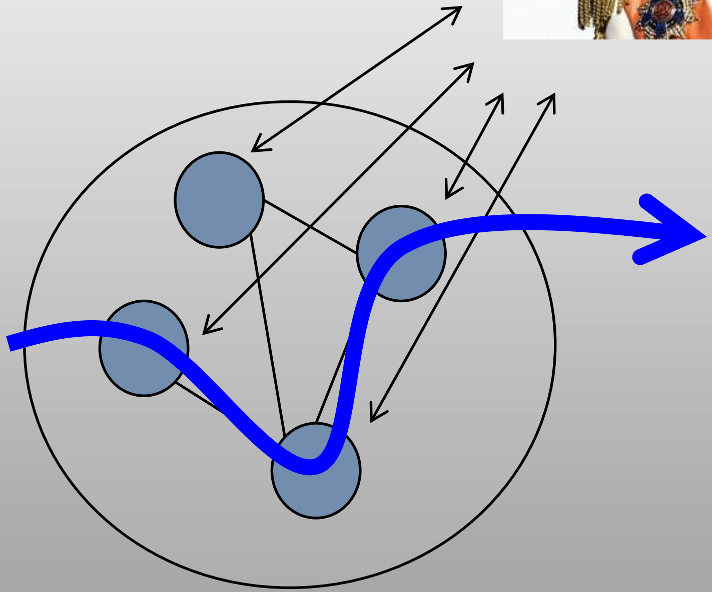
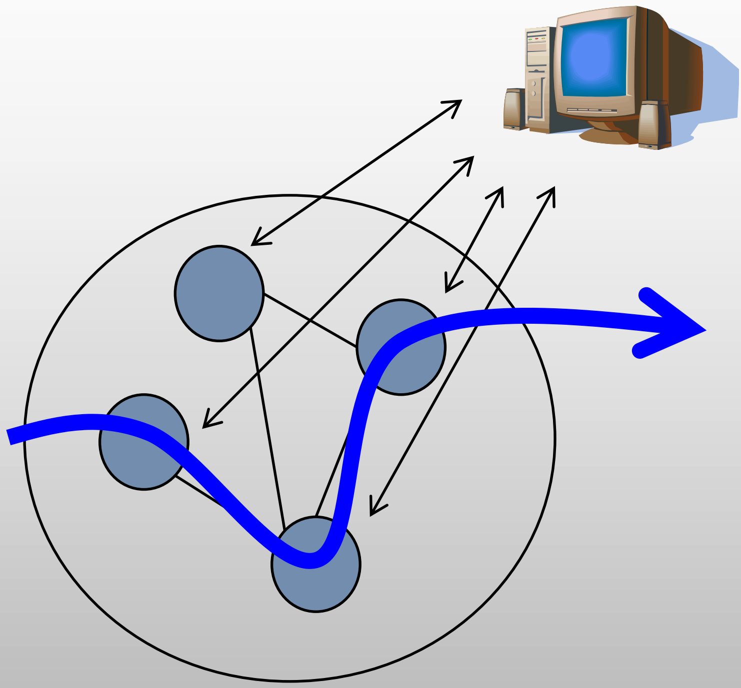
Exploiting Locality in Distributed SDN Control

Stefan Schmid (TU Berlin & T-Labs)
Jukka Suomela (Uni Helsinki)

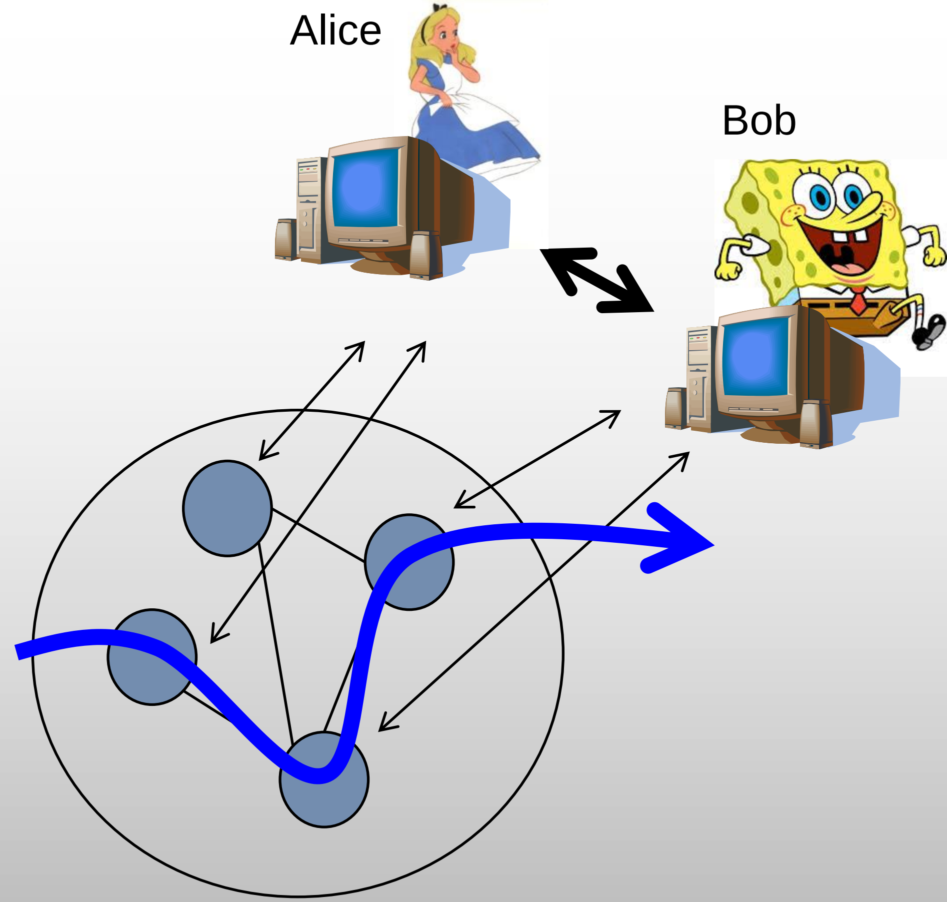
My view of SDN before I met Marco and Dan…



Logically Centralized, but Distributed!



VS



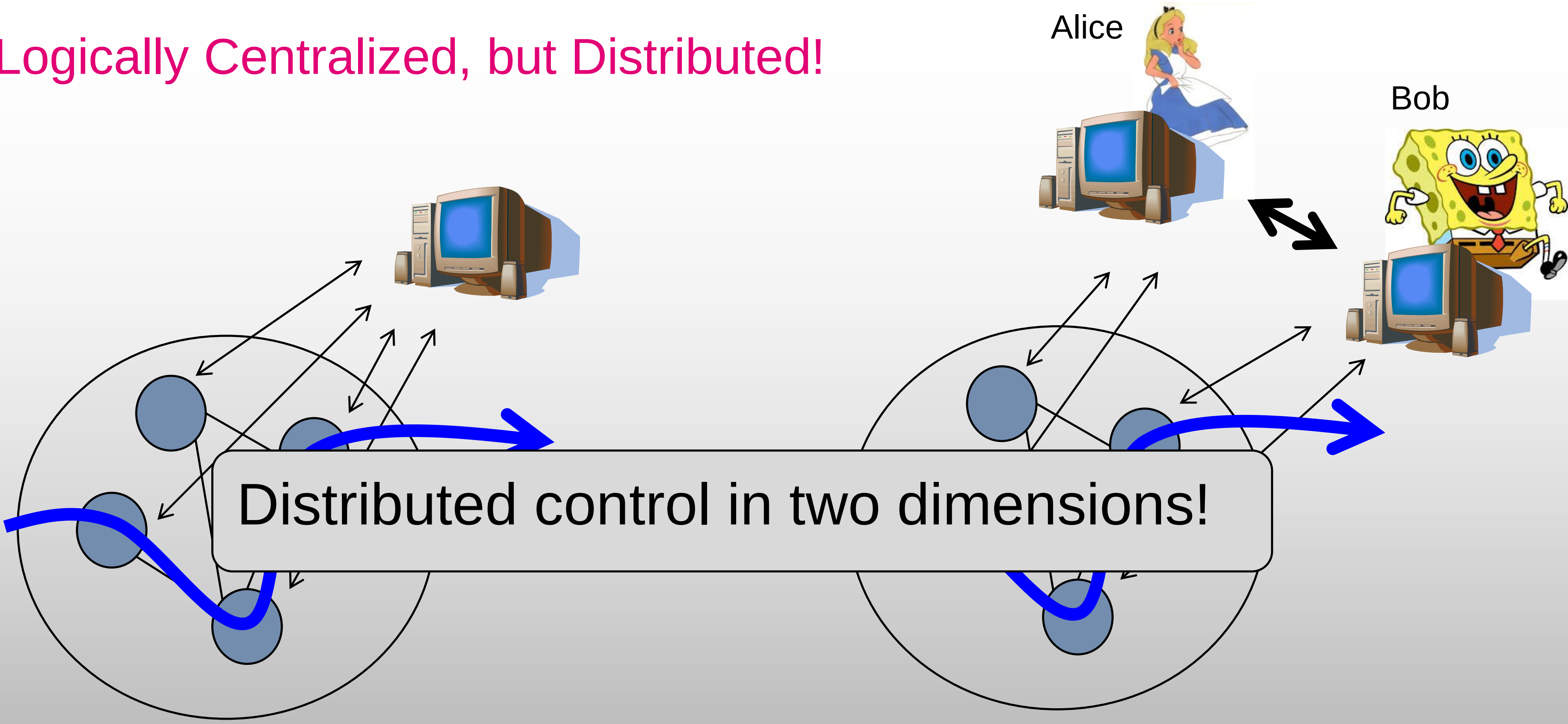
Vision:

- Control becomes distributed
- Controllers become near-sighted (control part of network or flow space)

Why:

- Enables wide-area SDN networks
- Administrative: Alice and Bob
  - Admin. domains, local provider footprint …
- Optimization: Latency and load-balancing
  - Latency e.g., FIBIUM
  - Handling certain events close to datapath and shield/load-balance more global controllers (e.g., Kandoo)

Logically Centralized, but Distributed!



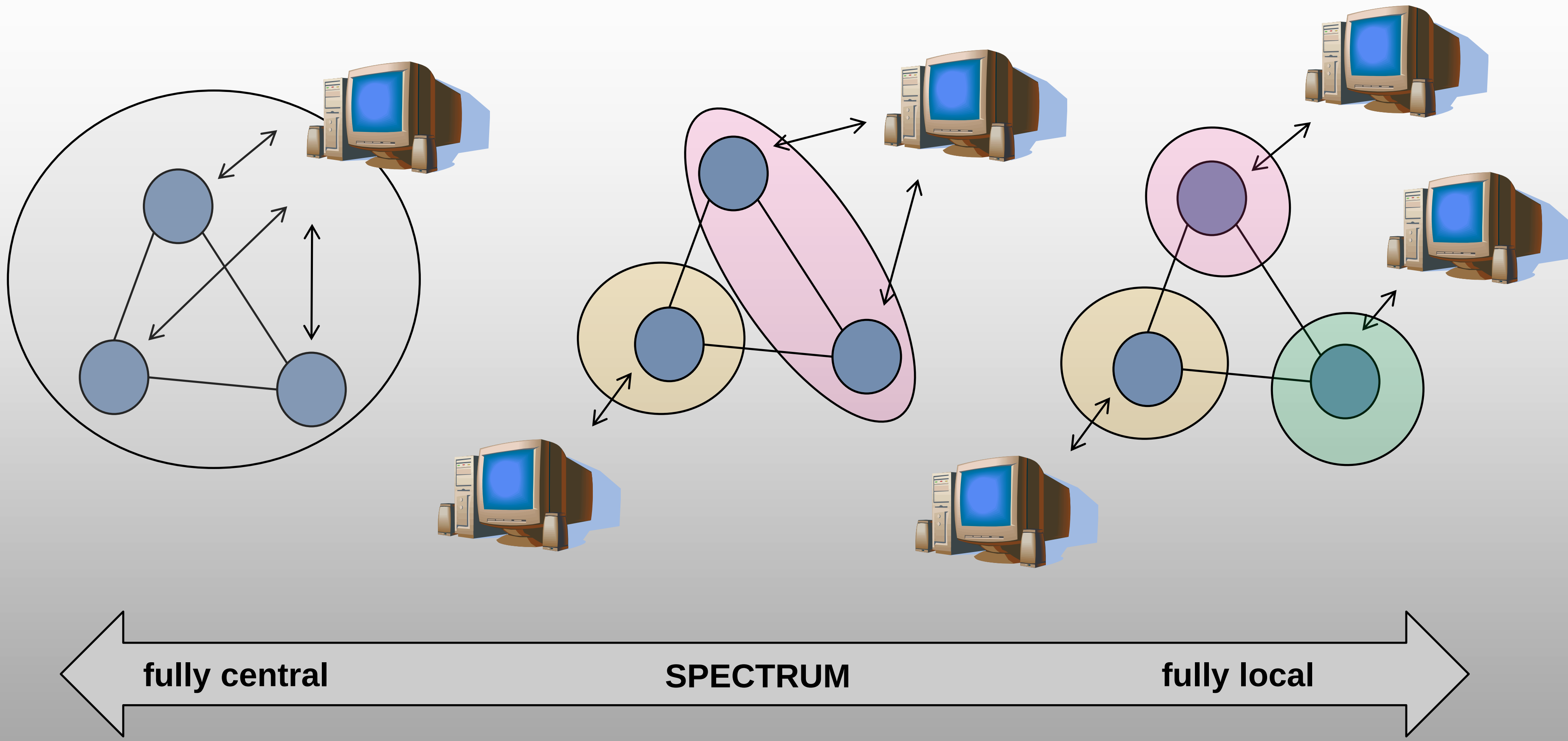
Vision:

- Control becomes distributed
- Controllers become near-sighted (control part of network or flow space)

Why:

- Enables wide-area SDN networks
- Administrative: Alice and Bob
  - Admin. domains, local provider footprint …
- Optimization: Latency and load-balancing
  - Latency e.g., FIBIUM
  - Handling certain events close to datapath and shield/load-balance more global controllers (e.g., Kandoo)

1st Dimension of Distribution: Flat SDN Control ("Divide Network")



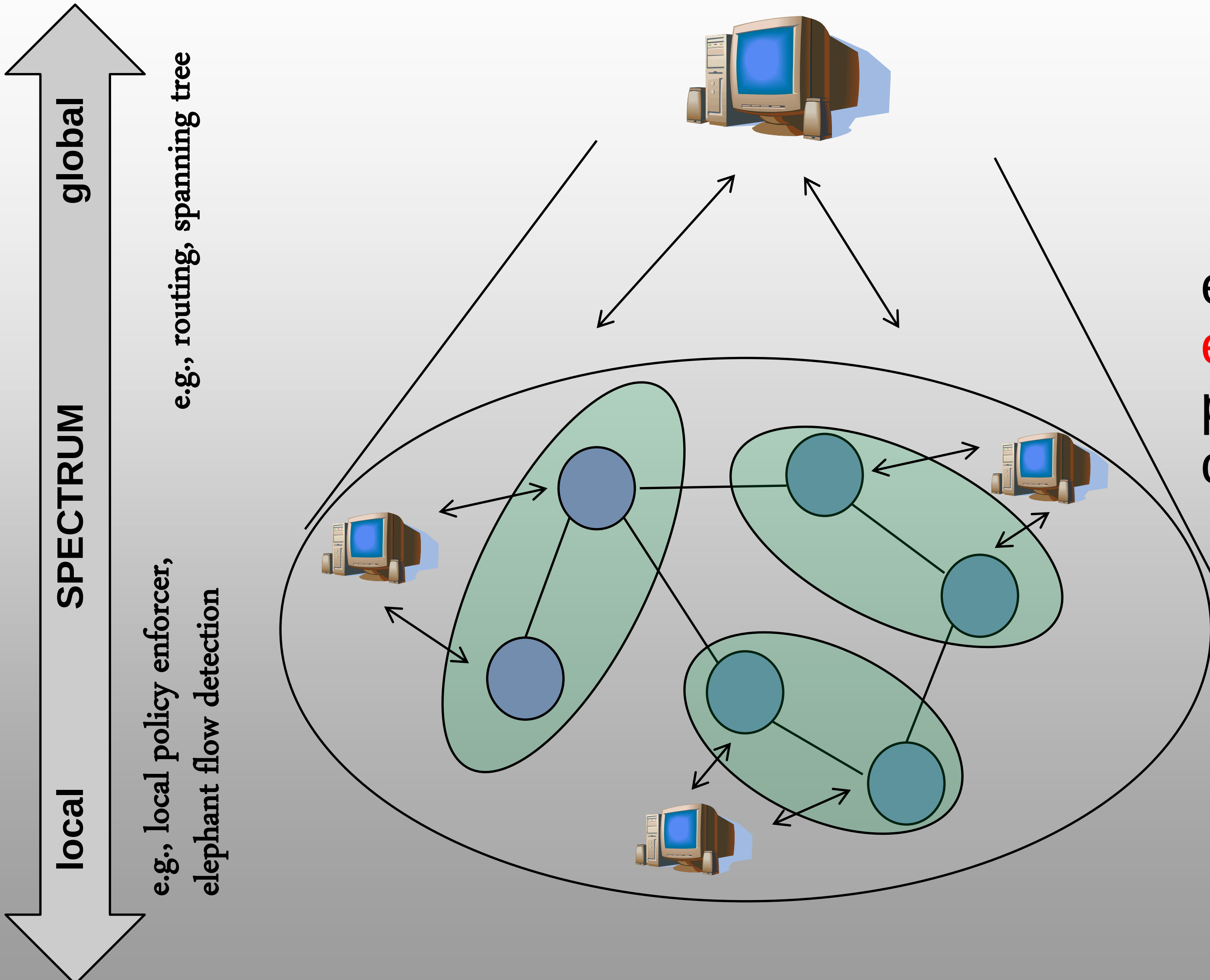
e.g., small network

e.g., routing control platform

e.g., SDN router (*FIBIUM*)

Stefan Schmid (T-Labs)

2nd Dimension of Distribution: Hierarchical SDN Control ("Flow Space")



e.g., handle frequent events close to data path, shield global controllers (*Kandoo*)

Stefan Schmid (T-Labs)

Questions Raised

- How to control a network if I have "local view" only?
- How to design distributed control plane (if I can), and how to divide it among controllers?
- Where to place controllers? (see Brandon!)
- Which tasks can be solved locally, which tasks need global control?
- ...

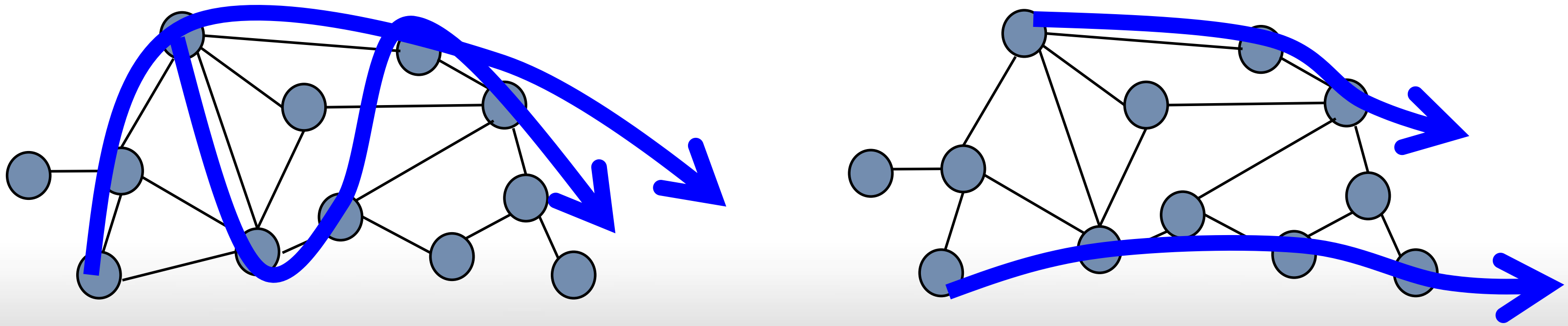
Our paper:

- Review and apply lessons to SDN from distributed computing and local algorithms* (emulation framework to make some results applicable)
- Study of case studies: (1) a *load balancing* application and (2) ensuring *loop-free forwarding* set
- First insights on what can be computed and verified locally (and how), and what cannot

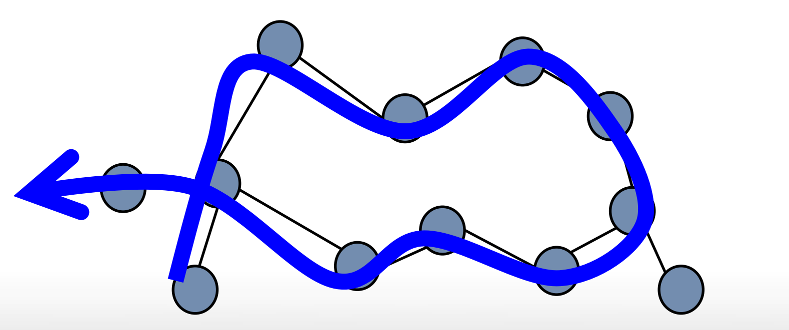
* Local algorithms = distributed algorithms with constant radius ("control infinite graphs in finite time")

Generic SDN Tasks: Load-Balancing and Ensuring Loop-free Paths

SDN for TE and Load-Balancing: Re-Route Flows



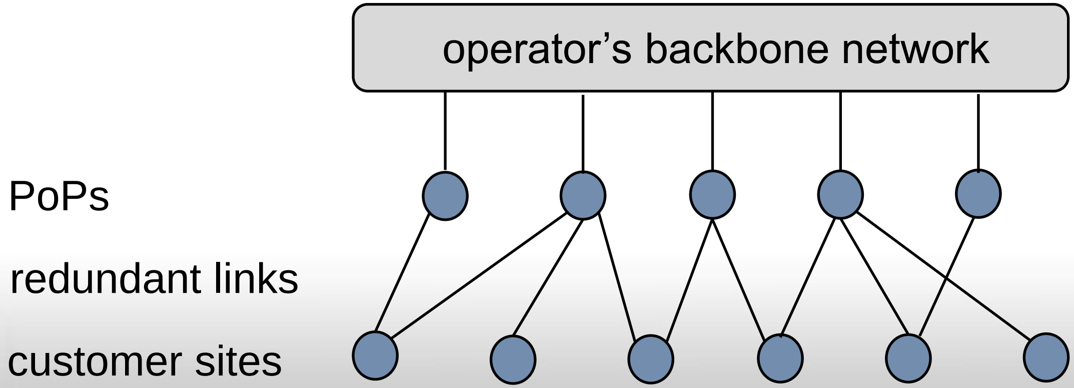
Compute and Ensure Loop-Free Forwarding Set



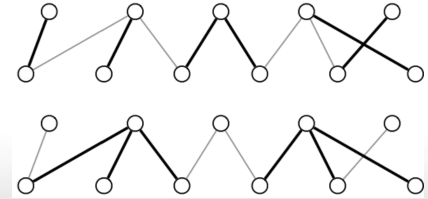
OK **not OK**

Concrete Tasks

SDN Task 1: Link Assignment („Semi-Matching Problem“)

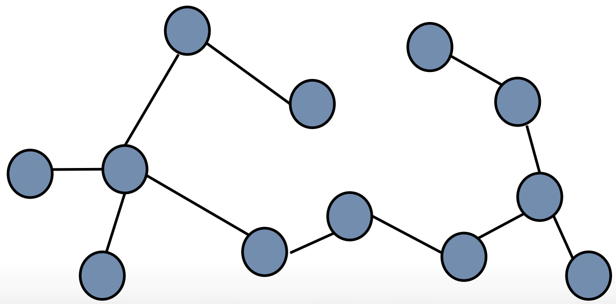


- Bipartite: customer to access routers
- How to assign?

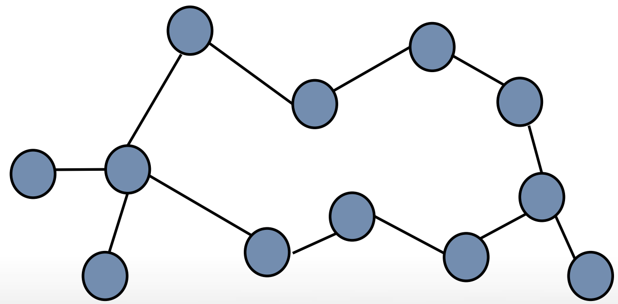


- Quick and balanced?

SDN Task 2: Spanning Tree Verification



OK

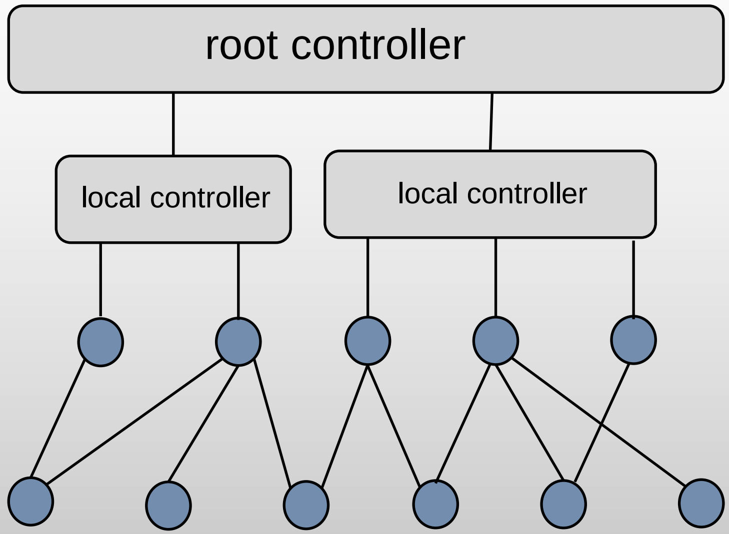


not OK

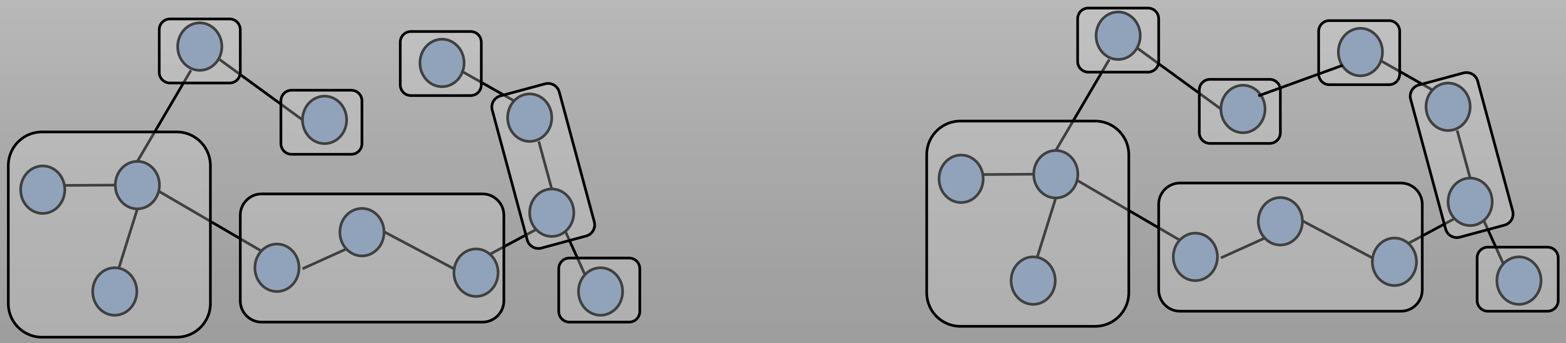
Both tasks are trivial under global control...!

… but not for distributed control plane!

- Hierarchical control:



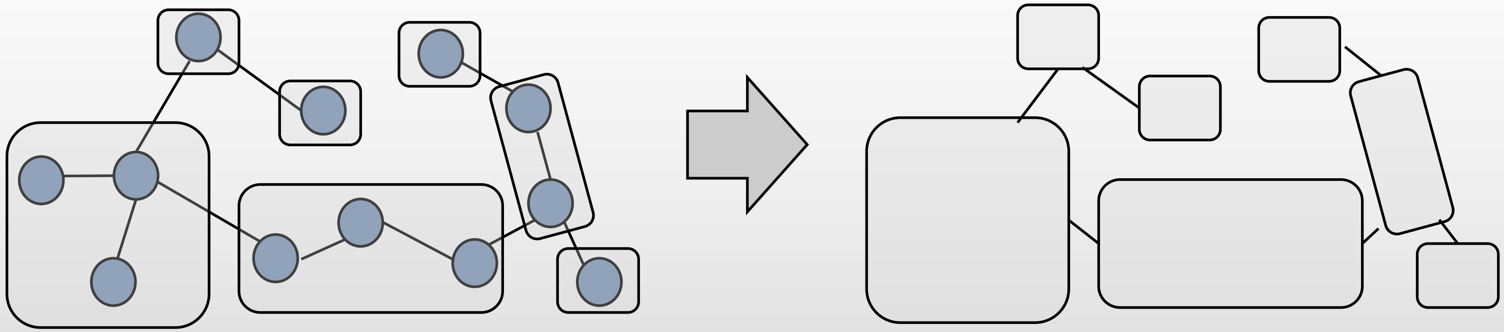
- Flat control:



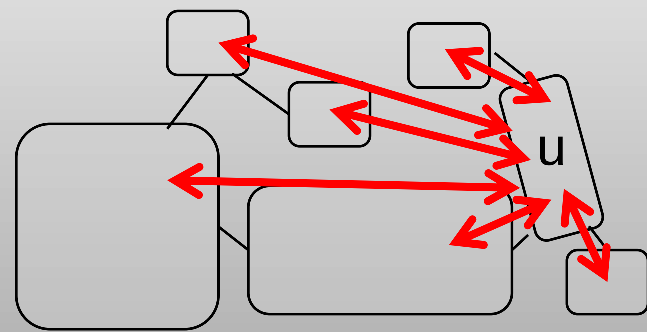
Stefan Schmid (T-Labs)

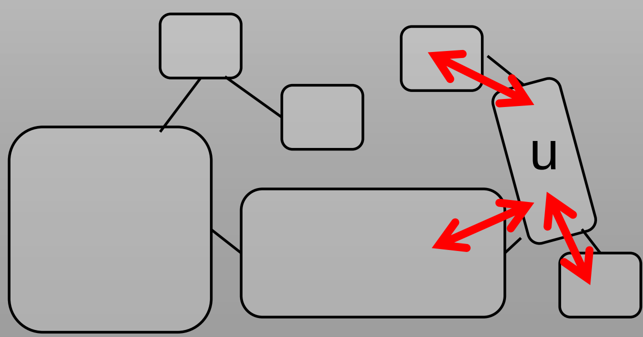
Local vs Global: Minimize Interactions Between Controllers

Useful abstraction and terminology: The “controllers graph”



Global task: inherently need to respond to events occurring at all devices.



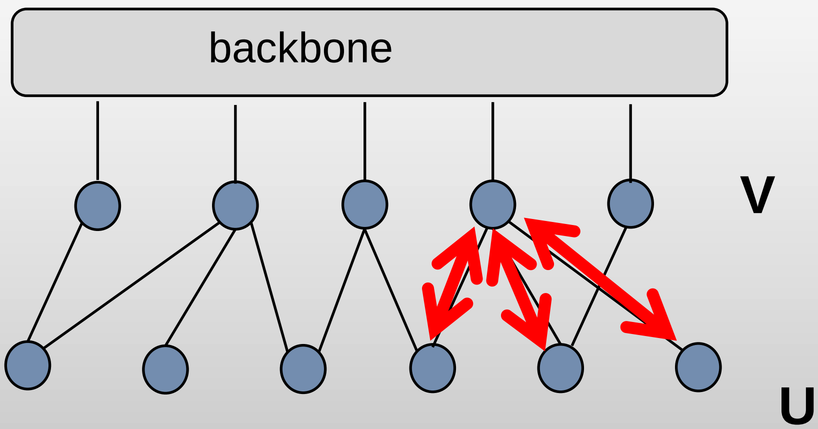


Local task: sufficient to respond to events occurring in vicinity!

Objective: minimize interactions (number of involved controllers and communication)

Take-home 1: Go for Local Approximations!

A semi-matching problem:

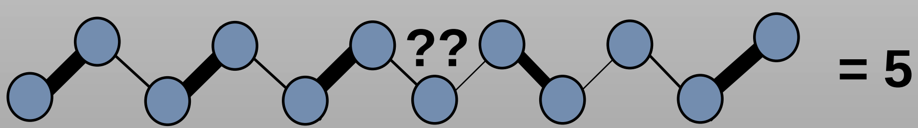


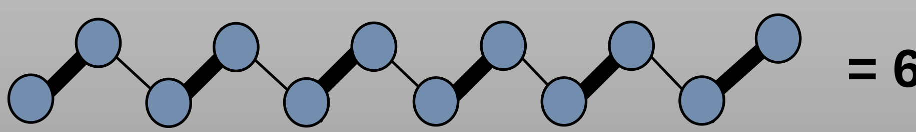
Semi-matching

If a customer u connects to a POP with c clients connected to it, the customer u costs c.

Minimize the average cost of customers!

The bad news: Generally the problem is inherently global e.g.,

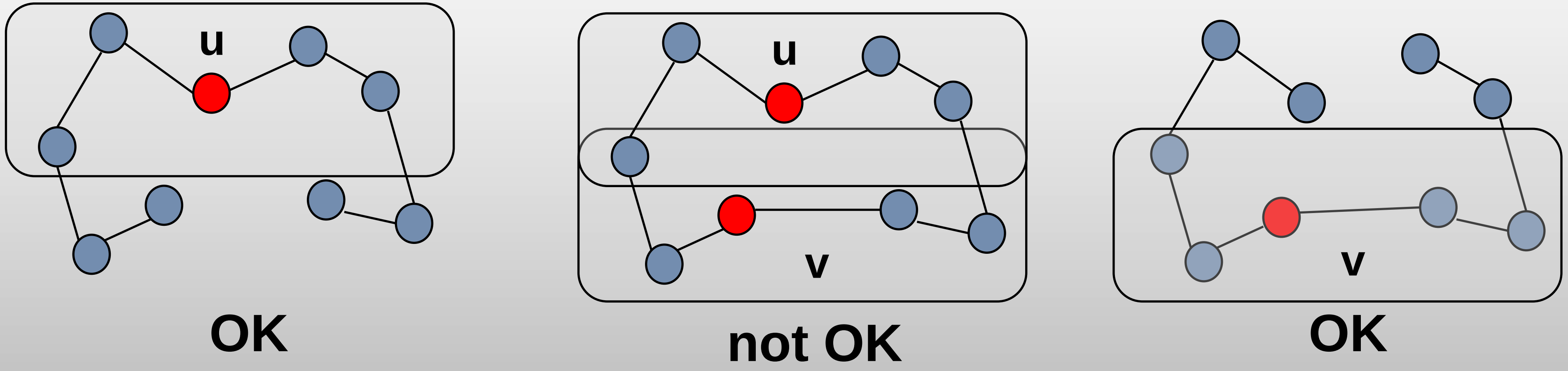




The good news: Near-optimal semi-matchings can be found efficiently and locally! Runtime independent of graph size and local communication only. (How? Paper! ☺)

Take-home 2: Verification is Easier than Computation

Bad news: Spanning tree computation (and even verification!) is an inherently global task.



2-hop local views of contrullers u and v: in the three examples, cannot distinguish the local view of a good instance from the local view of the bad instance.

Good news: However, at least verification can be made local, with minimal additional information / local communication between controllers (proof labels)!

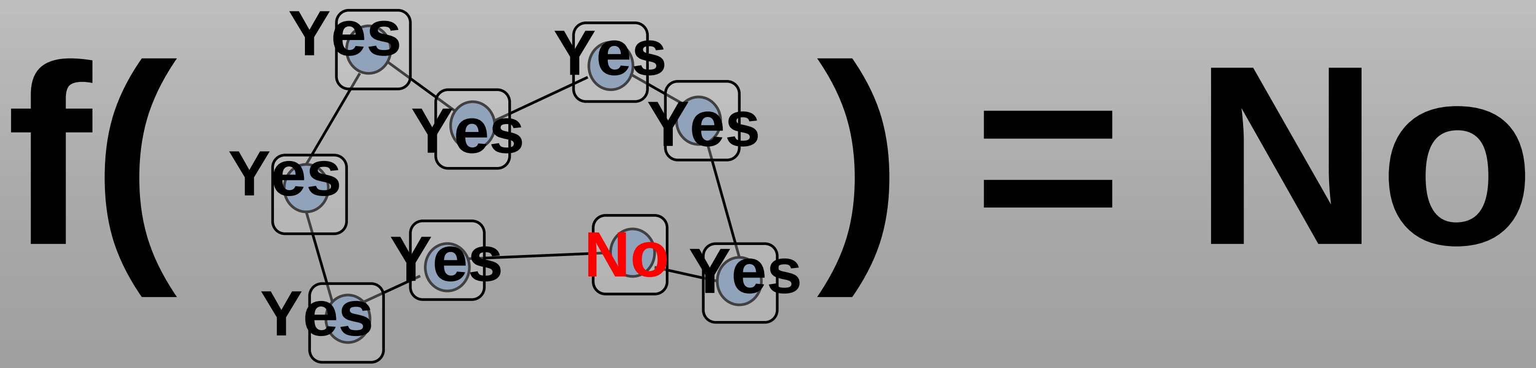
Stefan Schmid (T-Labs)

Proof Labeling Schemes

Idea: For verification, it is often sufficient if at least one controller notices local inconsistency: it can then trigger global re-computation!

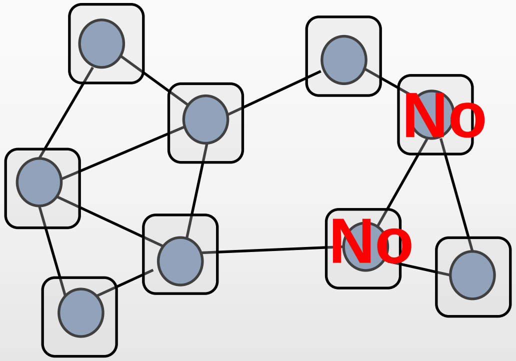
Requirements:

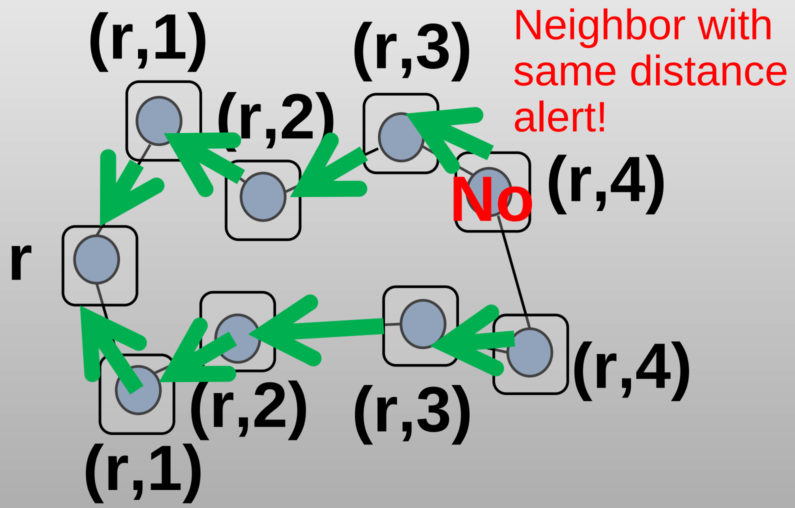
- Controllers exchange minimal amount of information ("proofs labels")
- Proof labels are small (an "SMS")
- Communicate only with controllers with incident domains
- Verification: if property not true, *at least one* controller will notice…
- … and raise alarm (re-compute labels)



Examples

Euler Property: Hard to compute Euler tour ("each edge exactly once"), but easy to verify! 0-bits (= no communication) : output whether degree is even.





Spanning Tree Property: Label encodes root node plus distance & direction to root. At least one node notices that root/distance not consistent! Requires O(log n) bits.

Any (Topological) Property: $O(n^2)$ bits.

Maybe also known from databases: efficient ancestor query! Given two log(n) labels.

Take-home 3: Not Purely Local, Pre-Processing Can Help!

Idea: If network changes happen at different time scales (e.g., topology vs traffic), pre-processing "(relatively) static state" (e.g., topology) can improve the performance of local algorithms (e.g., no need for symmetry breaking)!

Local problems often face two challenges: optimization and symmetry breaking. The latter may be overcome by pre-processing.

Example: Local Matchings

(M1) Maximal matching (only because of symm!)
(M2) Maximal matching on bicolored graph ← bipartite (like PoP assignment)
(M3) Maximum matching (symm+opt!)
(M4) Maximum matching on bicolored graph
(M5) Fractional maximum matching ← packing LP

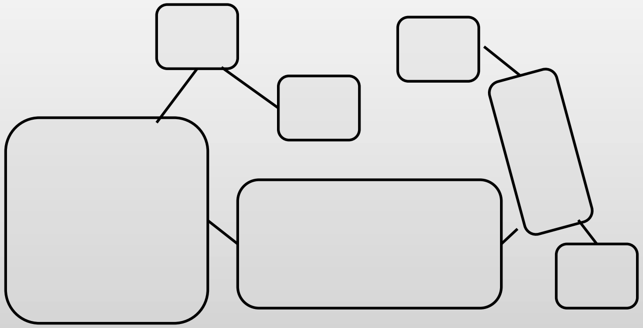
* **impossible, approx ok**, easy

Optimization:
(M1, M2): only need to find feasible solution!
(M1, M2, M3): need to find optimal solution!
Symmetry breaking:
(**M1**, M3): require symmetry breaking
(M2, M4): symmetry already broken
(M5): symmetry trivial

E.g., (M1) is simpler if graph can be pre-colored! Or *Dominating Set* (1. distance-2 coloring then 2. greedy [5]) , *MaxCut*, … The "supported locality model". ☺

Take-home >3: How to Design Control Plane

- Make your controller graph low-degree if you can!
- …



Problem	Approx. factor
Matching	$1+\varepsilon$
	$(\Delta+1)/2$
Weighted matching	$2+\varepsilon$
Simple 2-matching	$2+\varepsilon$
Semi-matching	$O(1)$
Edge cover	2
Vertex cover	2
	6
	$4+\varepsilon$
	3
	$2+\varepsilon$
	2
Dominating set	$\Delta+1$
	$2\lfloor\Delta/2\rfloor+1$
	$(\Delta+1)/2$
	$O(1)$
	$O(A\log\Delta)$
Domatic partition	$(\delta+1)/2$
Edge domin. set	$4-2/\Delta$

Exploiting Locality in Distributed SDN Control

Stefan Schmid
TU Berlin & T-Labs, Germany
stefan@net.t-labs.tu-berlin.de

Jukka Suomela
Helsinki Institute for Information Technology HIIT
Department of Computer Science
University of Helsinki
jukka.suomela@cs.helsinki.fi

ABSTRACT

Large SDN networks will be partitioned in multiple *controller domains*; each controller is responsible for one domain, and the controllers of adjacent domains may need to communicate to enforce global policies. This paper studies the implications of the local network view of the controllers. In particular, we establish a connection to the field of local algorithms and distributed computing, and discuss lessons for the design of a distributed control plane. We show that existing *local* algorithms can be used to develop efficient coordination protocols in which each controller only needs to respond to events that take place in its local neighborhood. However, while existing algorithms can be used, SDN networks also suggest a new approach to the study of locality in distributed computing. We introduce the so-called *supported locality model* of distributed computing. The new model is more expressive than the classical models that are commonly used in the design and analysis of distributed algorithms, and it is a better match with the features of SDN networks.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.4 [**Computer-Communication Networks**]: Distributed Systems; F.1.1 [**Computation by Abstract Devices**]: Models of Computation

Keywords

local algorithms, software defined networking

1. INTRODUCTION

The paradigm of **software defined networking** (SDN) advocates a more centralized approach of network control, where a controller manages and operates a network from a *global* view of the network. However, the controller may not necessarily represent a single, centralized device, but the control plane may consist of multiple controllers in charge of managing different administrative domains of the network or different parts of the flow space. The problem of managing a network and enforcing policies in such a distributed control plane, however, exhibits many similarities with the task of designing **local algorithms**—a well-studied subfield in the area of distributed computing. Local algorithms are distributed algorithms in which each device only needs to respond to events that take place in its local neighborhood, within some constant number of hops from it; put otherwise, these are algorithms that only need a constant number of communication rounds to solve the task at hand.

This paper highlights that there is a lot of potential for interactions between the two areas, the distributed control of SDN networks and local algorithms. On the one hand, we argue that there are many recent results related to local algorithms that are relevant in the efficient management and operation of SDN networks. On the other hand, we identify properties of SDN networks which raise additional and new challenges in the design of local algorithms. We describe a disparity between the features of SDN networks and the standard models of distributed systems that are used in the design and analysis of local algorithms. Indeed, there are many tasks that can be solved efficiently in real-world SDN networks, yet they do not admit a local algorithm in the traditional sense. We suggest a new model of distributed computing that separates the relatively static network structure (e.g., physical network equipment) and dynamic inputs (e.g., current traffic pattern).

1.1 Running Examples

We begin our exploration of the interactions between distributed SDN control and local algorithms with two scenarios that we will use as running examples.

Example 1. Link assignment. Consider an Internet Service Provider with a number of Points-of-Presence $v \in V$, and a number of customers $u \in U$. For each customer, there are multiple redundant connections between the customer's site and the operator's network. We can represent the connections between the customer sites and the access routers in the operator's network as a bipartite graph $G = (U \cup V, E)$, where an edge $\{u, v\} \in E$ indicates that there is a network link from customer site u to the access router in point-of-presence v.

Conclusion

- Local algorithms provide insights on how to design and operate distributed control plane. Not always literally, requires emulation! (No communication over customer site!)

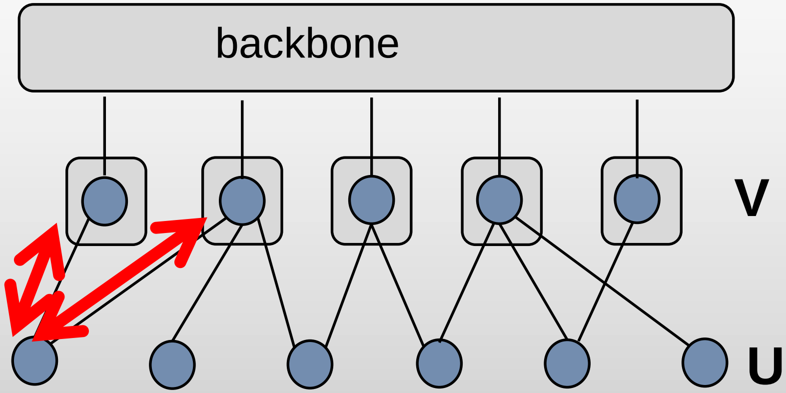
- **Take-home message 1:** Some tasks like **matching** are inherently global if they need to be solved optimally. But efficient **almost-optimal**, local solutions exist.
- **Take-home message 2:** Some tasks like **spanning tree computations** are inherently global but they can be locally verified efficiently with minimal additional communication!
- **Take-home message 3:** If network changes happen at different time scales, some pre-processing can speed up other tasks as well. A new non-purely local model.
- More in paper… ☺

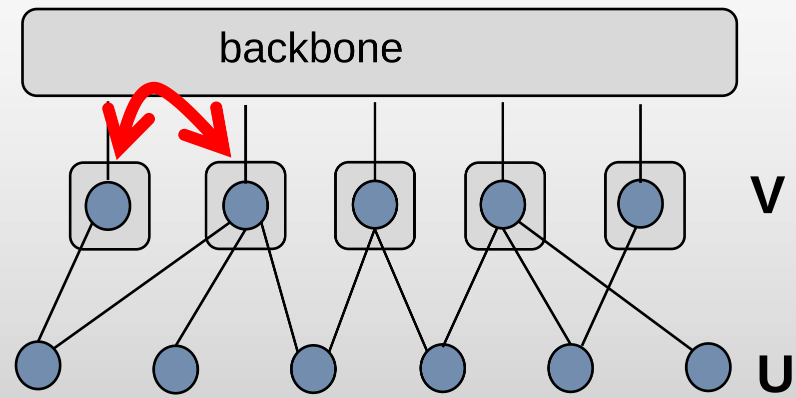
- And there are other distributed computing techniques that may be useful for SDN! See e.g., the upcoming talk on "Software Transactional Networking"

Backup: Locality Preserving Simulation

Controllers simulate execution on graph:

local controllers at PoPs





Algorithmic view:

distributed computation of the best matching

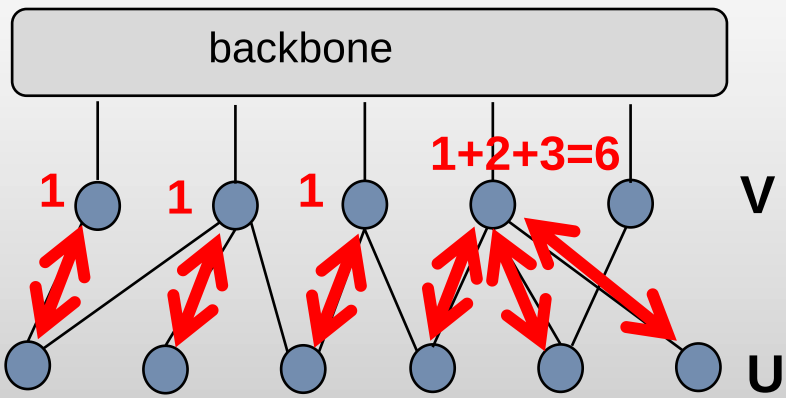
Reality:

controllers V simulate execution; each node v in V simulates its incident nodes in U

Locality: Controllers only need to communicate with controllers within 2-hop distance in matching graph.

Backup: From Local Algorithms to SDN: Link Assignment

A semi-matching problem:



Semi-matching

Connect *all* customers U: *exactly one* incident edge. If a customer u connects to a POP with c clients connected to it, the customer u costs c (not one: quadratic!).

Minimize the average cost of customers!

The bad news: Generally the problem is inherently global (e.g., a long path that would allow a perfect matching).

The good news: Near-optimal solutions can be found efficiently and locally! E.g., Czygrinow (DISC 2012): runtime independent of graph size and local communication only.