Central Control over Distributed Routing

fibbing.net

Stefano Vissicchio
UCLouvain

SIGCOMM’15

Joint work with
O. Tilmans (UCLouvain), L. Vanbever (ETH Zurich) and J. Rexford (Princeton)
Fibbing, a hybrid SDN architecture

Fibbing
central control over link-state IGPs
Fibbing *combines* advantages of SDN and traditional networking.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Traditional</th>
<th>Fibbing</th>
<th>SDN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manageability</td>
<td>low</td>
<td>high</td>
<td>high</td>
</tr>
<tr>
<td>Flexibility</td>
<td>low</td>
<td>high</td>
<td>highest</td>
</tr>
<tr>
<td>Scalability</td>
<td>by design</td>
<td>by design</td>
<td>ad hoc</td>
</tr>
<tr>
<td>Robustness</td>
<td>high</td>
<td>high</td>
<td>low</td>
</tr>
</tbody>
</table>
Central Control over Distributed Routing

fibbing.net

1. Manageability
2. Flexibility
3. Scalability
4. Robustness
SDN achieves high manageability by centralizing both computation and installation.

- computes paths
- derived FIB entries
- install FIB entries
Fibbing is as manageable as SDN, but centralizes only high-level decisions.
Fibbing keeps installation distributed, relying on distributed protocols.
Distributed installation is controlled by injecting carefully-computed information.
We study which messages to inject for controlling intra-domain routing protocols.
The output of the controlled protocol is specified by operators’ requirements.
To control IGP output, the Fibbing controller inverts the shortest-path function.
Central Control over Distributed Routing

fibbing.net

1. Manageability
2. Flexibility
3. Scalability
4. Robustness
Consider this simple network
(implemented with Cisco routers)
An IGP control-plane computes shortest paths on a shared weighted topology.
IGP shortest paths are translated into forwarding paths on the data-plane.
In Fibbing, operators can ask the controller to modify forwarding paths.
The Fibbing controller injects *fake nodes and links* to the IGP control-plane.

requirement \((C,A,B,X,D2)\)

node V1, link \((C,V1)\), map \((C,V1)\) to \((C,A)\)
Informations are flooded to all IGP routers in the network

requirement \((C,A,B,X,D2)\)

node V1, link \((C,V1)\), map \((C,V1)\) to \((C,A)\)
Fibbing messages *augment* the topology seen by all IGP routers

requirement \((C,A,B,X,D2)\)
Augmented topologies translate into new control-plane paths.

Requirement \((C,A,B,X,D2)\)

Node V1, link \((C,V1)\), map \((C,V1)\) to \((C,A)\)
Augmented topologies translate into new *data-plane* paths.

requirement `(C,A,B,X,D2)`

node V1, link (C,V1), map (C,V1) to (C,A)
By achieving full per-destination control, Fibbing is highly flexible

Any set of forwarding DAGs can be enforced by Fibbing

- fine-grained traffic steering (middleboxing)
- per-destination load balancing (traffic engineering)
- backup paths provisioning (failure recovery)
Central Control over Distributed Routing

fibbing.net

1. Manageability
2. Flexibility
3. Scalability
4. Robustness
We implemented a Fibbing controller
We also propose algorithms to compute augmented topologies of limited size.
For our Fibbing controller, we propose algorithms to be run in sequence.
Consider the following example, with a drastic forwarding path change.

original shortest-path
“down and to the right”

desired shortest-path
“up and to the right”
Simple adds one fake node for every router that has to change next-hop.
Merger iteratively merges fake nodes (starting from Simple’s output)
Merger iteratively merges fake nodes (starting from Simple’s output)
This way, Merger programs multiple next-hop changes with a single fake node.
Simple and Merger achieve different trade-offs in terms of time and optimization efficiency

We ran simulations on Rocketfuel topologies
Merger is relatively slower, but still, sub-second

![Graph showing computation time vs % of nodes changing next-hop. The graph compares 'merger (median)' and 'simple (median)'.
Merger reduces the size of the topology by 25% on average (50% in the best case)
We implemented the machinery to listen to OSPF and augment the topology.
We implemented a fully-fledged Fibbing prototype and tested it against real routers.

2 measurements

How many lies can a router sustain?

How long does it take to process a lie?
Experiments on real routers show that Fibbing has very limited impact on routers

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>router memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>0.7</td>
</tr>
<tr>
<td>5 000</td>
<td>6.8</td>
</tr>
<tr>
<td>10 000</td>
<td>14.5</td>
</tr>
<tr>
<td>50 000</td>
<td>76.0</td>
</tr>
<tr>
<td>100 000</td>
<td>153</td>
</tr>
</tbody>
</table>

DRAM is cheap

>> # real routers
Experiments on real routers show that Fibbing has very limited impact on routers.

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>router memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0.7</td>
</tr>
<tr>
<td>5000</td>
<td>6.8</td>
</tr>
<tr>
<td>10000</td>
<td>14.5</td>
</tr>
<tr>
<td>50000</td>
<td>76.0</td>
</tr>
<tr>
<td>100000</td>
<td>153</td>
</tr>
</tbody>
</table>

DRAM is cheap

CPU utilization always under 4%
Experiments on real routers show that Fibbing does not impact IGP convergence

Upon link failure, we registered *no difference* in the (sub-second) IGP convergence with

- no fake nodes
- up to 100,000 fake nodes and destinations
Experiments on real routers show that Fibbing achieves fast forwarding changes.

<table>
<thead>
<tr>
<th># fake nodes</th>
<th>installation time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 000</td>
<td>0.9</td>
</tr>
<tr>
<td>5 000</td>
<td>4.5</td>
</tr>
<tr>
<td>10 000</td>
<td>8.9</td>
</tr>
<tr>
<td>50 000</td>
<td>44.7</td>
</tr>
<tr>
<td>100 000</td>
<td>89.50</td>
</tr>
</tbody>
</table>

894.50 μs/entry
Central Control over Distributed Routing

fibbing.net

1. Manageability
2. Flexibility
3. Scalability
4. Robustness
Fibbing improves robustness by relying on the underlying IGP

- no controller action needed in some cases
  IGPs re-converge quickly

- IGP provides fast failure detection and control-plane sync
  thanks to its shared topology

- Fibbing supports fail-open and fail-close semantics
We ran a failure recovery case study, with distributed Fibbing controller

( fail-close(D1) );
( fail-open(D2) );
Fibbing survives replica failures with no impact on forwarding

( fail-close(D1) );
( fail-open(D2) );
Fibbing reacts to network failures quickly re-optimizing forwarding

\[(\text{fail-close(D1)})\];
\[(\text{fail-open(D2)})\];
Fibbing reacts to partitions, respecting fail-close and fail-open semantics

( fail-close(D1) );
( fail-open(D2) );
Fibbing recovers correctly
(as soon as failures are fixed)

\[
\begin{align*}
&\text{fail-open}(D2); \\
&\text{fail-close}(D1);
\end{align*}
\]
Fibbing shows the *benefits* of central control over distributed protocols

- Realizes SDN management model
  network-wide automated control

- Simplify controllers and improves robustness
  heavy work is still done by routers

- Works today, on existing networks
  avoids SDN deployment hurdles
Central Control over Distributed Routing

fibbing.net

Tell me lies, tell me sweet little lies
— Fleetwood Mac

Stefano Vissicchio
stefano.vissicchio@uclouvain.be
More on Compilation

• ( (E,C,D4) and (E,G,D4);
  ((A,*,B,*,D2) or (A,*,C,*,D2));
  (F,G,*,D3) as backup of ((F,H))); )

• While generating DAGs, check whether the graph is loop free