Recursive SDN
A Framework for Large-Network Connectivity

UC Berkeley
Murphy McCauley
Aurojit Panda, Zhi Liu, Kazemkhani, Teemu Koponen (VMware), Barath Raghavan (ICSI), Scott Shenker
Recursive SDN
A Framework for Large-Network Connectivity

UC Berkeley
Murphy McCauley
Aurojit Panda, Zhi Liu, Kazemkhani, Teemu Koponen (VMware), Barath Raghavan (ICSI), Scott Shenker
Outline

• Motivation / Introduction
  – More detail on programming model

• Virtualization-based network repair
  – New since last year

• Examples of Unicast routing algorithms in RSDN
  – More detail on two particular designs

• Traffic Engineering
  – New since last year
Motivation

• Carrier networks are large...
  – Numerically (10s of thousands of nodes)
  – Geographically (thousands of miles)

• Challenge: how to make SDN scale?
  – Handle large numbers
  – Maintain geographic locality
Simple answer: structure the control plane as a recursive hierarchy
- Carrier networks have structure/locality
- Hierarchy solves a lot of scalability problems!
- Proud networking tradition

Easy to say. How does it actually shake out?
Goals

Goals

• Work on networks with structure/locality

• Provide connectivity
  – Unicast, multicast, anycast
  – Failure recovery
  – Traffic engineering

• Framework that provides scalability and fast network repair
  – Scalability follows from recursive structure
  – Repair comes “for free” with the framework

Non-Goals

• Tree-like or mesh-like networks
• Abstract optimal structure

• Implement complex and arbitrary policy (do that at the edge)

• Mandate specific forms of routing, traffic engineering
Recursive Approach

Controllers correspond to LXBs

Same (or similar) code on each controller
Recursive Approach
Recursive Framework

• Inside a “normal” LXB:

  App-Specific Parent Messaging (if not root)

  Repair
  Child Graph
  Application 1 (e.g. Unicast)

  ... Application n (e.g. Multicast)

  App-Specific Child Messaging
Recursive Framework

to parent

App-Specific Parent Messaging (if not root)

Repair
Child Graph
Application 1 (e.g. Unicast)
...
Application n (e.g. Multicast)

App-Specific Child Messaging

--

App-Specific Parent Messaging

Child Graph

App 1
...

App n

Leaf Logic
Virtualization-Based Repair
Switch Interface

Switches

--

App-Specific Parent Messaging

Child Graph

App 1
...

App n

Leaf Logic
Virtualization-Based Repair
Switch Interface

Switches
Outline

• Motivation / Introduction

• Virtualization-based network repair

• Examples of Unicast routing in RSDN

• Traffic Engineering
Network Repair

• Start with link protection
  – Compute failover paths
  – Recompute them when things change

• This is a simple form of “virtualization”
  – Logical network remains unchanged
  – Merely providing logical versions of failed links
Extending Link Protection

• Link protection cannot handle general failures
  – Must rely on route recomputation (global)

• Our goal: extend this idea of virtualization
  – Handle completely general failure cases
  – Reduce need to recompute routes quickly
Network Repair

• RSDN’s virtualization-based strategy
  – When links fail, create logical link
  – When a node is unreachable…
    • Create a virtual version of the node which is reachable
    • Create logical links to its neighbors

  – You’re (virtually) repairing the topology, so…
    • existing routing state just works!
## Network Repair

A repair algorithm for node a

1. Initialize `NeedPath`
   
   ```
   NeedPath(a) \leftarrow \text{Nbr}(a)
   ```

2. Repeat while `NeedPath(a) \neq \emptyset`
   
   ```
   \textbf{while } \text{NeedPath}(a) \neq \emptyset \textbf{ do}
   
   b \leftarrow \text{member}(\text{NeedPath}(a)) \quad \triangleright \text{Take element } b
   
   \textbf{if } \text{ComputePath}(a,b) = \text{true} \textbf{ then}
   
   \quad \triangleright \text{If we can find a path from } a \text{ to } b
   
   \quad \text{NeedPath}(a) \leftarrow \text{NeedPath}(a) \setminus b
   
   \quad \text{HavePath}(a) \leftarrow \text{HavePath}(a) \cup b
   
   \textbf{else}
   
   \quad \triangleright \text{If no path is found}
   
   \quad \text{IncludeTable}(a,b)
   
   \quad \text{NewNbrs} \leftarrow \text{Nbr}(b) \setminus \text{HavePath}(a)
   
   \quad \text{NeedPath}(a) \leftarrow \text{NeedPath}(a) \cup \text{NewNbrs}
   
   \quad \text{NeedPath}(a) \leftarrow \text{NeedPath}(a) \setminus b
   
   \textbf{end if}
   
   \textbf{end while}
   ```

### Table: Network Repair Algorithm

<table>
<thead>
<tr>
<th>Nbr(a)</th>
<th>Neighbors of node a</th>
</tr>
</thead>
<tbody>
<tr>
<td>HavePath(a)</td>
<td>Nodes to which a already has a computed path.</td>
</tr>
<tr>
<td>NeedPath(a)</td>
<td>Nodes to which a needs a path.</td>
</tr>
<tr>
<td>ComputePath(a,b)</td>
<td>Computes a path between a and b, considering the current state of the network. If no such path exists, ComputePath(a,b) fails.</td>
</tr>
<tr>
<td>IncludeTable(a,b)</td>
<td>Include all the elements of b’s forwarding table into node a.</td>
</tr>
</tbody>
</table>
Network Repair

• If a neighbor isn’t unreachable…
  – Can you compute a tunnel to reach it?
    • Yes:
      – Do it.
    • No:
      – Virtualize that neighbor (import its tables)
      – Check reachability of its neighbors (recurse)
Network Repair

If a neighbor isn’t unreachable...
  – Can you compute a tunnel to reach it?
    • Yes:
      – Do it.
    • No:
      – Virtualize that neighbor (import its tables)
      – Check reachability of its neighbors (recurse)
Network Repair

- If a neighbor isn’t unreachable…
  - Can you compute a tunnel to reach it?
    - Yes:
      - Do it.
    - No:
      - Virtualize that neighbor (import its tables)
      - Check reachability of its neighbors (recurse)
Network Repair

• Updates are simple (computationally)
  – Copying tables
  – Computing short paths

• Updates are local (only near the failure)
  – Usually only requires a single controller

• By all means, fix your routes too
  – .. but it’s no longer time-critical
Network Repair

• How well does it work?

• Example: Assume average repair time of 50ms
  – 10ms to controller (conservative on average)
  – 10ms computation time (very conservative on average)
  – 10ms from controller (conservative on average)
  – 20ms FIB update (technology specific)

• Compare four measures of connectivity:
  – **Physical**: How many src/dst pairs *could* be connected
  – **No repair**: How many existing src/dst *routes* still work
  – **Link repair**: How many existing routes work with link protection
  – **Virtual repair**: How many existing routes work with virtualization
Network Repair

• Links go down once per day on average
• Nodes go down every two days on average

On average, how many physically connected pairs have broken routes?

<table>
<thead>
<tr>
<th>Protection</th>
<th>Broken Routes</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>12%</td>
</tr>
<tr>
<td>Link</td>
<td>2.7%</td>
</tr>
<tr>
<td>Virtualization</td>
<td>0.0002%</td>
</tr>
</tbody>
</table>

Note: This isn’t magic!

![CDF Graph](image)
Within the Framework

• We’ve been talking *about* the framework

• What’s implemented *using* the framework?
  – Unicast, multicast, anycast
  – Traffic Engineering

  – Utilize recursive structure (scaling, locality)
  – But choose different algorithms…
Outline

• Motivation / Introduction

• Virtualization-based network repair

• Examples of Unicast routing in RSDN

• Traffic Engineering
Unicast Routing

• Many tradeoffs and variations
  – Computation time
  – State size
  – Path properties (stretch, etc.)

• Two specific solutions I’ll be discussing:
  – Fine Granularity Routing (FGR)
  – Variable Granularity Routing (VGR)
Fine Granularity Routing

- LXBs compute shortest paths between edges
- Report distances to parent LXB as matrix
- Recurses upwards
  - Parent computes paths between its edges
- Leads to globally shortest paths
Fine Granularity Routing

- Total time spent computing is greater than a “flat” all-pairs shortest paths algorithm
  
  - But computation within a tier done in parallel

  - Figure of merit is the “tower time” – sum of completion times for each tier

  - FGR ends up being faster
FGR (Re)-Computation

• Full computation 2.7x faster than Dijkstra on 10k nodes
  – Increasingly faster on larger graphs
  – But full computation usually isn’t required!
  – .. and it’s only the root which is a real killer!

• Computation breakdown by tier:

<table>
<thead>
<tr>
<th>Tier</th>
<th>Amount of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (Root)</td>
<td>97.3%</td>
</tr>
<tr>
<td>1</td>
<td>0.3%</td>
</tr>
<tr>
<td>2</td>
<td>2.2%</td>
</tr>
<tr>
<td>3</td>
<td>0.2%</td>
</tr>
</tbody>
</table>
Variable Granularity Routing

- FGR scales with number of destinations
- Idea: Reduce granularity for “far away” things

- Get shortest path to nearest point of sibling LXB containing a destination
  - Recurse one level down the hierarchy until reaching destination
  - (Greedy shortest path approximation)
Variable Granularity Routing
Variable Granularity Routing

Same process takes place in lower tiers
Variable Granularity Routing

Same process takes place in lower tiers
Variable Granularity Routing

• Solves problem of “top heavy” FGR
  – Root time is only 3% of FGR

• Similarly dramatic improvements in switch state

• Incurs topology-dependent stretch
  – In the same 10k node topology…
    • 90% of paths have stretch less than 13% by hop
    • 95% of paths have stretch less than 10% geographically
Wrapping up Unicast

- **FGR**
  - Faster than “flat” all-pairs
  - Shortest paths

- **VGR**
  - Lower computation time and switch state
  - Incurs some stretch

- Only two of many possibilities
Outline

• Motivation / Introduction

• Virtualization-based network repair

• Examples of Unicast routing in RSDN

• Traffic Engineering
Traffic Engineering

• Current area of RSDN work
• Currently pursuing two different strategies
  – Recursive Adaptive Multipath
    • Inspired by MATE, DATE, TeXCP, etc.
  – Recursive Linear Programming approach
    • LP TE approach adapted for recursive structure
Recursive LP Traffic Engineering

• In short: do LP-based optimization at each tier
• Spreads out the work

• *Early* simulation results:
  – Achieves very close to optimal balance
  – But scales *much* better than global solver
    (100 *times* faster on relatively small topology!)
RSDN in Brief

• Recursively aggregate network
• Structure controllers to mirror aggregation
• Provides framework for connectivity-oriented applications (unicast, multicast, TE)
  – Allows variety of designs / tradeoffs
• Virtualization-based repair recovers quickly regardless of routing algorithm
• Recursive TE almost optimal (early results)
The End
**Table Size**

- Average switch table sizes on 10k node topo

<table>
<thead>
<tr>
<th>Routing</th>
<th>Label-Based</th>
<th>Consec-LPM</th>
<th>Random-LPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>FGR</td>
<td>2,943</td>
<td>1,296</td>
<td>41,206</td>
</tr>
<tr>
<td>VGR</td>
<td>212</td>
<td>382</td>
<td>13,088</td>
</tr>
</tbody>
</table>