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!

**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)

Stefan Schmid (T-Labs)
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")

- Fully central
  - e.g., small network

- SPECTRUM
  - e.g., routing control platform

- Fully local
  - e.g., SDN router (FIBIUM)
2\textsuperscript{nd} Dimension of Distribution: Hierarchical SDN Control ("Flow Space")

- **Local**
  - e.g., local policy enforcer, elephant flow detection

- **Global**
  - e.g., routing, spanning tree

- SPECTRUM

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

PoPs
redundant links
customer sites

operator's backbone network

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:
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.

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

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

\[ f( \text{Yes, Yes, Yes, Yes, Yes, Yes, Yes, No, Yes} ) = \text{No} \]
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
(M3) Maximum matching (symm+opt!)
(M4) Maximum matching on bicolored graph
(M5) Fractional maximum matching

*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”. 😊

---

Stefan Schmid (T-Labs)
Take-home >3: How to Design Control Plane

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

- ...
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**”
Controllers simulate execution on graph:

**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.
A semi-matching problem:

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.