - **A should reach E via Firewall F**

**Stream of Updates**

1. **mod A->D to A->B**
2. **add D->F**
3. **mod F->D to F->B**
4. **add B->E**

**Diagram**

- **A**
- **B**
- **C**
- **D**
- **E**
- **F**

- A should reach E via Firewall F
Consistency under Uncertainty — Problem

Controller

Update queue

Verifier

Stream of Updates

Network Model

Verification Engine

Confirmanations

A should reach E via Firewall F

1 mod A->D to A->B
2 add D->F
3 mod F->D to F->B
4 add B->E

CCG

Safe?

Yes

No

A

E

B

F

D

2 mod A->D to A->B

3 add D->F

4 mod F->D to F->B

5 add B->E
Consistency under Uncertainty — Problem

Controller

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

A should reach E via Firewall F

A

B

F

E

D

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E
Consistency under Uncertainty — Problem

**Controller**

1. Stream of Updates
   1. mod A->D to A->B
   2. add D->F
   3. mod F->D to F->B
   4. add B->E

**CCG**

- Update queue
- No

**Verifier**

- Verification Engine
- Network Model

**A should reach E via Firewall F**

**Safe?**

- Yes
- No

**Confirmations**

- A should reach E via Firewall F
Consistency under Uncertainty — Problem

CCG
Update queue
Verifier
Verification Engine
Network Model

Stream of Updates

Controller

Safe?

A should reach E via Firewall F

Yes
No

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A
B
C
D
E

10
20
30
40
Consistency under Uncertainty — Problem

Stream of Updates

Controller

CCG

Update queue

Safe?

Verifier

Verification Engine

Network Model

Confirmations

A should reach E via Firewall F

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

104x685 to 809x715
296x88 to 975x489
324x545 to 451x572
890x429 to 1193x730
Consistency under Uncertainty — Problem

Stream of Updates

Controller

CCG

Update queue

Verifier

Verification Engine

Network Model

Confirmations

A should reach E via Firewall F

1. mod A->D to A->B
2. add D->F
3. mod F->D to F->B
4. add B->E

A

B

F

D

E
Consistency under Uncertainty — Problem

Controller

Stream of Updates

Verifier

Verification Engine

Network Model

CCG

Update queue

Yes

No

Safe?

Confirmations

A should reach E via Firewall F

1 mod A->D to A->B
2 add D->F
3 mod F->D to F->B
4 add B->E

A should reach E via Firewall F
Consistency under Uncertainty — Solution

Stream of Updates

CCG

Update queue

No

Safe?

Yes

Verifier

Verification Engine

Network Model

Confirmations
Consistency under Uncertainty — Solution

CCG

Stream of Updates

Update queue

Verifier

Verification Engine

Network Model

Confirmations

Safe?

No

Yes

No, and stuck!
Consistency under Uncertainty — Solution

Stream of Updates

CCG

Verifier

Network Model

Confirmation

Update queue

Safe?

Yes

No

Fallback Mechanism

No, and stuck!
Consistency under Uncertainty — Solution

- **CCG**
  - Fallback Mechanism
  - No, and stuck!

- **Stream of Updates**
  - Update queue
  - Safe?
    - Yes
    - No
      - Fallback Mechanism

- **Verifier**
  - Verification Engine
  - Network Model
  - Confirmations
Consistency under Uncertainty — Solution

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

Stream of Updates

CCG

Verifier

Network Model

Update queue

Verification Engine

Safe?

Confirmation

No

No, and stuck!

Yes

Fallback Mechanism
Consistency under Uncertainty

Stream of Updates

CCG

Update queue

Verifier

Verification Engine

Network Model

Safe?

Yes

No

Confirmations
Consistency under Uncertainty

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

- connectivity,
- waypointing,
- access control,
- service chaining, …
Consistency under Uncertainty

Waypoint Properties: flows are required to traverse a set of waypoints
- connectivity,
- waypointing,
- access control,
- service chaining, ...

Theorem: Segment independent properties are guaranteed to complete using greedy updates.
System Structure

CCG

Stream of Updates

Controller

Update queue

Verifier

Verification Engine

Network Model

Safe?

Yes

No

No, cannot

Fallback Mechanism

Confirmation
System Structure

- **Controller**
  - Stream of Updates
  - Verifier
    - Verification Engine
    - Network Model
  - Confirmation

**CCG**
- Fallback Mechanism
  - No, cannot

**Update queue**
- Safe?
  - Yes
  - No

- **No loop/black hole, Resource isolation, No suboptimal routing, No VLAN leak, ...**
System Structure

**CCG**
- Fallback Mechanism

**Controller**
- Stream of Updates
- No loop/black hole
- Resource isolation
- No suboptimal routing
- No VLAN leak

**Verifier**
- Verification Engine
- Network Model
- Confirmations

**Update queue**
- Safe?

**Consistent Updates, SWAN, ...**

**No**
- No, cannot
Evaluation

Can CCG improve performance over prior work?

• Segment-independent Properties

• Non-segment-independent Properties

Evaluation approaches:

• Emulations

• Testbed experiments
Emulation: Segment-independent Properties

NOX (Shortest path & load balancing)

CCG

Mininet
Emulation: Segment-independent Properties

NOX (Shortest path & load balancing)
CCG

Mininet
Emulation: Segment-independent Properties

![Diagram of network simulation with NOX and Mininet]
Emulation: Segment-independent Properties

Controller-switch delay = network delay + processing delay
Emulation: Segment-independent Properties

Controller-switch delay = network delay + processing delay

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

Measure: path completion time

Controller-switch delay = network delay + processing delay

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

Measure: path completion time
Emulation: Segment-independent Properties

![Graph showing fraction of trials vs milliseconds]
Emulation: Segment-independent Properties

Fraction of trials vs. Milliseconds
Emulation: Segment-independent Properties

Fraction of trials vs. Milliseconds graph showing different curves indicating CCG-connectivity and an Optimal curve.
Emulation: Segment-independent Properties
Emulation: Segment-independent Properties

- **Optimal CCG-connectivity**
- **Consistent Updates**
- **CCG-waypoint**

Fraction of trials vs. Milliseconds
Emulation: Segment-independent Properties

Fraction of trials

Milliseconds

Optimal CCG-connectivity

Consistent Updates

Incremental Consistent Updates

CCG-waypoint
Emulation: Segment-independent Properties

- **Optimal CCG-connectivity**
- **CCG-waypoint**
- **Dionysus-pkt-coherence**
- **Consistent Updates**
- **Incremental Consistent Updates**
Emulation: Segment-independent Properties

- Optimal CCG-connectivity
- CCG-waypoint
- Consistent Updates
- Dionysus-pkt-coherence
- Incremental Consistent Updates
Emulation: Segment-independent Properties
Emulation: Segment-independent Properties
Emulation: Segment-independent Properties

Fraction of trials vs. Milliseconds
Emulation: Segment-independent Properties

- No fallback triggered
- No additional memory

Local
Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds
Emulation: Segment-independent Properties

No fallback triggered
No additional memory

Local

Wide area

Fraction of trials

Milliseconds

<table>
<thead>
<tr>
<th>Fraction of trials</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
</tr>
<tr>
<td>0.4</td>
</tr>
<tr>
<td>0.6</td>
</tr>
<tr>
<td>0.8</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

Milliseconds

<table>
<thead>
<tr>
<th>0</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
<th>250</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
<td>400</td>
<td>600</td>
<td>800</td>
<td>1000</td>
</tr>
</tbody>
</table>

Optimal
CCG
CCG-waypoint
Dionysus-pkt-coherence
Consistent Updates
Incremental CU
Emulation: Non-segment-independent Properties

Traces from a operational 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.

Properties: Black hole freedom + Loop freedom

- Rules overlapped with longest prefix match, not segment-independent.
Emulation: Non-segment-independent Properties

Traces from a operational 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.

Properties: Black hole freedom + Loop freedom

- Rules overlapped with longest prefix match, not segment-independent.
Emulation: Non-segment-independent Properties

Traces from a operational 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.

Properties: Black hole freedom + Loop freedom

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

- Fallbacks happened rarely.

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

CCG, a system that
Conclusion

CCG, a system that

- enforces customizable network consistency properties with
Conclusion

CCG, a system that

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

CCG, a system that

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

CCG, a system that

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

Ongoing work:
Conclusion

CCG, a system that

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

Ongoing work:

- Study generality of segment independency
Conclusion

CCG, a system that

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

Ongoing work:

- Study generality of segment independency
- Handle network-initiated changes
Conclusion

CCG, a system that

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

Ongoing work:

- Study generality of segment independency
- Handle network-initiated changes
- ...

Conclusion

CCG, a system that

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

Ongoing work:

- Study generality of segment independency
- Handle network-initiated changes
- …