Network Updates

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

- But what happens if the WAN is controlled by a single entity?
  - Examples: Microsoft & Amazon & Google ...
    They spend hundreds of millions of dollars per year

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)

- Centralized controller implemented with replication, e.g. Paxos
Dependencies

Version Numbers

“Better” Solution

Minimum SDN Updates?

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

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!

Loop Detection

- Will a new rule $u \rightarrow new = v$ induce a loop? $u \rightarrow new \rightarrow v$
  - 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

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

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
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!

Main Contribution

For a given consistency property, what is the minimal dependency possible?

Consistency Space

<table>
<thead>
<tr>
<th>Breach limit</th>
<th>Core limit</th>
<th>Downstream add</th>
<th>Downstream used</th>
<th>Global</th>
</tr>
</thead>
<tbody>
<tr>
<td>Always</td>
<td>Always</td>
<td>Cannot</td>
<td>Cannot</td>
<td>Global</td>
</tr>
<tr>
<td>Impossible</td>
<td>Impossible</td>
<td>Add before remove</td>
<td>Remove before add</td>
<td>Cannot</td>
</tr>
<tr>
<td>Impossible</td>
<td>Impossible</td>
<td>None</td>
<td>None</td>
<td>Global</td>
</tr>
<tr>
<td>Impossible</td>
<td>Impossible</td>
<td>None</td>
<td>None</td>
<td>Global</td>
</tr>
</tbody>
</table>

It’s not just how to compute new rules.

It is also how to gracefully get from current to new configuration, respecting consistency.
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

Update DAG

- Insert rule \( p \) at node \( u \)
- Insert rule \( q \) at node \( w \)
- Wait 10s
- Logical OR
- Remove rule \( q \) at node \( x \)
- Remove rule \( s \) at node \( v \)
- Insert rule \( r \) at node \( y \)
Architecture

Routing policy
Rule generator → New rules → Update plan generator → Update DAG → Plan optimizer and executor

Consistency property

Network characteristics

Breaking Cycles

Are Minimal Dependencies Good?

...it depends

(But Plan optimizer and executor will fix it.)
Real Application: Inter-Data Center WANs

Think: Google, Amazon, Microsoft

Problem: Typical Network Utilization

~ 50% peak reduction

Peak before rate adaptation

Mean

Problem: Typical Network Utilization

Background traffic

Mean

Non-background traffic

Peak before rate adaptation

~ 50% peak reduction

Peak after rate adaptation

Mean

Time [1 Day]
Another Problem: Online Routing Decisions

flow arrival order: A, B, C
each link can carry at most one flow (in both directions)

Algorithms?

- Priority classes (2-3)
  - Allocate highest priority first

- Solve with multi-commodity flow (LP) within each class
  - Flows are splittable
    - Well understood, fast enough for our input (seconds)

- But: Within a priority class we want max min fairness \( f_1 \geq f_2, \max f'' \)
  - Definition: Make nobody richer at cost of someone poorer
  - Works, but now one has to solve linearly many LPs, which is too slow (hours)
  - A perfect example of algorithm engineering?

- Solution: Fairness approximation!

Multicommodity Flow LP

Maximize throughput
\[ \max \sum f_i \]

Flow less than demand
\[ 0 \leq f_i \leq d_i \]

Flow less than capacity
\[ \sum f_i(e) \leq c(e) \]

Flow conservation on inner nodes
\[ \sum_{u} f_i(u, v) = \sum_{w} f_i(v, w) \]

Flow definition on source, destination
\[ \sum_{v} f_i(s_i, v) = \sum_{u} f_i(u, t_i) = f_i \]

The SWAN Project

[global optimization for high utilization]

SWAN controller

traffic demand
rate allocation
network configuration
topology, traffic

Hosts

hosts, switches

forwarding plane update
Approximated max-min fairness

- In theory, this process is $(1 + \varepsilon)$ competitive
- In practice, with $\varepsilon = 1$, only 4% of flows deviate over 5% from their fair share

Fairness: SWAN vs. MPLS TE

Flows sorted according to demand

Problem: Consistent Updates
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

Init. state
\[ A = 2/3 \quad B = 2/3 \]

Target state
\[ B = 1/3 \]

Evaluation platforms

- Prototype
  - 5 DCs across 3 continents
  - 10 switches

- Data-driven evaluation
  - 40+ DCs across 3 continents
  - 80+ switches

Only growing flows
\[ f_i^u \leq f_i^k \]

Flow less than capacity
\[ \sum_{e} \max\left(f_i^j(e), f_i^{j+1}(e)\right) < c(e) \]

Flow conservation on inner nodes
\[ \sum_{u} f_i^j(u, v) = \sum_{w} f_i^j(v, w) \]

Flow definition on source, destination
\[ \sum_{v} f_i^j(s_i, v) = \sum_{u} f_i^j(u, t_i) = f_i^j \]
Time for One Network Update

1. Compute allocation & rule change plan
2. Compute congestion-controlled plan
3. Wait for rate limiting
4. Change switch rules
5. Rate limiting

Update time [s]

Prototype Evaluation

Dips due to rate adaptation
Optimal line

Traffic: (VDC-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

Summary

Routing policy
Consistency property
Network characteristics

Plan optimizer and executor
References

- Introducing consistent network updates was done in Mark Reitblatt et. al., SIGCOMM 2012
- For minimal loop-free updates and more see Ratul Mahajan et. al., HotNets 2013
- Deciding if a simple update schedule exists is hard was proven in Laurent Vanhaecke et. al., IEEE/ACM Trans. Netw. 2012
- For one of the first papers on loop-detection you can look at Robert Tarjan, Depth-first search and linear graph algorithms, 1972
- For more on the SWAN-project see Chi-Yao Hong et. al., SIGCOMM 2013