Managing Dynamic Networks:
Distributed or Centralized Control?

Roger Wattenhofer

ETH Zurich – Distributed Computing Group
“On Distributed Communications” (1964)
"On Distributed Communications" (1964)

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.
"On Distributed Communications" (1964)

1. Node & Edge Destruction
2. Distributed Routing

Fig. 1—(a) Centralized. (b) Decentralized. (c) Distributed networks.
people stopped worrying about the bomb!
Today: Inter-Data Center WANs

Think: Google, Amazon, Microsoft
Problem: Typical Network Utilization

- Utilization peak before rate adaptation
- > 50% peak reduction
- Mean utilization
Problem: Typical Network Utilization

![Graph showing network utilization over time with background and non-background traffic](image-url)

- **Utilization**
- **Time [1 Day]**

- **Background traffic**
- **Non-background traffic**

**Mean**
Problem: Typical Network Utilization

- Background traffic
- Non-background traffic

Utilization

Time [1 Day]

peak before rate adaptation

peak after rate adaptation

> 50% peak reduction
Another Problem: Online Routing Decisions

flow arrival order: A, B, C

each link can carry at most one flow (in both directions)
Software Defined Networks (SDNs)
Dealing with Network Dynamics: The SWAN Project

SWAN controller

[global optimization for high utilization]

Hosts

WAN switches

[rate limiting]

[topology, traffic]

[forwarding plane update]

[rate limiting]

network configuration

rate allocation

traffic demand
Solution: Multicommodity Flow LP

Maximize throughput of flows $f_i$

$$\max \sum_i f_i$$

Flow less than demand $d_i$

$$0 \leq f_i \leq d_i$$

Flows less than capacity $c(e)$

$$\sum_i f_i(e) \leq c(e)$$

Flow conservation on inner nodes

$$\sum_u f_i(u, v) = \sum_w f_i(v, w)$$

Flow definition on source, destination

$$\sum_v f_i(s_i, v) = \sum_u f_i(u, t_i) = f_i$$
Network Dynamics
Problem: Consistent Updates

Initial state

Target state
Capacity-Consistent Updates

• Not directly, but maybe through intermediate states?

• Solution: Leave a fraction $s$ slack on each edge, less than $1/s$ steps

• Example: Slack = $1/3$ of link capacity,
Example: Slack = 1/3 of link capacity
Capacity-Consistent Updates

Alternatively: Try whether a solvable LP with $k$ steps exist, for $k = 1, 2, 3 \ldots$
(Sum of flows in steps $j$ and $j + 1$, together, must be less than capacity limit)

Only growing flows

$$f_i^0 \leq f_i^k$$

Flow less than capacity

$$\sum_i \max \left( f_i^j (e), f_i^{j+1} (e) \right) \leq c(e)$$

Flow conservation on inner nodes

$$\sum_u f_i^j (u, \nu) = \sum_w f_i^j (\nu, w)$$

Flow definition on source, destination

$$\sum_v f_i^j (s_i, \nu) = \sum_u f_i^j (u, t_i) = f_i^j$$

[Hong et al., SIGCOMM 2013]
Prototype Evaluation

Traffic: (∀DC-pair) 125 TCP flows per class

High utilization
SWAN’s goodput:
98% of an optimal method

Flexible sharing
Interactive protected;
background rate-adapted
Data-driven Evaluation of 40+ DCs

Utilization

- SWAN
- SWAN w/o Rate Control
- MPLS TE
Another Problem: Straggler Switches

CDF of 100 updates on a switch, in seconds

Dionysus: Make updates dynamic, i.e., work around straggling switches

[Jin et al., SIGCOMM 2014]
Yet Another Problem: Memory Limits at Switches

Surprisingly, with memory limits, updates are difficult (NP-complete).
Example: We want to swap all flows between two switches $u$ and $v$.
Each switch has capacity $c$, and memory limit $k$.

\[ \frac{c}{2} \leq k - 1 \]

[Jin et al., SIGCOMM 2014]
Updating Dynamic Networks:

A Bigger Picture?
### Consistency Space

<table>
<thead>
<tr>
<th></th>
<th>None</th>
<th>Self</th>
<th>Downstream subset</th>
<th>Downstream all</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eventual consistency</strong></td>
<td>Always guaranteed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Drop freedom</strong></td>
<td>Impossible</td>
<td>Add before remove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory limit</strong></td>
<td>Impossible</td>
<td>Remove before add</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Loop freedom</strong></td>
<td>Impossible</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>Packet coherence</strong></td>
<td>Impossible</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Bandwidth limit</strong></td>
<td>Impossible</td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

[Mahajan & W, HotNets 2013]
Example

SDN Controller
Example

SDN Controller

[Reitblatt et al., SIGCOMM 2012]
Dependencies

Version Numbers

“Better” Solution

+ stronger packet coherence
– version number in packets
– switches need to store both versions
Minimum SDN Updates?
Minimum Updates: Another Example

\[\begin{align*}
&\text{or} \\
&\begin{align*}
&u \\
v &\quad w \\
d &\quad w \\
u &\quad v \\
w &\quad w \\
d
\end{align*}
\end{align*}\]
No node can improve without hurting another node.
Minimal Dependency Forest

Next: An algorithm to compute minimal dependency forest.
Algorithm for Minimal Dependency Forest

- Each node in one of three states: old, new, and limbo (both old and new)
Algorithm for Minimal Dependency Forest

- Each node in one of three states: old, new, and limbo (both old and new)
- Originally, destination node in new state, all other nodes in old state
- Invariant: No loop!
Algorithm for Minimal Dependency Forest

Initialization

- **Old** node \( u \): No loop* when adding new pointer, move node to limbo!
- This node \( u \) will be a root in dependency forest

*Loop Detection: Simple procedure, see next slide
Loop Detection

• Will a new rule $u.new = v$ induce a loop?
  – We know that the graph so far has no loops
  – Any new loop must contain the edge $(u,v)$

• In other words, is node $u$ now reachable from node $v$?

• Depth first search (DFS) at node $v$
  – If we visit node $u$: the new rule induces a loop
  – Else: no loop
Algorithm for Minimal Dependency Forest

- **Limbo node** $u$: Remove *old* pointer (move node to *new*)
- Consequence: Some *old* nodes $v$ might move to limbo!
- Node $v$ will be child of $u$ in dependency forest!

![Diagram showing the algorithm](image-url)
Algorithm for Minimal Dependency Forest

Process terminates

- You can always move a node from limbo to new.
- Can you ever have old nodes but no limbo nodes? No, because...

...one can easily derive a contradiction!
For a given \textit{consistency property}, what is the \textit{minimal dependency} possible?
### Consistency Space

<table>
<thead>
<tr>
<th></th>
<th>None</th>
<th>Self</th>
<th>Downstream subset</th>
<th>Downstream all</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Eventual consistency</strong></td>
<td>Always guaranteed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Drop freedom</strong></td>
<td>Impossible</td>
<td>Add before remove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Impossible</td>
<td>Remove before add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Memory limit</strong></td>
<td>Impossible</td>
<td></td>
<td>Rule dep. forest</td>
<td>Rule dep. tree</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Loop freedom</strong></td>
<td>Impossible</td>
<td></td>
<td></td>
<td></td>
<td>Global ver. numbers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Packet coherence</strong></td>
<td>Impossible</td>
<td></td>
<td>Per-flow ver. numbers</td>
<td>Global ver. numbers</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Bandwidth limit</strong></td>
<td>Impossible</td>
<td></td>
<td></td>
<td></td>
<td>Staged partial moves</td>
</tr>
</tbody>
</table>

[Mahajan & W, HotNets 2013]
Multiple Destinations using Prefix-Based Routing

- No new “default” rule can be introduced without causing loops
- Solution: Rule-Dependency Graphs!
- Deciding if simple update schedule exists is hard!
Breaking Cycles

Insert \( u \rightarrow w \)  
Remove \( u \rightarrow v \)  
Insert \( v \rightarrow u \)  
Remove \( v \rightarrow w \)  

Insert at \( w \): dest \( v: w \rightarrow v \)  
Remove at \( w \): dest \( v: w \rightarrow v \)
Summary

![Graph and Table]

<table>
<thead>
<tr>
<th>Eventual consistency</th>
<th>None</th>
<th>Self</th>
<th>Downstream subset</th>
<th>Downstream all</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always guaranteed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Drop freedom</td>
<td>Impossibly</td>
<td>Add before remove</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory limit</td>
<td>Impossibly</td>
<td>Remove before add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop freedom</td>
<td>Impossibly</td>
<td>Rule dep. forest</td>
<td>Rule dep. tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Packet coherence</td>
<td>Impossibly</td>
<td>Per-flow ver. numbers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bandwidth limit</td>
<td>Impossibly</td>
<td>Staged partial moves</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Thank You!

Questions & Comments?

www.disco.ethz.ch