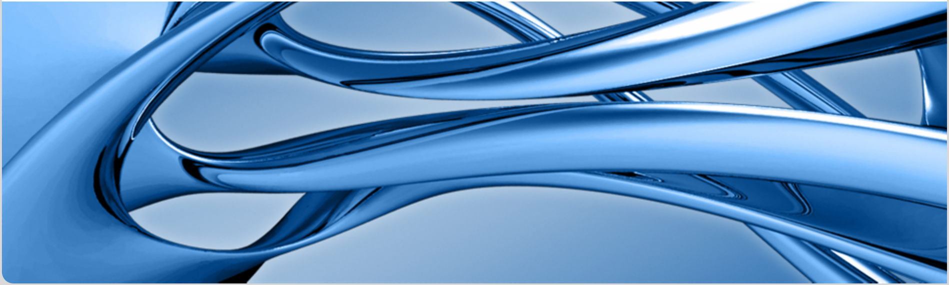
Towards a Resilient In-Band SDN Control Channel

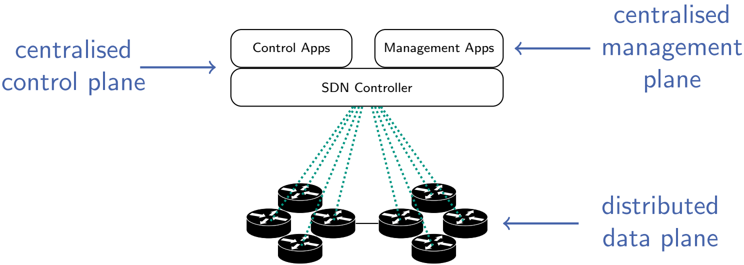
Polina Goltsman, Martina Zitterbart, Artur Hecker*, Roland Bless

***Huawei European Research Center**

Institute of Telematics



Background



centralised control plane

Control Apps

Management Apps

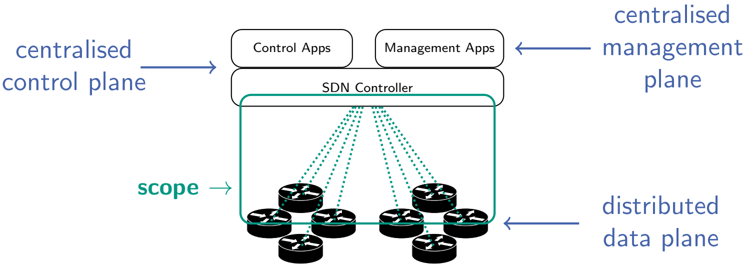
SDN Controller

centralised management plane

distributed data plane

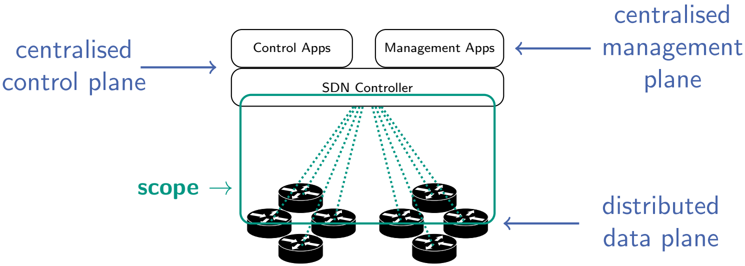
- in-band control

Background



- in-band control
- how do the planes communicate, that is, how to establish a connectivity of the *SDN control channel*

Background



centralised control plane

Control Apps

Management Apps

SDN Controller

centralised management plane

scope

distributed data plane

- in-band control
- how do the planes communicate, that is, how to establish a connectivity of the *SDN control channel*
    - *initial scope:* pure SDN network and single controller

Our Approach

switches are responsible for maintaining connectivity to the SDN controller

➜ SDN control channel is maintained by *a distributed protocol*

Why Distributed?

① control channel is *maintained separately* from normal paths

- ➜ protocol executed by the switches is separated from the data plane configuration by the controller

② optimized solely for *robustness/resilience* of connectivity under failures

- ➜ distributed protocols are more robust/resilient
- ➜ one optimization goal is simple enough to be solved by a distributed protocol

next

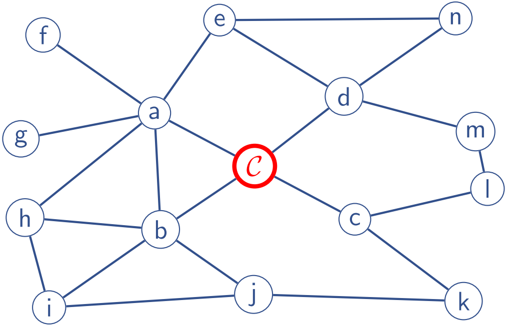
The routing protocol IS-C

IS-C – Intermediate System to Controller Protocol

Problem Statement

- maintain *bi-direction* communication paths between *every node* in the network and *the controller node*
- optimized solely for *robustness* of connectivity under failures

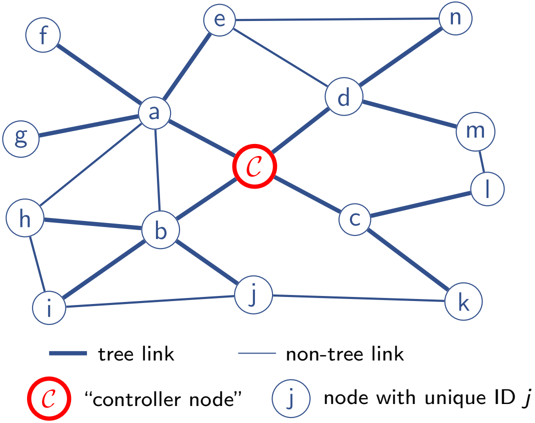
Basic Idea



$\mathcal{C}$ “controller node” j node with unique ID *j*

0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

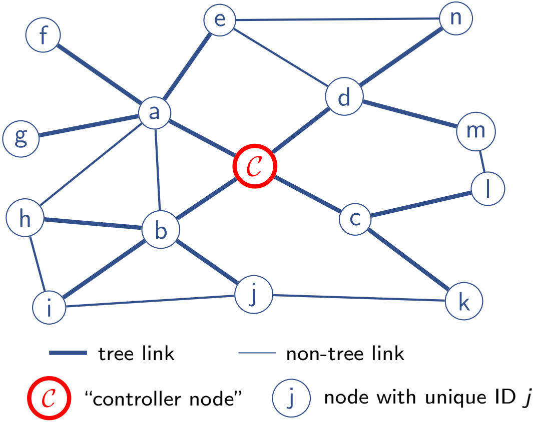
Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

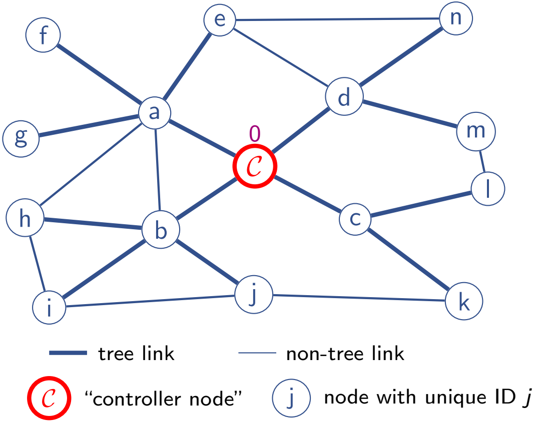
1. spanning tree with root at the controller

Basic Idea



0. initially
    - nodes are preconfigured with unique IDs
    - controller node knows its status
1. spanning tree with root at the controller
2. prefixed-based labeling scheme

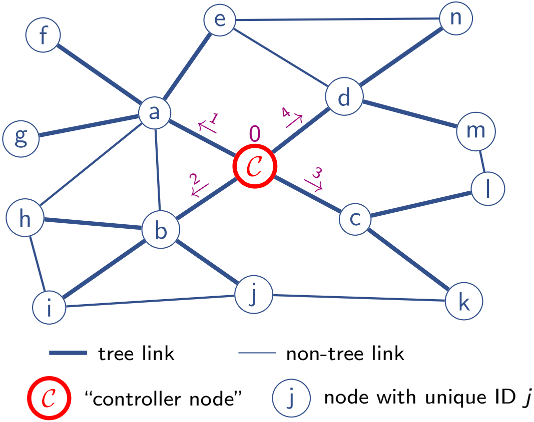
Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

1. spanning tree with root at the controller
2. prefixed-based labeling scheme

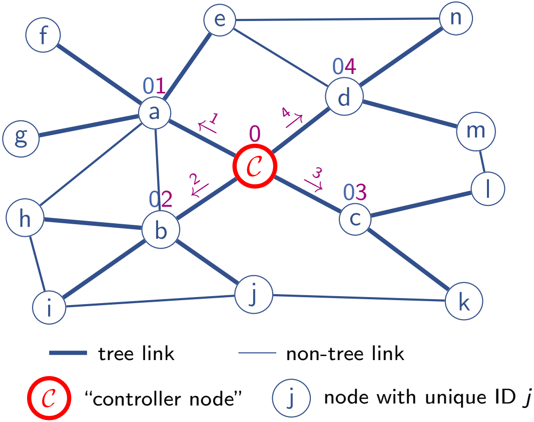
Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

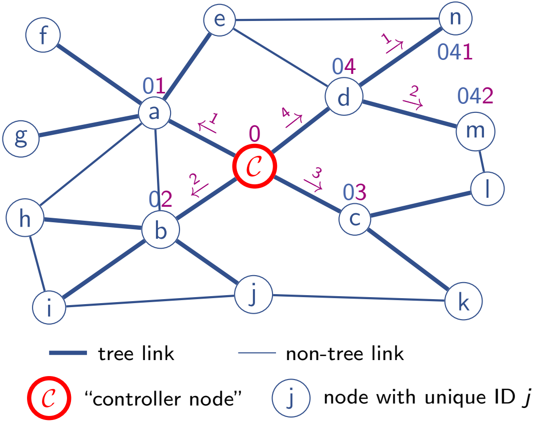
1. spanning tree with root at the controller
2. prefixed-based labeling scheme

Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status
1. spanning tree with root at the controller
2. prefixed-based labeling scheme

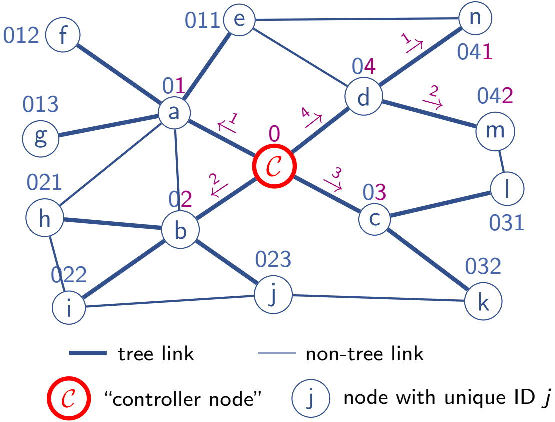
Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

1. spanning tree with root at the controller
2. prefixed-based labeling scheme

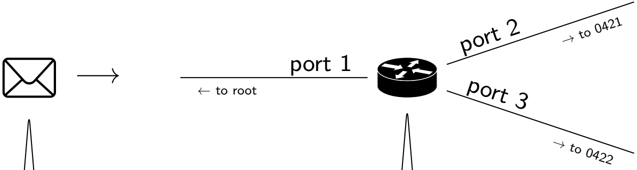
Basic Idea



0. initially
   - nodes are preconfigured with unique IDs
   - controller node knows its status

1. spanning tree with root at the controller
2. prefixed-based labeling scheme
3. robustness (*in 2 slides*)

Basic Idea – Data Plane Sketch



header

src label	0
dst label	**0422**345
dst ID	j

…

$id = a,\ label = 042$

routing table

dst label	output
0	port 1
0421*	port 2
0422*	port 3

- labels → IPv6 Addresses
- IDs → upper layer protocol

Basic Idea – Summary

1. spanning tree with root at the controller
   → necessary communication paths
   - breadth-first-search tree → shortest paths
2. prefixed-based labeling scheme
   - arbitrary unique IDs
     - no management overhead
   - temporary topology-dependant addresses
   → address aggregation
     → small tables
3. *Next:* robustness

Robustness of Connectivity under Failures

Requirements

❶ fast-failover scheme in the data plane

- ➜ connectivity while control plane converges

❷ convergence – triggered vs. periodic

- transient failures / link flapping
  - ➜ overhead
- stability issues

❸ consistent updates

- avoid inconsistent state in the dataplane
- (also known as loop-free convergence in IP)

Fast-Failover

- *Approach:* pre-computed backup paths
  - requirement: practically implementable
    - ➜ static forwarding tables and fixed “small” state in packets

- *Basic Idea:*
  - each node learns the topology of its *local* neighborhood
    - ∼layered breadth-first search untill paths are found
  - each node calculates *required* backup paths based on this topology
  - each node installs backup paths on its neighbors

- *Initial Goal:* single link or node failure
  - protection against link-to-parent and parent node failure

- *Ultimate Goal:* multiple-failure protection
  - number of backup paths is *local* decision of each node

Route Maintenance – Basic Idea

- spanning tree is updated in the background after failures
- tree is re-labeled periodically
- labels have sequence numbers so that two trees can coexist in the dataplane

Route Maintenance – Details I

Spanning Tree Computation

- distance vector-based spanning tree
- a solution for count-to-infinity

- control plane converges after each failure in the “background”
    - no dataplane updates

Route Maintenance – Details II

Relabeling / Route Updates

- periodic with fairly large period
  - ➜ link flapping

- algorithm overview
  - root node signals start of the relabeling
  - nodes assign labels based on current state of the control plane
  - nodes update dataplane
  - once every node has updated the dataplane, root issues another signal (see next slide)

- distributed algorithms, the root node acts as a *coordinator*

Route Maintenance – Details III

Updates Consistency

Basic Idea: *maintain "old" and "new" trees during relabeling*

- ensure that "old" and "new" trees can coexist in the dataplane
    - sequence numbers for labels
        - each label has a sequence number
        - changed on each label change
        - address = sequence number + label
- use "old" state while "new" state is being calculated
- coordinate switching to the "new" state
    - $\sim$ use new address

routing table

seq/dst label	output
1/0421*	port 2
2/0421*	port 3

. . .

Implementation Status

- *step 1:* simulation of an algorith on abstract graphs
    - graph with nodes and links
    - prototype currently in progress
    - next step: evaluation
- *step 2:* implementation and functional evaluation in a mininet testbed
    - demo
    - performance evaluation if possible

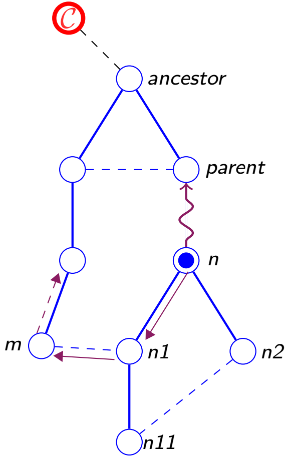
Conclusion

1. connectivity of the in-band control channel is maintained by a *distributed protocol*
    - robust/resilient
    - separated from the normal control plane
2. IS-C – the proposed protocol design
    - spanning-tree + labeling scheme
    - periodic relabeling and sequence numbers
        - ➜ consistent network updates
    - pre-computed backup paths scheme for trees
        - ➜ connectivity while control plane converges

Backup Slides

Fast-Failover-1 – Basic Scheme

node ⟶ root



● calculating node *n*

∿ protected link

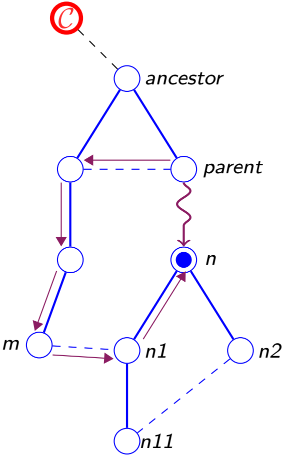
→ backup path, calculated by node *n*

⇢ resume normal forwarding

- *each node know a path to* $\mathcal{C}$
- *n needs to ensure that the backup path does not go through the protected link*
- *n finds m: lower common ancestor is “above” n’s parent in the tree*
  - *determined from node labels*

Fast-Failover-1 – Basic Scheme

root ⟶ node, link failure



● calculating node *n*

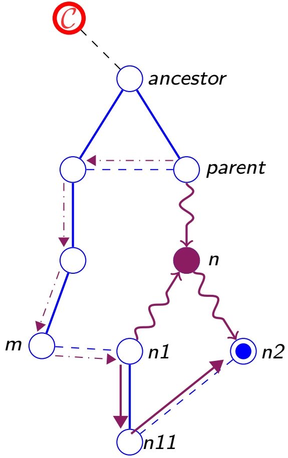
∿ protected link

→ backup path, calculated by node *n*

💡 *n finds best route to its parent*

Fast-Failover-1 – Basic Scheme

root ⟶ node, node failure



- ● calculating node *n*2
- ● protected node *n*
- ∿ equivalent protected links
- ⇢ backup path from *parent* to *n*
- → backup path, calculated by node *n*2

- *node failure ∼ failure of all links*
  - *parent assumes link failure*
  - *n*1 *will also see link failure*

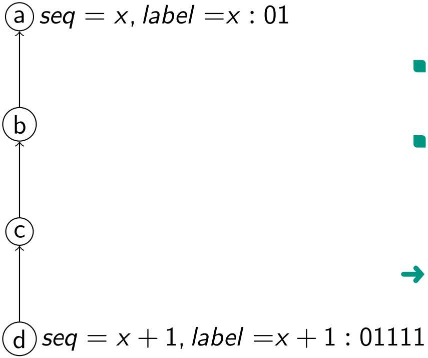
➜ *n*2 *calculates path from n*1

Realisation with OpenFlow DataPlane

“standard” packet format with different semantics
(original goal of OpenFlow?)

- IPv6
    - IPv6 unique local addresses:
        - `IPv6 prefix : sequence number : current label`
    - src/dst – node label/SDN controller label
        - ➜ flow entry matches src/dst pair
    - traffic class for flags
- additionally
    - rule timeouts
    - FAST-FAILOVER group tables
    - ? MPLS labels

Why global sequence numbers?


a $seq = x$, $label = x : 01$

b

c

d $seq = x + 1$, $label = x + 1 : 01111$

- address in packet/OF rules:
  - `sequence number : label`
- what rules to install in a,b, and c?
  - should be:
    $seq \geq x$ && $label = 01*$
  - encode in OpenFlow?

➜ global sequence number

Related Work

- Peregrine: in-band Ethernet SDN
    - bootstrap in STP mode; disjoined spanning trees
- Spark: SDNs for Carrier Networks
    - control channel isolation
    - controller managed control channel requires precomputed backup paths
    - how to establish control channel is not specified
- Medieval:
    - controller-centric approach, supports multiple controllers
    - on initiation and loss of connectivity – ARP messages
- ResilientFlow
    - distributed protocol to recover from large scale link failures
    - focuses on the integration of distributed and centralized control planes
- Tesseract: an implementation of 4D control plane
    - source routing on the “control channel”
    - focuses on security