Can't Touch This: Consistent Network Updates for Multiple Policies

Szymon Dudycz, Arne Ludwig and Stefan Schmid

June 29, 2016

Motivation

Downtime on GitHub

In 2012 GitHub introduced loops in their network which led to performance drops for a day and 18 minutes of hard downtime needed to reconfigure the switches.

Motivation

Downtime on GitHub

In 2012 GitHub introduced loops in their network which led to performance drops for a day and 18 minutes of hard downtime needed to reconfigure the switches.

Other companies that had misconfigured their switches include etc. Amazon, GoDaddy and United Airlines.

Software Defined Network

Software Defined Network

Switches are controlled by a centralized controller. In principle, SDN is designed to enable formally consistent network operations.

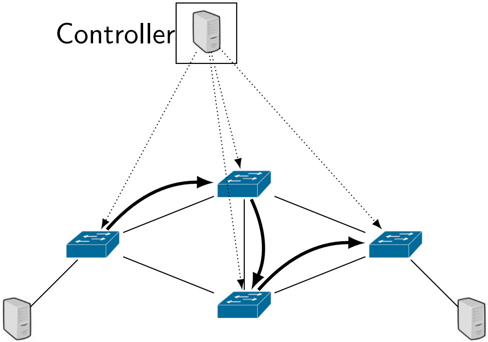
Software Defined Network

Software Defined Network

Switches are controlled by a centralized controller. In principle, SDN is designed to enable formally consistent network operations.

However there are still challenges related to distributed computing, in particular delays when updating switches.

Software Defined Network



Tagging

Algorithm (Reitblatt et al. SIGCOMM'12)

- Each switch remembers two paths for each policy
- Additional field in header of each packet

Tagging

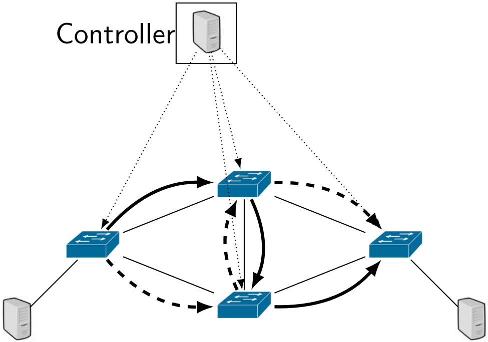
Algorithm (Reitblatt et al. SIGCOMM'12)

- Each switch remembers two paths for each policy
- Additional field in header of each packet

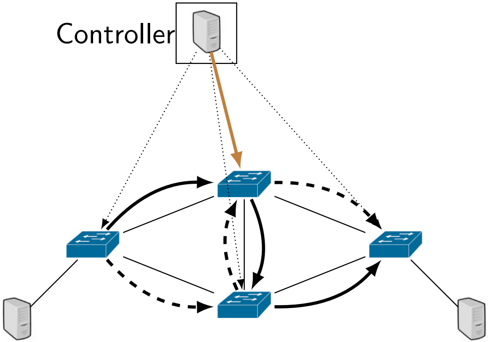
Cons

- Middleboxes may change header
- Requires more space in switch
- New links start being used late

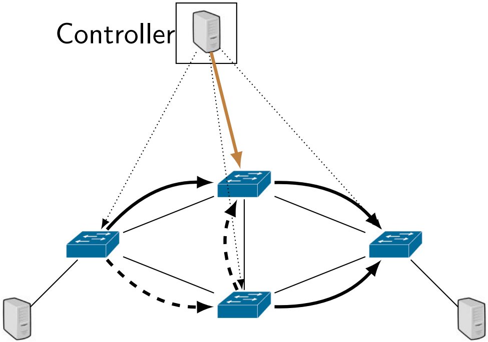
Policies



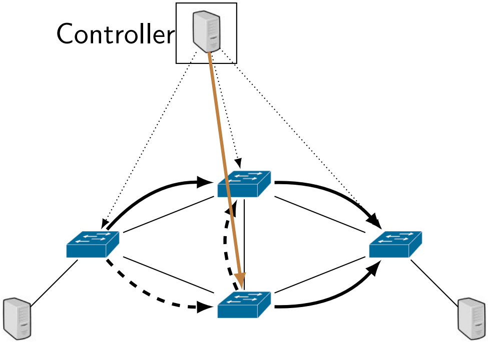
Policies



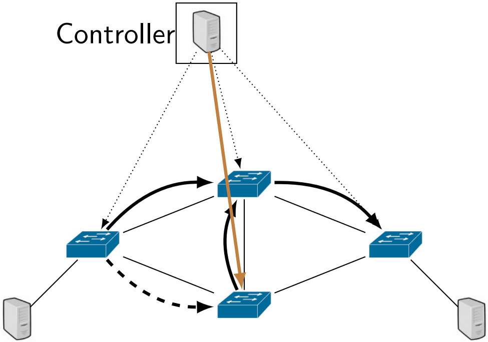
Policies



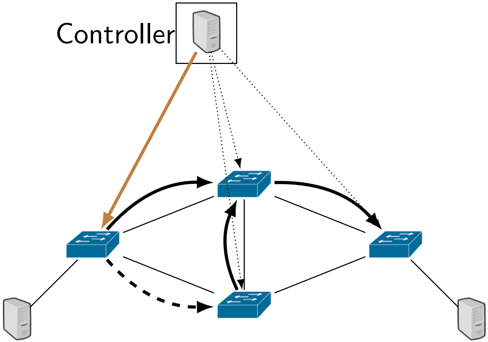
Policies



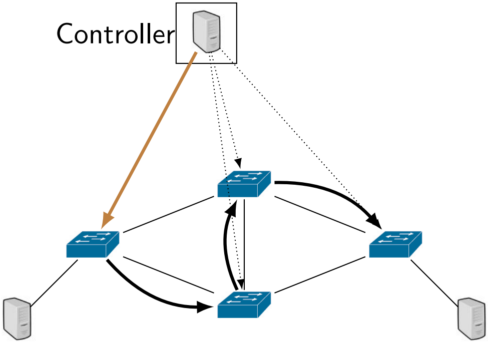
Policies



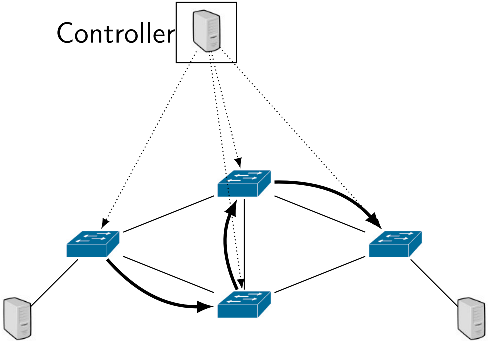
Policies



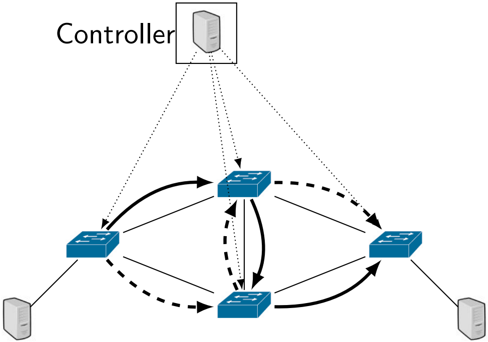
Policies



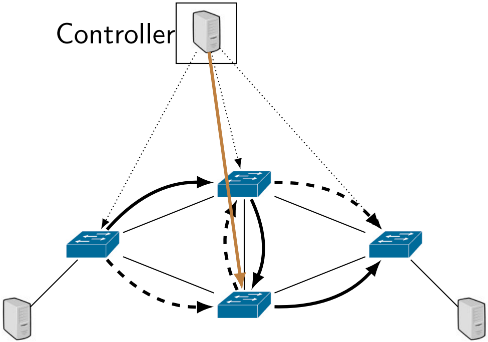
Policies



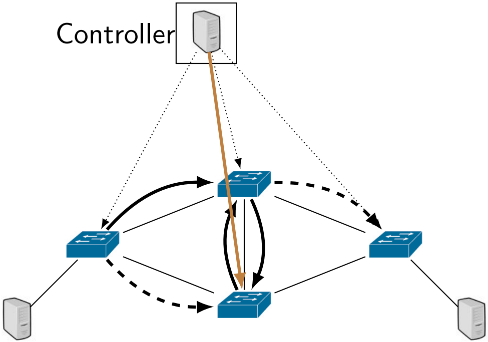
Loop freedom



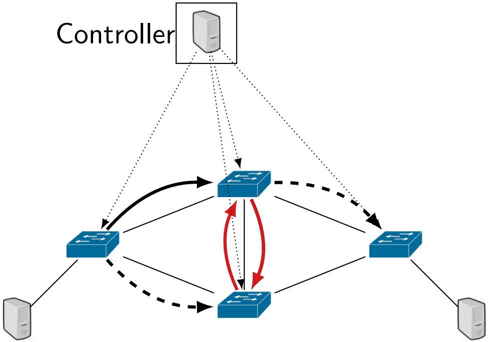
Loop freedom



Loop freedom



Loop freedom

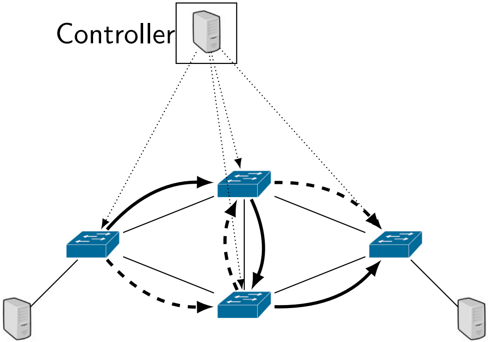


Rounds

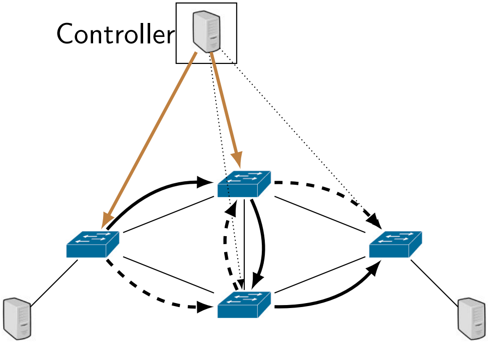
Rounds

At the beginning of the round, the controller updates any subset of switches. The switches in this subset can be updated in any order.

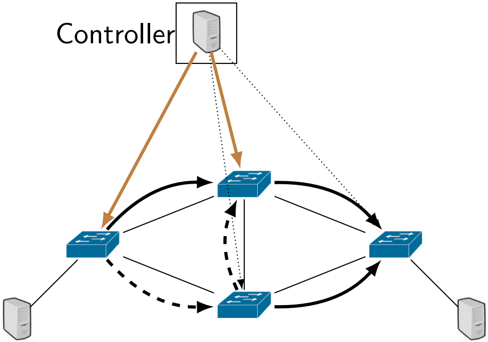
Rounds



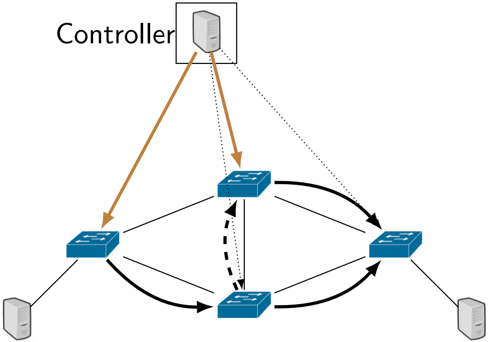
Rounds



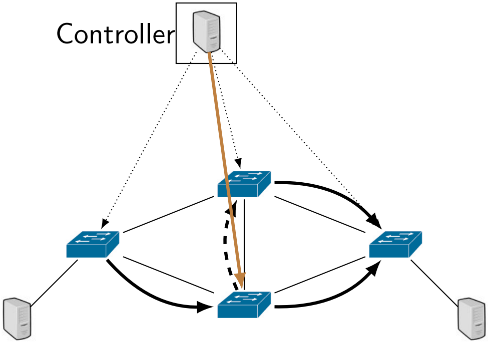
Rounds



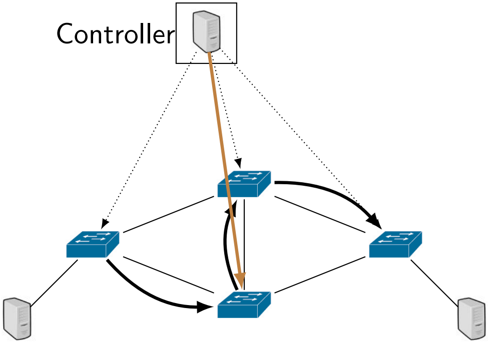
Rounds



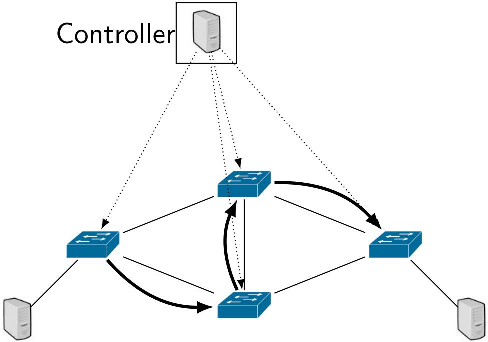
Rounds



Rounds



Rounds



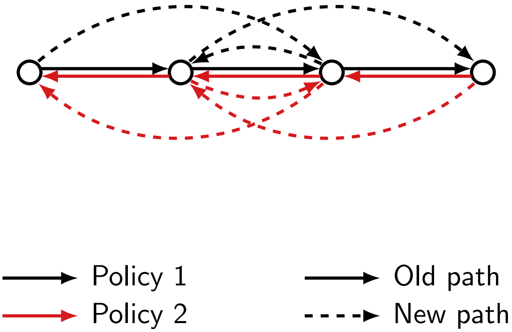
Rounds

Rounds

At the beginning of the round, the controller updates any subset of switches. The switches in this subset can be updated in any order.

Ludwig et al. showed that $\mathcal{O}(\log n)$ rounds is always enough to update a single policy in loop-free manner.

Policies

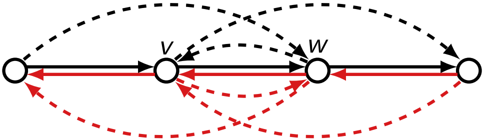


Touches

Touches

Multiple policies rules in one switch can be updated in bulk. We call one switch interaction a touch.

Policies



In → w must be updated after v

In → v must be updated after w

Goal

How to minimize number of touches?

$\mathcal{NP}$-hardness

It is $\mathcal{NP}$-hard to minimize number of touches even with only 3 policies.

$\mathcal{NP}$-hardness

It is $\mathcal{NP}$-hard to minimize number of touches even with only 3 policies.
The reduction is from a restricted variant of shortest common supersequence.

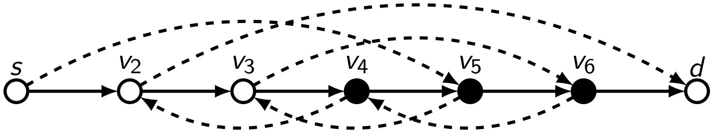
Easy case?

What about 2 policies, each of them updatable in 2 rounds?

Easy case?

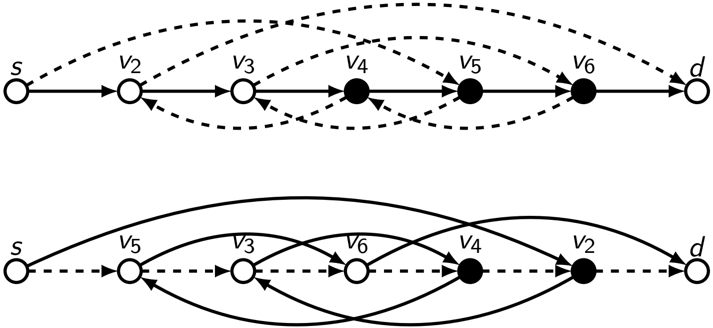
What about 2 policies, each of them updatable in 2 rounds?
Still $\mathcal{NP}$-hard.

Node classification

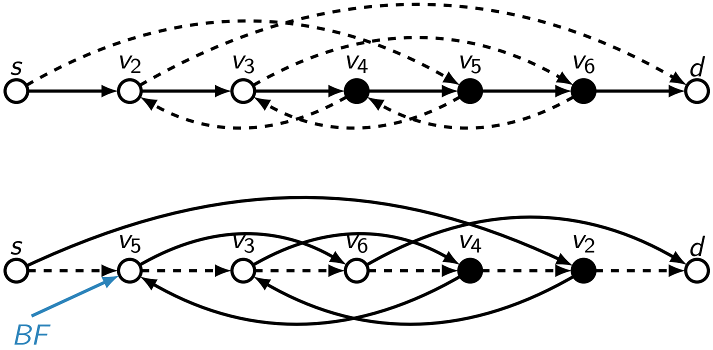


○*FORWARD* ●*BACKWARD*

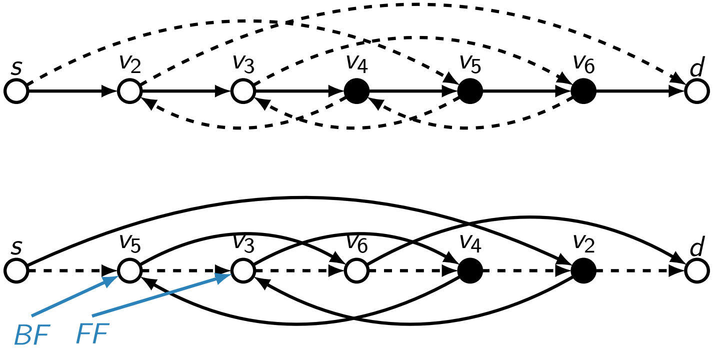
Node classification



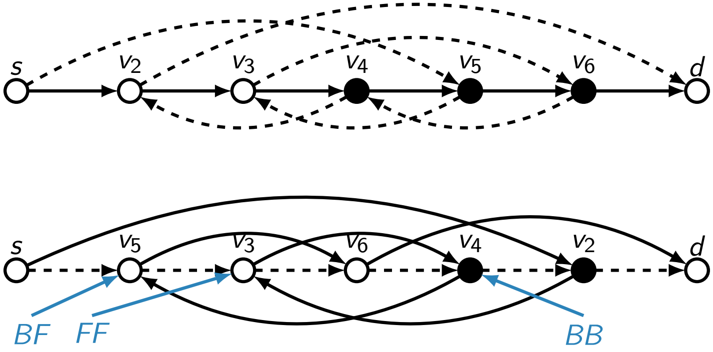
Node classification



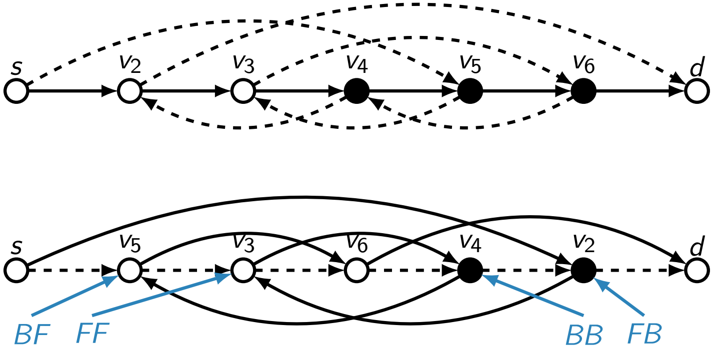
Node classification



Node classification



Node classification



Sketch of proof

In single policy 2 round schedules: (Ludwig et al. PODC'15)

- there cannot be any *BB* nodes
- *FB* nodes must be updated in the first round, and *BF* nodes must be updated in the last round
- *FF* nodes can be updated in any round

Sketch of proof

In 2 policies 3 round schedules:

Round

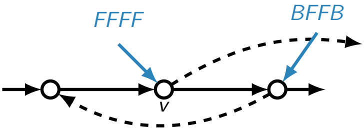
1	2	3
FBFB	*FBBF*	*BFBF*
FBFF	*BFFB*	*BFFF*
FFFB		*FFBF*
FFFF		*FFFF*

Sketch of proof

Idea:

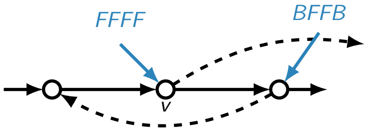
- Reduction from Max-2SAT in which each variable appears in at most 3 clauses.
- Create for each variable node of type *FFFF* and for each clause 2 nodes of type *FBBF*.
- Decision whether to update variable node in 1st or 3rd round corresponds to decision whether it should be assigned true or false.

Sketch of proof

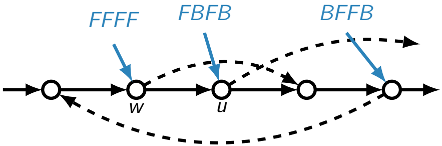


Round		
1	2	3
FBFB	*FBBF*	*BFBF*
FBFF	*BFFB*	*BFFF*
FFFB		*FFBF*
FFFF		*FFFF*

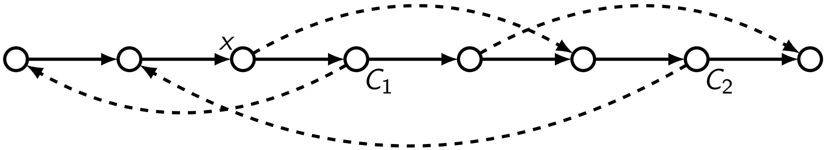
Sketch of proof

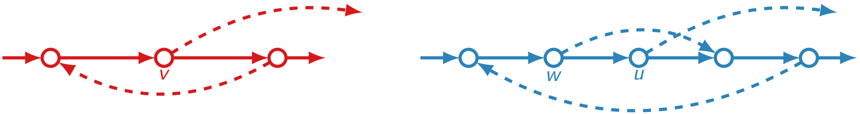


Round		
1	2	3
FBFB	FBBF	BFBF
FBFF	BFFB	BFFF
FFFB		FFBF
FFFF		FFFF

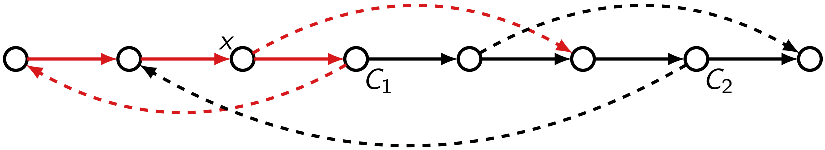


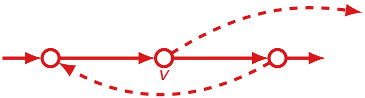
Sketch of proof

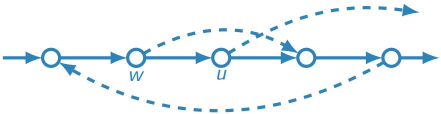




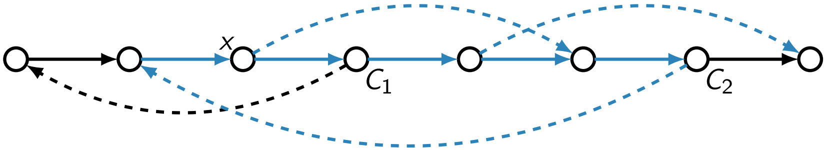
Sketch of proof

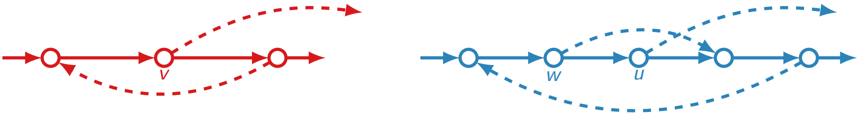






Sketch of proof





Sketch of proof

Steps to finish the reduction:

- Clause gadget, so that only one variable of each clause must be satisfied.
- Splitting clauses into two policies.
- Connecting the pieces, so that all vertices are of required type.

Schedule composition

What if we just need to compose schedules for each policies?

Schedule composition

What if we just need to compose schedules for each policies?
It is polynomial time computable as long as number of policies is constant.

Schedule composition

Policy 1: v_1, v_5, v_3, v_4

Policy 2: v_6, v_3, v_4, v_2 $\Longrightarrow$ $v_1, v_4, v_5, v_6, v_3, v_1, v_4, v_2$

Policy 3: v_4, v_5, v_6, v_1, v_2

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1			
$v_2 v_3$	2			
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2		
$v_2 v_3$	2			
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	
$v_2 v_3$	2			
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	3
$v_2 v_3$	2			
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	3
$v_2 v_3$	2	3		
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	3
$v_2 v_3$	2	3	3	
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	3
$v_2 v_3$	2	3	3	3
$v_2 v_3 v_1$	3			

Dynamic algorithm

Policy 1: v_1, v_2, v_3
Policy 2: v_2, v_3, v_1

	ϵ	v_1	$v_1 v_2$	$v_1 v_2 v_3$
ϵ	0	1	2	3
v_2	1	2	2	3
$v_2 v_3$	2	3	3	3
$v_2 v_3 v_1$	3	3	4	4

Schedule Composition

- When there are more than two policies, the array becomes multidimensional
- The running time is $\mathcal{O}(mn^m)$, where n is the number of nodes and m number of policies
- If number of policies is not constant, then the problem is $\mathcal{NP}$-hard
  - Reduction from directed feedback vertex set

Conclusion

Results:

- It is $\mathcal{NP}$-hard to minimize number of touches.
- If there is constant number of policies then there is polynomial time algorithm for composing schedules.
- However if the number of policies is not constant then the problem is also $\mathcal{NP}$-hard.

Conclusion

Open problems:

- Characterization of easy instances or good heuristic algorithm for loop-free multiple policy problem.
- Other consistency properties
  - Waypoint enforcement with loop freedom is in Ludwig et al. SIGMETRICS'2016
- Algorithm for loop-free single policy problem optimized for number of rounds
  - Improving $\mathcal{O}(\log n)$ rounds algorithm by Ludwig et al. PODC' 2015