Towards Decentralized Fast Consistent Updates

MARCO CHIESA
UNIVERSITÉ CATHOLIQUE DE LOUVAIN

JOINT WORK WITH:
THANH DANG NGUYEN (UC LOUVAIN),
MARCO CANINI (UC LOUVAIN)
Updating the network configuration

A fundamental network operations performed whenever:

- network policy changes
- network devices fail
- traffic load changes
- security attacks
Updating the network configuration

A fundamental network operations performed whenever:

- network policy changes
- network devices fail
- traffic load changes
- security attacks
Updating the network configuration

A fundamental network operations performed whenever:

- network policy changes  —— planned updates
- network devices fail
- traffic load changes
- network attacks  —— unplanned updates
Unplanned network update scenario

network controller

current routing

protection routing

new routing
Unplanned network update scenario

- Current routing
- Network controller
- Protection routing
- New routing
- Suboptimal network event
Unplanned network update scenario

Current routing

Protection routing

Network controller computes new routing

Suboptimal

Optimal
Desiderata: **fast network update** to the new configuration

- Current routing
- Protection routing
- New routing

*Network controller*

*Suboptimal* → *fast update wanted* → *optimal*
One-shot updates are **dangerous**

The network is a distributed system

Asynchronous update messages

current

```
1 ——— 2 ——— 3
```

target

```
1 ——— 2 ——— 3
```
One-shot updates are **dangerous**

The network is a distributed system

Asynchronous update messages

If Switch 2 updates BLUE before Switch 5 installs the forwarding rule for BLUE, traffic is blackholed!!
Flow dependencies must be met to avoid link congestion

A simple network:
- Each link capacity is 10
- Each flow requires 5 units

Goal: update from current to target
- GREEN and BLUE are updated

If Switch 1 updates BLUE before GREEN, link 1-2 is congested!!
Flow dependencies must be met to avoid link congestion

A simple network:
- Each link capacity is 10
- Each flow requires 5 units

Goal: update from current to target
- **GREEN** and **BLUE** are updated

If Switch 1 updates BLUE before GREEN, link 1-2 is congested!!
Most relevant related work

Dyonisus: centralized synchronization scheduling computation

Synchronization between switches and controller slows down the update

Multiple rounds depending on flow dependencies
Most relevant related work

**Dyonisus**: centralized synchronization scheduling computation

- Synchronization between switches and controller slows down the update

**TIME4**: one-shot update by means of clock synchronization

- Inaccuracy in clock synchronization leads to anomalies
Decentralized fast network update
Ez-segway

Decentralized fast network update

**Key idea**
Move simple, yet powerful, logic from the controller to the switches
Ez-segway
Separation of concerns

Central controller role:
• detects flow dependencies
• computes flows partial ordering
• sends ordering to the switches
Ez-segway
Separation of concerns

Central controller role:
• detects flow dependencies
• computes flows partial ordering
• sends ordering to the switches

Switches role:
• combine local and global (pre-computed) information to perform the update
• coordinate with neighbors
Central controller role:
1. detects flow dependencies
   • constructs a dependency graph

Ez-segway
the controller perspective
Central controller role: 1. Detects flow dependencies
The dependency graph

Central controller role: 1. Detects flow dependencies

Current:

Target:

1,4 5
4,2 5
1,2 0
Central controller role: 1. Detects flow dependencies

The dependency graph

**Current**

1 -> 2 -> 3

**Target**

1 -> 2 -> 3

1,4 5  4,2 5  1,2 0

RED:5  GREEN:5  BLUE:5
Central controller role: 1. Detects flow dependencies

The dependency graph

current

1
2
3

1,4 5

4,2 5

1,2 0

4
5

5

5

target

1
2
3

RED:5
GREEN:5
BLUE:5
Central controller role: 1. Detects flow dependencies

The dependency graph

**current**

**target**

- 1. Detects flow dependencies
Central controller role: 1. Detects flow dependencies

The dependency graph

Current

Target

Congestion if performed

1, 4, 5
4, 2, 5
1, 2, 0
Central controller role: 1. Detects flow dependencies

The dependency graph

current

1 2 3

4 5

target

1 2 3

4 5

RED:5
GREEN:5
BLUE:5

congestion-free operation

1,4 5
4,2 5
Ez-segway
the controller perspective

Central controller role:
1. detects flow dependencies
   - constructs a dependency graph
2. computes flows *partial* ordering
   - assign priorities to flows
Assigning flow priorities

Central controller role: 2. computes flow partial ordering

Current

Target

GREY:5
RED:5
GREEN:5
BLUE:5
Assigning flow priorities

Central controller role: 2. computes flow partial ordering

current

target

GREY: 5
RED: 5
GREEN: 5
BLUE: 5

Update GREY first...
Assigning priorities

Central controller role: 2. computes flow partial ordering

Update GREY first... ...leads to link congestion
Assignign priorities

Central controller role: 2. computes flow partial ordering

current

```
Assignign priorities

Central controller role: 2. computes flow partial ordering

current

```

target

```
Assignign priorities

Central controller role: 2. computes flow partial ordering

current

```

Update GREEN first...
Assign priorities

Central controller role: 2. computes flow partial ordering

current

1
2
3

4
5

target

1
2
3

4
5

GREY:5
RED:5
GREEN:5
BLUE:5

1,4 0->5
2,4 0->5
1,2 5->0

Update GREEN first...
...BLUE second...

2. computes flow partial ordering
Assigning priorities

Central controller role: 2. computes flow partial ordering

current

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

target

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

GREEN:5  RED:5  BLUE:5

Update GREEN first... ...BLUE second... then GREY
Assigning flow priorities

Central controller role: 2. computes flow partial ordering

GREY should be given lower priority
Ez-segway: the controller perspective

Central controller role:
1. detects flow dependencies
   • constructs a dependency graph
2. computes flows *partial* ordering
   • assign priorities to flows
3. sends scheduling to the switches
   • only once
Ez-segway: the switches perspective

Switches role:
• coordinate with neighbors
In-band message passing

Switch role: coordinate with neighbors

Current

Target

New path installed upwards
In-band message passing

Switch role: coordinate with neighbors

New path installed upwards
In-band message passing

Switch role: coordinate with neighbors

New path installed upwards

current

1
2
3
4
5

target

1
2
3
4
5
In-band message passing

Switch role: coordinate with neighbors

New path installed upwards
Packets are routed on the new path
In-band message passing

Switch role: coordinate with neighbors

Current:

New path installed upwards
Packets are routed on the new path
Old path removed downwards

Target:
In-band message passing

Switch role: coordinate with neighbors

current

New path installed upwards
Packets are routed on the new path
Old path removed downwards

target
In-band message passing

Switch role: coordinate with neighbors

Current path:
- New path installed upwards
- Packets are routed on the new path
- Old path removed downwards

Target path:
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths

Parallelizable segment updates
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths

Parallelizable segment updates
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths
Parallelizable segment updates
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths
Parallelizable segment updates
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths

Parallelizable segment updates
Update speed up via segmentation

Switch role: coordinate with neighbors

Switch 2 is traversed in both paths

Parallelizable segment updates
Ez-segway: the switches perspective

Switches role:
• coordinate with neighbors
• combine local and global (pre-computed) information to perform the update
  • perform an update operations only if
    • i) there is enough spare capacity
    • ii) the update operations will not prevent any higher-priority update that is still not executable
Enforcing priorities

Switch 4 can move both GREEN and GREY. It first moves GREEN since it has higher priority than GREY.
Large-scale simulations

ez-segway: switches coordinate the update
centralized: controller coordinates the update

Measure: total update time

Setting:
• 6 real topologies from RocketFuel
• link capacities: 1…100 Gbps
• controller placed at centroid
• gravity traffic model
• shortest-path-via-random-node
• updates triggered by link failures
• 10 executions per topology
Update time comparison [ms]

Completion time reduced by 15%-50%
Summary

ezSegway design

• **Control plane** computes flows partial ordering
• **Data plane** coordinates the update

Better performance: Speeding up the update (up to 2x)

Ongoing work:
• Mininet evaluation
• Feasibility check on Centec switch
• Formalization