Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates

Klaus-T. Foerster
Brief Preamble

• Focus on algorithmic/complexity issues in consistent updates in Software Defined Networks (SDNs)
  ◦ Not so much on system etc. issues respectively SDNs themselves

• Two “bigger” connections to classic distributed computing halfway-in
  ◦ Proof Labeling Schemes
  ◦ Distributed Control Plane
Network Updates

• The Internet: Designed for selfish participants
  ◦ Often inefficient (low utilization of links), but robust

• But what if eg the Wide-Area Network is controlled by a single entity?
  ◦ Examples: Microsoft & Amazon & Google ...
  ◦ They spend hundreds of millions of dollars per year
Network Updates

Think: Google, Amazon, Microsoft

Also relevant in e.g. Data Center Networks, for ISPs etc

Software-Defined Networking

• Possible solution:
  ◦ Software-Defined Networking (SDNs)

• General Idea: Separate data & control plane in a network

• Centralized controller updates networks rules for optimization
  ◦ Controller (control plane) updates the switches/routers (data plane)

• Logically centralized controller (eg implemented with replication)

Note: There is also a lot of (prior) research on consistency before SDNs – can’t cover everything in this tutorial

See history section in:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019
old network rules

network updates

new network rules
old network rules

network updates

new network rules

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
old network rules

possible solution: be fast!

e.g., B4 (Google, 2013)

new network rules

But they deviated from that a bit in the B4 2018 version...
old network rules

Alternative: Be consistent!
• Algorithms with guarantees

network updates

new network rules
Toy Example
Toy Example

Link should not be used anymore
eg repair, congestion, policy change etc
Toy Example
Toy Example
Toy Example
Toy Example
Appears in Practice

“Data plane updates may fall behind the control plane acknowledgments and may be even reordered.”
Kuzniar et al., PAM 2015

“...the inbound latency is quite variable with a [...] standard deviation of 31.34ms...”
He et al., SOSR 2015

“some switches can ‘straggle,’ taking substantially more time than average (e.g., 10-100x) to apply an update”
Jin et al., SIGCOMM 2014
Toy Example

Old and new states exist simultaneously in a limbo state
Ordering Solution: Go backwards through the new routing tree
Ordering Solution: Go backwards through the new routing tree
Ordering Solution: Go backwards through the new routing tree
Ordering Solution: Go backwards through the new routing tree

![Routing Tree Diagram]

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
Ordering Solution: Go backwards through the new routing tree

![Diagram showing the process of going backwards through the routing tree with nodes v, u, and d, and an update highlighted.](image)
Ordering Solution: Go backwards through the new routing tree

- Always works for single-destination rules
  - Also for multi-destination with sufficient memory ("split")
- Schedule length: tree depth (up to $\Omega(n)$)
  - Optimal scheduling algorithms?

More on scheduling multiple policies:
Basta et al: Efficient Loop-Free Rerouting of Multiple SDN Flows. ToN 2018
Greedy? Update as many as possible per round

- Always works 😊
network updates

...
greedy maximal update
a & b update $\rightarrow$ all others wait
2 nodes update
greedy maximal update
a & b update → all others wait
2 nodes update

maximum update
a waits → all others update
all but 1 update
How hard?

**greedy maximal update**
- a & b update → all others wait
  - 2 nodes update

**maximum update**
- a waits→ all others update
  - all but 1 update
Find maximum update?

• Let’s go more general
• Delete all cycles in a graph
Find maximum update?

• Let’s go more general
• Delete all cycles in a graph
Find maximum update?

- Let’s go more general
- Delete all cycles in a graph
- **NP-hard** to approximate
  - *Feedback Arc Set*
Find maximum update?

• Let’s go more general
• Delete all cycles in a graph
• **NP-hard** to approximate
  – *Feedback Arc Set*
• And it’s (essentially) equivalent 😞
Find maximum update?

• Let’s go more general
• Delete all cycles in a graph
• **NP-hard** to approximate
  – *Feedback Arc Set*
• And it’s (essentially) equivalent 😞
Greedy? Update as many as possible per round

- Always works 😊

- Maximizing is NP-hard 😞

- Single greedy update: $O(1)$ rounds $\Rightarrow$ $\Omega(n)$ rounds 😞 😞

- In general: Does a 3-round schedule exist? NP-hard 😞 😞 😞
Relax And Take it Easy!
Scheduling Loop-free Network Updates: It's Good to Relax! [Ludwig et al., PODC 2015]

Two key ideas:
1. destination-based source-destination pairs \( <s,d> \) 
2. no forwarding loops no loops between \( <s,d> \)
Scheduling Loop-free Network Updates: It's Good to Relax!

• Non-relaxed? $\Omega(n)$ rounds

• Relaxed?
Scheduling Loop-free Network Updates: It's Good to Relax!

• Non-relaxed? $\Omega(n)$ rounds

• Relaxed?

Round 1

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? $\Omega(n)$ rounds

- Relaxed?

Round 1
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? $\Omega(n)$ rounds

- Relaxed?

Round 1
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? $\Omega(n)$ rounds

- Relaxed?

Round 2
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? $\Omega(n)$ rounds

- Relaxed? Just 3 rounds
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed? Just 3 rounds
  - In general: O(log n) rounds (“Peacock“)

---

(a) The graph $G_1$ with 16 nodes. When Peacock selects the edge from $0/8$ to $4/8$ as a shortcut, pruning results in the graph in Fig. 10b.

(b) After two rounds with Peacock, isomorphic to $G_0$ in Fig. 10c.

(c) The graph $G_0$ with 8 nodes. $0/8$ to $4/8$ is the next shortcut.

(d) To the left, the output of Peacock on $G_0$ after two rounds. To the right, after two more rounds, selecting the first forward edge as a shortcut each time.

(e) The resulting updated graph, expanded into 16 nodes again.
Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? $\Omega(n)$ rounds

- Relaxed? Just 3 rounds
  - In general: $O(\log n)$ rounds ("Peacock")
  - But: Peacock instances with $\Omega(\log n)$ rounds

---

Some Open Questions for scheduling loop free updates:

- For both models: Approximation algorithms for #rounds?

Relaxed:
- Optimal #rounds: NP-hard or in P?
- What is the real lower bound?

Non-relaxed:
- NP-hard for $O(1) < k < \Omega(n)$ rounds?

More open questions and specifics:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019
So Far Everything Was Sort of Centralized...

• ...can we make it more distributed?
Decentralized Updates for „Tree-Ordering“

• So far: every round:
  ◦ Controller computes and sends out updates
  ◦ Switches implement them and send acks
  ◦ Controller receives acks
Decentralized Updates for „Tree-Ordering“

• So far: every round:
  ◦ Controller computes and sends out updates
  ◦ Switches implement them and send acks
  ◦ Controller receives acks

• Alternative: Use dualism to so-called *proof labeling schemes*
Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 1908-02

Proof-Labeling Schemes
Deciding vs Checking

Prove

Verify

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
Brief Selected Background

- [Naor and Stockmeyer, STOC 1993]: 
  *What can be computed locally?*

- [Korman et al., PODC 2005]:
  *Proof Labeling Schemes (PLS)*

- [Göös and Suomela, PODC 2011]:
  *Locally Checkable Proofs (LCP)*

- [Fraigniaud et al., FOCS 2011,...]:
  *Nondeterministic Local Decision (NLD)*

- And many more recent works, e.g., on approximation, randomization etc.
Example
Example

Model

- Each of the $n$ nodes \( \bigcirc \) is a computer, connected by links
- Synchronous rounds
  - Simplified: unlimited message size & computational power, unique identifiers for nodes
Example

• Is $n$ even?
Example

- Is $n$ even?
- $\Omega(n)$ rounds
Example

- Is $n$ even?
- $\Omega(n)$ rounds
- What if I tell you it is even? Why should you trust me ☺
Example

- Is $n$ even?
- $\Omega(n)$ rounds
- Prover assigns 1 bit?
Example

• Is $n$ even?
• $\Omega(n)$ rounds
• Prover assigns 1 bit $\Rightarrow$ Verify in 1 round
Example

- Is \( n \) even?
- \( \Omega(n) \) rounds
- Prover assigns 1 bit -> Verify in 1 round
- Other way to think of it: 1 bit of non-determinism
- General question: How many bits necessary/sufficient?
Accepting a proof

- Every node outputs **Yes** -> Proof accepted
- One node outputs **No** -> Proof rejected
Accepting a proof

• Every node outputs \textbf{Yes} \rightarrow \text{Proof accepted}

• One node outputs \textbf{No} \rightarrow \text{Proof rejected}
  \begin{itemize}
    \item \textit{Prover} chose the wrong proof
  \end{itemize}
Accepting a proof

- Every node outputs Yes -> Proof accepted
- One node outputs No -> Proof rejected
  - Prover chose the wrong proof
  - Property does not hold

Back to SDNs: Switch from a proof to another
Decentralized Updates for "Tree Ordering"

When should I update?
Decentralized Updates for True Ordering“

Once my parent updates!
Decentralized Updates for Tree Ordering

Once my parent updates!

Send parent ID
Decentralized Updates for „Tree-Ordering“
Decentralized Updates for Tree Ordering“

I’ll update too!

I updated
Decentralized Updates for “Tree-Ordering”

+ Only one controller-switch interaction per route change
+ New route changes can be pushed before old ones done *(include “version#”)*
+ Incorrect updates can be locally detected *(include depth in tree, prevents loops)*

+/− Speed benefit/penalty depends on update scenario and topology

- Requires switch-to-switch communication e.g., [Nguyen et al., SOSR 2017]

Can we also make the initial computation decentralized?

• Classic setting of distributed computing (e.g. LOCAL or CONGEST model)
  ◦ Possible benefit in SDNs:
    - We do not need to compute from scratch!
      • In wired networks, problems depend on a subset of the network
        - Leverage Preprocessing

• Further explored in eg:
  ◦ Exploiting Locality in Distributed SDN Control. S. Schmid, J. Suomela, HotSDN 2013
Coloring of rings (LOCAL model)
Coloring of rings (LOCAL model)

• 2-coloring:
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs $\Omega(n)$ rounds
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs $\Omega(n)$ rounds

- 3-coloring:
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs $\Omega(n)$ rounds

- 3-coloring:
  - Needs non-constant time
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs $\Omega(n)$ rounds

- 3-coloring:
  - Needs non-constant time

- Cannot improve in the LOCAL model 🙁
Coloring of rings (LOCAL model) – with Preprocessing

- 2-coloring:

- 3-coloring:
Coloring of rings (LOCAL model) – with Preprocessing

• 2-coloring:
  ◦ 0 rounds 😊

• 3-coloring:
  ◦ 0 rounds 😊
Coloring of rings (LOCAL model) – with Preprocessing

• 2-coloring:
  ◦ 0 rounds 😊

• 3-coloring:
  ◦ 0 rounds 😊
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

• How about a coloring of a subgraph?

• Local model: runtime does not change
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

• How about a coloring of a subgraph?

• Local model: runtime does not change

• With preprocessing: fast!
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?

- Local model: runtime does not change

- With preprocessing: fast!
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

• How about a coloring of a subgraph?

• Local model: runtime does not change

• With preprocessing: fast!
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!
  - Coloring remains valid
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

• How about a coloring of a subgraph?

• Local model: runtime does not change

• With preprocessing: fast!
  ◦ Coloring remains valid

• What are further application scenarios?
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?

- Local model: runtime does not change

- With preprocessing: fast!
  - Coloring remains valid

- What are further application scenarios?
- What else can we do with the SUPPORT of Preprocessing?
Practical Motivation for Preprocessing
Practical Motivation for Preprocessing

- Decentralization aids scalability
Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms
Practical Motivation for Preprocessing

• Decentralization aids scalability
  ◦ But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

• Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
Practical Motivation for Preprocessing

• Decentralization aids scalability
  ◦ But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

• Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  ◦ Unknown network state too strong assumption for many scenarios
Practical Motivation for Preprocessing

• Decentralization aids scalability
  ◦ But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

• Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  ◦ Unknown network state too strong assumption for many scenarios
  ◦ Often we just react to events, physical topology in wired networks does not grow suddenly
Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  - Unknown network state too strong assumption for many scenarios
  - Often we just react to events, physical topology in wired networks does not grow suddenly

- Example: Software-Defined Networking, single (logically centralized) controller does not scale
Practical Motivation for Preprocessing

• Decentralization aids scalability
  ◦ But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

• Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  ◦ Unknown network state too strong assumption for many scenarios
  ◦ Often we just react to events, physical topology in wired networks does not grow suddenly

• Example: Software-Defined Networking, single (logically centralized) controller does not scale
  ◦ Create many local controllers that can react quickly, that control small set of “dumb” nodes
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
  E.g. MAC-address
The SUPPORTED Model

• Extends the LOCAL model (with unique IDs) with preprocessing
  E.g. MAC-address

• Original structure given as the SUPPORT graph $H=(V(H),E(H))$
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
  - E.g. MAC-address
- Original structure given as the SUPPORT graph $H=(V(H), E(H))$
- Problem instance is a subgraph $G=(V, E)$ of $H$
The SUPPORTED Model

• Extends the LOCAL model (w. unique IDs) with preprocessing

  E.g. MAC-address

• Original structure given as the SUPPORT graph $H=(V(H),E(H))$

• Problem instance is a subgraph $G=(V,E)$ of $H$

• Two phases:
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing (e.g. MAC-address)
- Original structure given as the SUPPORT graph $H=(V(H),E(H))$
- Problem instance is a subgraph $G=(V,E)$ of $H$

- Two phases:
  1. Preprocessing: compute any function on $H$ and store output locally
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
  - E.g. MAC-address

- Original structure given as the SUPPORT graph \( H=(V(H),E(H)) \)

- Problem instance is a subgraph \( G=(V,E) \) of \( H \)

- Two phases:
  1. Preprocessing: compute any function on \( H \) and store output locally
  2. Solve problem on \( G \) in LOCAL model with preprocessed outputs
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
  - E.g. MAC-address
- Original structure given as the SUPPORT graph $H = (V(H), E(H))$
- Problem instance is a subgraph $G = (V, E)$ of $H$

Two phases:

1. Preprocessing: compute any function on $H$ and store output locally
2. Solve problem on $G$ in LOCAL model with preprocessed outputs
   - Runtime: Number of $t$ rounds in (2), denoted as SUPPORTED($t$)
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing
  - E.g. MAC-address

- Original structure given as the SUPPORT graph \( H=(V(H),E(H)) \)

- Problem instance is a subgraph \( G=(V,E) \) of \( H \)

- Two phases:
  1. Preprocessing: compute any function on \( H \) and store output locally
  2. Solve problem on \( G \) in LOCAL model with preprocessed outputs.
     - Runtime: Number of \( t \) rounds in (2), denoted as SUPPORTED(t)
Does the SUPPORTED Model make everything easy?

- Task: Leader election ($\Theta$(diameter) runtime in LOCAL model)
  - Easy if $G=H$: precompute leader, 0 rounds
  - But for different $G$:
    - We need to compute a leader for each connected component of $G$!
      - Component has no leader? Re-elect ☹
      - Component has multiple leaders? Re-elect ☹
      - Components can have asymptotically same diameter ☹
  - SUPPORTED model does not provide a “silver bullet”
    - Not even for the *active* variant
Maybe even useless in general?
Maybe even useless in general?

- Let the support graph $H$ be a complete graph
Maybe even useless in general?

- Let the support graph $H$ be a complete graph
- What sort of meaningful information (for $G$) can we precompute?
Maybe even useless in general?

- Let the support graph $H$ be a complete graph
- What sort of meaningful information (for $G$) can we precompute?
  - Upper bound on ID-space / network size...?
Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size...
  - Problem: G can be arbitrary
Maybe even useless in general?

• Let the support graph $H$ be a complete graph

• What sort of meaningful information (for $G$) can we precompute?
  ◦ Upper bound on ID-space / network size...?
  ◦ Problem: $G$ can be arbitrary

• For example, if a SUPPORTED algorithm has polylogarithmic runtime
  ◦ $\exists$ LOCAL algorithm with constant factor overhead
Maybe even useless in general?

- Let the support graph $H$ be a complete graph
- What sort of meaningful information (for $G$) can we precompute?
  - Upper bound on ID-space / network size...?
  - Problem: $G$ can be arbitrary
- For example, if a SUPPORTED algorithm has polylogarithmic runtime
  - $\exists$ LOCAL algorithm with constant factor overhead

Idea: simulate that support graph $H$ is a complete graph
Maybe even useless in general?

• Let the support graph $H$ be a complete graph

• What sort of meaningful information (for $G$) can we precompute?
  ◦ Upper bound on ID-space / network size...?
  ◦ Problem: $G$ can be arbitrary

• For example, if a SUPPORTED algorithm has polylogarithmic runtime
  ◦ $\exists$ LOCAL algorithm with constant factor overhead

In *active* model: Congested Clique
But: Restricted Graph Families are Useful 😊

- Real topologies are usually not complete graphs

- Case study: planar graphs
  - Remain planar under edge deletions
  - Are 4-colorable

„Geloeste und ungeloste Mathematische Probleme aus alter und neuer Zeit" by Heinrich Tietze
http://www.math.harvard.edu/~knill/graphgeometry/faqg.html
Case Study: Dominating Set

- Task: Find subset D of nodes s.t. every node
  - Has a neighbor in D or is in D

- Can we pre-compute?
  - A bad one yes: everyone in D!
  - But not an optimal one!
    - Graph can look very different
Case Study: Minimum Dominating Set in Planar Graphs
Case Study: Minimum Dominating Set in Planar Graphs

• (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
Case Study: Minimum Dominating Set in Planar Graphs

- $(1+\delta)$-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?
Case Study: Minimum Dominating Set in Planar Graphs

• (1+\(\delta\))-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  ◦ But maybe in the SUPPORTED model?

• Let’s analyze their LOCAL algorithm:
Case Study: Minimum Dominating Set in Planar Graphs

• (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  ◦ But maybe in the SUPPORTED model?

• Let’s analyze their LOCAL algorithm:
  ◦ Find weight-appropriate pseudo-forest [constant time ☺]
Case Study: Minimum Dominating Set in Planar Graphs

• (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  ◦ But maybe in the SUPPORTED model?

• Let’s analyze their LOCAL algorithm:
  ◦ Find weight-appropriate pseudo-forest [constant time 😊]
Case Study: Minimum Dominating Set in Planar Graphs

• (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  ◦ But maybe in the SUPPORTED model?

• Let’s analyze their LOCAL algorithm:
  ◦ Find weight-appropriate pseudo-forest [constant time 😊]
  ◦ 3-color pseudo-forest [non-constant time ☹️]
Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let’s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]
Case Study: Minimum Dominating Set in Planar Graphs

- (1+\(\delta\))-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let’s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time 😞]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]

SUPPORTED speed-up:
1) precompute 4-coloring
2) reduce 4-colored pseudo-forest to 3 colors in 2 rounds
Case Study: Minimum Dominating Set in Planar Graphs

• $(1+\delta)$-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  ◦ But maybe in the SUPPORTED model?

• Let’s analyze their LOCAL algorithm:
  ◦ Find weight-appropriate pseudo-forest [constant time ☺]
  ◦ 3-color pseudo-forest [non-constant time ☹][constant time SUPPORTED model ☺]
  ◦ Run clustering/optimization algorithms on components of constant size [constant time ☺]
Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let’s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time 😊]
  - 3-color pseudo-forest [non-constant time 😊][constant time SUPPORTED model 😊]
  - Run clustering/optimization algorithms on components of constant size [constant time 😊]

- Also works for O(1)-genus graphs [extending work of Akhoondian Amiri et al.]
Case Study: Minimum Dominating Set in Planar Graphs

- 
  - (1+\(\delta\))-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let’s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time 😊]
  - 3-color pseudo-forest [non-constant time 😐][constant time SUPPORTED model 😊]
  - Run clustering/optimization algorithms on components of constant size [constant time 😊]

- Also works for O(1)-genus graphs [extending work of Akhoondian Amiri et al.]
  - Also for planar graphs for maximum independent set & maximum matching
Further Results in the Active SUPPORTED Model
Further Results in the *Active SUPPORTED* Model

Use all edges of $H$ for communication
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
Further Results in the Active SUPPORTED Model

• Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*\text{poly log } n)): e.g. MIS in SUPPORTED(poly log n)
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t\cdot\text{poly log } n)): e.g. MIS in SUPPORTED(poly log n)

Use all edges of H for communication

Best LOCAL algorithm: \(2^{O(\sqrt{\log n})}\)
Further Results in the Active SUPPORTE

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*\text{poly log } n))

Through known connections, this general derandomization leads to better deterministic and randomized distributed algorithms for numerous problems. A sampling of end-results includes poly(log n)-round deterministic algorithms for MIS, \(\Delta + 1\) coloring, the Lovász Local Lemma\(^3\), hypergraph splitting, and defective coloring. These also lead to substantially improved randomized algorithms, including a poly(log log n)-time randomized \(\Delta + 1\) coloring [CLP18] and a poly(log log n)-time randomized algorithm for Lovász Local Lemma in constant degree graphs [GHK18].
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)

Best LOCAL algorithm: $2^{O(\sqrt{\log n})}$
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)

Also works in passive model:
SLOCAL(t) → SUPPORTED(Δ^O(t))

Best LOCAL algorithm:
2^O(\sqrt{\log n})
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(\(O(t \cdot \text{poly log } n)\)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

Also works in *passive* model:
\[ \text{SLOCAL}(t) \rightarrow \text{SUPPORTED}(\Delta^{O(t)}) \]

Best LOCAL algorithm:
\[ 2^{O\left(\sqrt{\log n}\right)} \]
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
  - e.g. network size, restricted H, known inputs...

Also works in *passive model*: SLOCAL(t) → SUPPORTED(Δ^{O(t)})

Best LOCAL algorithm:

\[ 2^{O(\sqrt{\log n})} \]
Further Results in the Active SUPPORTED Model

• Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
    e.g. network size, restricted H, known inputs..

• Locally Checkable Labelings LCL:

Also works in *passive* model:
SLOCAL(t) \rightarrow SUPPORTED(\Delta^{O(t)})

Best LOCAL algorithm:
2^{O(\sqrt{\log n})}
Further Results in the Active SUPPORTED Model

• Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
    e.g. network size, restricted H, known inputs..

• Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(\(O(t \cdot \text{poly log } n)\)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in \(O(1)\) in the SUPPORTED model

- Use all edges of \(H\) for communication

- Also works in \textit{passive} model:
  \[ \text{SLOCAL}(t) \rightarrow \text{SUPPORTED}(\Delta^{O(t)}) \]

- Best LOCAL algorithm:
  \[ 2^{O(\sqrt{\log n})} \]

- Also works without the \textit{active} model

- e.g. network size, restricted \(H\), known inputs..
Further Results in the Active SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
    - e.g. network size, restricted H, known inputs..

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

- Optimization problem: Maximum Independent Set, of size α(G)

Use all edges of H for communication

Also works in passive model: SLOCAL(t) → SUPPORTED(Δ^O(t))

Best LOCAL algorithm: \(2^O(\sqrt{\log n})\)

Also works without the active model
Further Results in the Active SUPPORTED Model

• Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
  
  - e.g. network size, restricted H, known inputs..

• Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

• Optimization problem: Maximum Independent Set, of size $\alpha(G)$
  - Set of size $(\alpha(G) - \varepsilon)n$ in $O(\log_{1+\varepsilon} n)$, respectively $(1+\varepsilon)$ approximation if maximum degree $\Delta$ constant

Best LOCAL algorithm:

$2^{O(\sqrt{\log n})}$

Also works in passive model:

SLOCAL(t) $\rightarrow$ SUPPORTED($\Delta^{O(t)}$)

Also works without the active model

Use all edges of H for communication
Further Results in the Active SUPPORTED Model

• Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t*poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question
    - e.g. network size, restricted H, known inputs..

• Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

• Optimization problem: Maximum Independent Set, of size \( \alpha(G) \)
  - Set of size \((\alpha(G)-\varepsilon)n\) in \(O(\log^{1+\varepsilon} n)\), respectively \((1+\varepsilon)\) approximation if maximum degree \(\Delta\) constant
  - Cannot be approximated by \(o(\Delta/\log \Delta)\) in time \(o(\log_\Delta n)\) in the active SUPPORTED model
Bigger Open Question/Opportunity

How to efficiently leverage such preprocessing/distributed computing to efficiently scale controllers (and network updates)?

So far largely unexplored

So let's get back things we know about 😊 Congestion and network functions?
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
Congestion?

• “Stronger” consistency constraint: also do not violate link capacities
Congestion?

• "Stronger" consistency constraint: also do not violate link capacities
  ◦ Flow size: 1
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1

Round 1
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1

Round 0
Congestion?

• “Stronger” consistency constraint: also do not violate link capacities
  ◦ Flow size: 1

---

Round 1
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1

Round 2
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1

Round 0
Congestion?

• “Stronger” consistency constraint: also do not violate link capacities
  ◦ Flow size: 1, 1

Round 1
**Congestion?**

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1

---

**Round 2**
Congestion?

• “Stronger” consistency constraint: also do not violate link capacities
  ◦ Flow size: 1, 1

Round 0
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1
### Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: $1, 1$

---

**Round 2**
Congestion?

• “Stronger” consistency constraint: also do not violate link capacities
  ◦ Flow size: 1, 1

Round 3
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1

Round 4
Complexity of Avoiding Congestion?

- NP-hard already for 2 unit size flows on general graphs

- Also NP-hard on acyclic graphs for $k$ flows
  - But can be FPT characterized for $k$ flows on acyclic graphs: $O\left(2^{O(k \log k)} |G| \right)$
  - In other words, linear runtime for constant $k$ on DAGs

- For just 2 unit size flows (where old/new individually is a DAG): Optimal schedule in P (NPH for 6)

*Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP’18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking’19*
Complexity of Avoiding Congestion?

- NP-hard already for 2 unit size flows on general graphs

- For just 2 unit size flows (where old/new individually is a DAG): Optimal schedule in P (NPH for 6)

*Congestion-Free Rerouting of Flows on DAGs.* S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP’18

*On Polynomial-Time Congestion-Free Software-Defined Network Updates.* AA, D., M. Parham, S., S. W., Networking’19
• For just 2 unit size flows (where old/new \textit{individually} is a DAG): Optimal schedule in P (NPH for 6)

\textit{Congestion-Free Rerouting of Flows on DAGs.} S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP’18

\textit{On Polynomial-Time Congestion-Free Software-Defined Network Updates.} AA, D., M. Parham, S., S. W., Networking’19
Complexity of Avoiding Congestion?

• NP-hard already for 2 unit size flows on general graphs

• Also NP-hard on acyclic graphs for 6 flows
  ◦ But can be FPT characterized for $k$ flows on acyclic graphs: $O\left(2^{O(k \log k)} |G|\right)$
    - In other words, linear runtime for constant $k$ on DAGs

• For just 2 unit size flows (where old/new individually is a DAG): Optimal schedule in P

Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP’18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking’19
Take a Step Back: No Loops and a Firewall

Which forwarding rule to update first?
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall

![Diagram showing network with nodes s and d, and a firewall symbol]
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall
Take a Step Back: No Loops and a Firewall

However: If packets must either take the new or the old path (and no mix), then polynomial-time solvable (Cerný et al., DISC 2016)

Satisfy both & can conflict! & NP-hard!

**Different model: “tagged” Flows**

- Identified by a “tag“ in the packet header, update via
  - Install new tag rules
  - Switch from tag to tag at source
If we move a flow, will there be congestion?
If we move a flow, will there be congestion?

- How do we move a flow $F$? Usually: 2-phase commit: [Reitblatt et al., SIGCOMM’12]
If we move a flow, will there be congestion?

• How do we move a flow $F$? Usually: 2-phase commit:
  ◦ Deploy new flow rules $F'$
If we move a flow, will there be congestion?

• How do we move a flow \( F \)? Usually: 2-phase commit:
  ◦ Deploy new flow rules \( F' \)
  ◦ Change packet tag at source from \( F \) to \( F' \)
If we move a flow, will there be congestion?

- How do we move a flow $F$? Usually: 2-phase commit:
  - Deploy new flow rules $F'$
  - Change packet tag at source from $F$ to $F'$

Can also be implemented by proof-labeling techniques

"hand holding"?

Go backwards with distance information

Respects network functions!
If we move a flow, will there be congestion?

- How do we move a flow $F$? Usually: 2-phase commit:
  - Deploy new flow rules $F'$
  - Change packet tag at source from $F$ to $F'$
  - Clean-up of old rules
If we move a flow, will there be congestion?

• How do we move a flow $F$? Usually: 2-phase commit:
  ◦ Deploy new flow rules $F'$
  ◦ Change packet tag at source from $F$ to $F'$
  ◦ Clean-up of old rules

• First check:
  ◦ Is the new network state without congestion?
  ◦ Easy 😊 (flow size versus capacity)

• Is that it?
A Small Sample Network

Unit size flows and capacities
Green wants to send as well

Unit size flows and capacities
Congestion!

Unit size flows and capacities
This would work

Unit size flows and capacities
So let's go back

Unit size flows and capacities
But Red is a bit Slow..
Congestion Again!

Unit size flows and capacities
So let's go Back ...

Round 0 (old)

Unit size flows and capacities
First, Red switches
Then, Blue ...
And then, Green ...
Done

Round 3 (new)
How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition
How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition

Intermediate flow allocations not restricted to *old* and *new*:

- NP-hard already for just 2 unit size flows

*On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds* (Foerster, NCA 2017)
How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition

Intermediate flow allocations not restricted to *old* and *new*:

- NP-hard already for just 2 unit size flows

- Is the problem at least in NP?

*Some flows might need to move back and forth repeatedly.*

---

*On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds* (Foerster, NCA 2017)
How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

• NP-hard via reduction from Partition

Intermediate flow allocations not restricted to *old* and *new*:

• NP-hard already for just 2 unit size flows

Not clear if the problem is in NP! (It is known to be in EXPTIME)

---

*On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds* (Foerster, NCA 2017)
Consistent Migration of Splittable Flows

Idea: Flows can be on the old or new route w.r.t. an update.

For all edges: $\sum_{vF} \max(\text{old}, \text{new}) \leq \text{capacity}$

No ordering exists ($2/3 + 2/3 > 1$)
Consistent Migration of Splittable Flows

Approach of SWAN*: use slack $x$ (i.e., %)

Here $x = \frac{1}{3}$

Move slack $x \Rightarrow \lfloor \frac{1}{x} \rfloor - 1$ staged partial moves
Consistent Migration of Splittable Flows

Approach of SWAN: use slack $x$ (i.e., %)

Here $x = 1/3$

Move slack $x \Rightarrow \lfloor 1/x \rfloor - 1$ staged partial moves

Update 1 of 2
Consistent Migration of Splittable Flows

Approach of SWAN: use slack $x$ (i.e., %)

Here $x = 1/3$

Move slack $x \Rightarrow [1/x] - 1$ staged partial moves

Update 1 of 2
Consistent Migration of Splittable Flows

Approach of SWAN: use slack $x$ (i.e., %)

Here $x = 1/3$

Move slack $x \Rightarrow [1/x] - 1$ staged partial moves

Update 2 of 2
Consistent Migration of Splittable Flows

No slack on flow edges?
Consistent Migration of Splittable Flows

Alternate routes?
Consistent Migration of Splittable Flows

Think: variable swapping of $b$ & $g$

1. $x := b$, 2. $b := g$, 3. $g := x$
Consistent Migration of Splittable Flows

Think: variable swapping of $b$ & $g$

1. $x := b$, 2. $b := g$, 3. $g := x$
Consistent Migration of Splittable Flows

Think: variable swapping of $b$ & $g$

1. $x := b$, 2. $b := g$, 3. $g := x$
Consistent Migration of Splittable Flows

SWAN: LP-approach with binary search
1 update? 2 updates? 4 updates? ...

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
Consistent Migration of Splittable Flows

SWAN: LP-approach with binary search
1 update? 2 updates? 4 updates? ...
Consistent Migration of Splittable Flows

**SWAN**: LP-approach with binary search

$\Theta(1/\varepsilon)$ updates 😞
Consistent Migration of Splittable Flows

Can we decide in \textit{(polynomial)} time?

Flow migration

“Halting Problem”

LP

Yes

No
To Slack or not to Slack?

Slack of $x$ on all flow edges?

$[1/x] - 1$ updates
To Slack or not to Slack?

What if not?
Try to create slack
To Slack or not to Slack?

Combinatorial approach

Augmenting paths
Combinatorial Approach

Move single commodities at a time
Combinatorial Approach

Where to increase flow?
Combinatorial Approach

Where to push back flow?

Diagram showing a network with nodes and edges, illustrating the concept of flow and its direction.
Combinatorial Approach

Resulting residual network
Combinatorial Approach

We found an augmenting path $\Rightarrow$ create slack on $e$
High-level Algorithm Idea

• No slack on flow edges? Find augmenting paths
  ◦ On both initial and desired state (updates can be performed in reverse)
  ◦ Success? Use SWAN method to migrate

• Can’t create slack on some flow edge?
  ◦ Consistent migration impossible
    By contradiction (else augmenting paths would create slack)

• Runtime: $O(Fm^3)$
  ◦ ($F$ being #commodities, $m$ being #edges)
Open problems for scheduling flow migration

• What happens when we can pick the new paths?
  ◦ Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

Maybe surprisingly: If the new flows fit in somehow, we can migrate consistently!
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of each flow: 1
capacity of links: 1
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)
size of flows: 1, 2, 1, 1
capacity of links: 1 (or marked)
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)
size of flows: 1+3=4, 2, 1, 1
capacity of links: 1 (or marked)
• Flows end up at the wrong destination!

• So let’s stick with augmenting flows that don’t mix destinations
size of each flow: 1
capacity of each links: 1
size of each flow: 1
capacity of each links: 1
size of each flow: 1
capacity of each links: 1
size of each flow: 1
capacity of each links: 1
“it is unlikely that similar techniques can be developed for constructing multicommodity flows”

[Hu, 1963]

size of each flow: 1

capacity of each links: 1
Open Problems for scheduling flow migration

• What happens when we can pick the new paths?
  ◦ Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

• Unsplittable flow migration:
  ◦ In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  ◦ ”Interesting“ polynomial cases?
Open Problems for scheduling flow migration

• What happens when we can pick the new paths?
  ◦ Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

• Unsplittable flow migration:
  ◦ In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  ◦ ”Interesting“ polynomial cases?

Maybe surprisingly:
If the new flows fit in somehow, we can migrate consistently!

Maybe further development needs better understanding of augmenting flows?

More open questions and specifics:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019
Open Problems for scheduling flow migration

• What happens when we can pick the new paths?
  ◦ Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

• Unsplittable flow migration:
  ◦ In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  ◦ ”Interesting“ polynomial cases?

• What happens when considering Link Latency?
The Impact of Latency (in Testbed)

- ping of old path
- ping of new path
The Impact of Latency (in Testbed)

The diagram illustrates the impact of latency in a testbed setting. The x-axis represents time in seconds (sec), ranging from 10 to 30. The y-axis represents RTT in milliseconds (ms), ranging from 0 to 12. Two ping measurements are shown: one for the old path (orange line) and one for the new path (red line). The graph shows how the latency changes over time, with the new path experiencing a temporary increase in latency but eventually stabilizing.
The Impact of Latency (in Testbed)

![Graph showing the impact of latency in a testbed.](image)

- ping of old path
- ping of new path

**Diagram:**
- Time (sec): 10 to 30
- RTT (ms): 2 to 12

**Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02**

257
The Impact of Latency (in Testbed)

- **UDP**

![Graph showing latency over time](image)

- **sec**
- **RTT in ms**

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
The Impact of Latency (in Testbed)

TCP

RTT in ms

sec

0 5 10 15 20 25 30

10 15 20 25 30

capacity = 1
size = 1
high latency
low latency

The Impact of Latency (in Testbed)

Because there is also work that focuses on better time synchronization, notably by Mizrahi et al. 
https://sites.google.com/site/timedsdn/
CDF of the Congestion Duration
Recap

• Common (coarse-grained) model:
  ◦ Sum for all flows: \( \text{Max( old flow rules, new flow rules )} \) does not violate capacity [SWAN, SIGCOMM’13]
  ◦ Decidable in polynomial time [Brandt et al., INFOCOM’16]
    - For unsplittable flows: NP-hard already for 2 flows

• Does not capture congestion due to flows congesting themselves!
  ◦ How hard?
How hard?

- Unit latencies and splittable flow of unit size:
  - Already NP-hard for a single flow!

Find a temporary path to offload parts of the flow
Recap of the last few slides

- Common (coarse-grained) model:
  - Sum for all flows: Max( old flow rules, new flow rules ) does not violate capacity [SWAN, SIGCOMM’13]
  - Decidable in polynomial time [Brandt et al., INFOCOM’16]
    - For unsplittable flows: NP-hard already for 2 flows

- Does not capture congestion due to flows congesting themselves!
  - How hard?
    - NP-hard for unit size/latency and splittable flows 😊

- How to fix?
  - Treat old and new flow rules as separate flows?
Old and New as Different Entities

- Idea: We can handle interplay between different flows
  - Handle old and new as different flows?
    - Prevents such congestion in popular approaches, eg SWAN, Dionysus, zUpdate etc.
Relax And Take it Easy!
Relax for Polynomial-Time Lossless Updates

• Idea: Relax the problem formulation
  ◦ Be congestion-free for any set of latencies
    - (i.e., adversary may change latencies at any time)

• Now congestion-free intermediate steps become reversible

• Rough structure of the algorithm (for splittable flows):
  ◦ Take old (new) state, reach intermediate state where critical set of edges have spare capacity
    - Not possible? No congestion-free migration possible.
Recap of the last few slides

• Common (coarse-grained) model:
  ◦ Sum for all flows: Max( old flow rules , new flow rules ) does not violate capacity [SWAN, SIGCOMM’13]
  ◦ Decidable in polynomial time [Brandt et al., INFOCOM’16]
    - For unsplittable flows: NP-hard already for 2 flows

• Does not capture congestion due to flows congesting themselves!
  ◦ NP-hard for unit size/latency and splittable flows 😞

• By relaxing latency constraints:
  ◦ Again polynomial-time decidable

• Interestingly: Augmenting flow idea still works even without relaxing latency constraints!
Open Problems and Outlook in General

• Various algorithmic and complexity questions for a centralized controller
  ◦ See recent survey

• First connections to more classic distributed computing topics are made
  ◦ Proof-labeling
    - Very basic right now, how to build more complex/efficient systems?

• Maybe the bigger question: How to properly distribute the centralized controller
  ◦ Opportunity: The SUPPORTED model / preprocessing
Some References

- Local Checkability, No Strings Attached: (A)cyclicity, Reachability, Loop Free Updates in SDNs. Tycho Foerster, Stefan Schmid, and Jukka Suomela. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Vienna, Austria, September 2019.
- Some References

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates, 19-08-02
Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates

Klaus-T. Foerster