HybridTE
Traffic Engineering for Very Low-Cost Software-Defined Data-Center Networks

Philip Wette, Holger Karl
Universität Paderborn, Germany
Control treatment of data flows in a network to improve desirable performance metrics

**Treatment options:**
- Routing
- Priorities
- Forwarding disciplines
- Coding
- ...

**Metrics:**
- Average throughput
- Percentiles of throughput
- Latency
- Reliability
- ...

Traffic engineering
Why do we need yet another TE technique?

Related work:
- ECMP
- Hedera, NSDI 2010
- DevoFlow, SIGCOMM 2011
- MicroTE, CoNEXT 2011
- CONGA, SIGCOMM 2014
- ...

Existing TE techniques either require non-standard features or have too high requirements on the switches.

Need for a TE technique for data centers with low requirements
- Limited flow table sizes
- Limited processing capabilities
- Limited memory speed
Design a routing algorithm / TE technique for data centers built from low-cost off-the-shelf OpenFlow switches
Data Center Traffic – Properties

- Topology features a lot of alternative paths
- Software running in data centers is typically distributed (Big Data, multi-tier)

- Average flow size is 146 KB
- 80% of all flows smaller than 10 KB
- Most bytes are transported by a few elephant flows

Concentrate effort

Two elephants on the same link?

Typical idea: concentrate traffic engineering effort on elephant flows

- Old idea, e.g., Hedera, …
- Reality check: small flow tables, small update rates, slow memory
- Simple scheme!
HybridTE

Idea of HybridTE:

"If we can't route all flows individually, let's only try the elephants"

- Initially handle all flows using proactively installed default routes
- Use information about elephant flows to handle elephants individually

What is the expected quality of such a simple scheme?
Hybrid TE: Main ingredients

- Default routes
- Routes for elephants
- Recognizing elephants
**HybridTE Default Routes – Destination Routing**

- **Goal**: Initially spread the traffic among all available paths
- **Use one forwarding tree per destination rack**
  - Each tree is constructed at random along shortest paths
  - Heavily based on wildcards
  - Already gives some degree of load-balancing

- Requires R OpenFlow flow entries per switch (R is # of racks)
HybridTE Elephant Rerouting

- SDN controller constantly monitors link utilization

- Apart from the existence and its route, we have initially no information about the elephant
  - Hiding behind the wildcard rules → no byte count, data rate, etc.
  - Makes it hard to make proper routing decisions

- Elephants are initially rerouted to the least loaded shortest path
  - After rerouting, we can collect information about the elephant
Periodic Rerouting

- HybridTE reroutes all elephants every 5 seconds
  - 3 Phases
- 1. Compute data rates of background traffic
  - Hypothetically remove elephants
- 2. Compute the **natural demand**
  - Maximum possible data rate if the network were non-blocking (limited at host NICs only)

![Diagram showing network traffic and rerouting]

<table>
<thead>
<tr>
<th>Rate</th>
<th>Demand</th>
</tr>
</thead>
<tbody>
<tr>
<td>500 Mbps</td>
<td>1.4 Gbps</td>
</tr>
<tr>
<td>2 Gbps</td>
<td></td>
</tr>
<tr>
<td>3 Gbps</td>
<td>5 Gbps</td>
</tr>
<tr>
<td>5 Gbps</td>
<td></td>
</tr>
</tbody>
</table>

- 3. Find optimal routing (NP-complete problem)
  - Reroute elephants (one after another in randomized order) on the least loaded shortest path
  - Update expected resulting data rate
How to recognize elephants?

Related work:
- Participatory Applications
- Logfile analysis: HadoopWatch, INFOCOM 2014
- Hardware devices like ElephantTrap, Stanford 2007
- In-network traffic analysis (e.g., by Virtual Network Functions)

Here: Randomized packet sampling!
- Switches send every packet with probability $p$ to the SDN controller
- Controller considers any flow as elephant if it received at least $n$ packets
- Recall: Cheap and simple!
Evaluation – Plot line

- Is HybridTE viable with limited information available in practice?
  - Start with a crystal ball for elephants, derive operational regions, try to achieve them in practice

- Formally: Evaluated HybridTE using different numbers
  - of false positives: Some mice erroneously treated as elephants
  - of false negatives: Some elephants erroneously treated as mice
  - of reporting delays: how long does it take to detect an elephant?

- Compare against ECMP and Hedera
**Evaluation scenario**

- **Emulation of a data center of 1440 hosts**
  - 20 hosts per rack → 72 racks in total, 2 core switches, 8 racks per pod / 2 pod switches, 10G full-duplex links
  - Emulated via Maxinet on 9 machines, time dilation 150

![MaxiNet diagram](image)

- **Created 10 traffic traces (60s each) using DCT²Gen**
  - Each 60 simulated seconds
  - Highly realistic data-center traffic
    - Traffic on layer 2 has the same statistical properties as reported by Benson et al.
Evaluation scenario

- Different load levels were created by artificially limiting link speeds
  - **Load level** $x \rightarrow 10/x$ G links
    - Load level 1 $\rightarrow 10$ G links
    - Load level 2 $\rightarrow 5$ G links

- **Elephants**: Flows larger than 10 Mbyte

- **Metric**: Average flow completion time
Results – Load level 1, false negatives

- ECMP produces 7% higher flow completion times than HybridTE100
- Hedera does not outperform ECMP (consistent with original paper)
- HybridTE works better with more information
Results – Load level 1.5, false negatives

- Gap between HybridTE and ECMP increases with increasing load
  - On average, ECMP yields 20% higher flow completion times
  - Gap increases for higher load levels
Results – Load level 1.75, false negatives

- ECMP yields 28.7% larger flow completion times than HybridTE
Impact of False Positives on the Flow Completion Time

- Experiment with HybridTE100 on flow trace 7 with load level 1, increasing false positives rate

HybridTE is resilient against high numbers of false positives
Impact of Reporting Delays, false negatives

Reporting delay:
Time between the start of an elephant and it's report to HybridTE

Decrease (in percent) of the flow completion time relative to ECMP results on flow trace 7 at load level 1
Elephant Detection Through Simple Packet Sampling

- Replace the crystal ball!
  - p: Probability to send packets;
  - n: number of packets to see before classifying a flow as elephant
## Conclusion

- HybridTE outperforms ECMP and Hedera in a scenario with realistic traffic
  - ECMP yields up to 28.7% larger flow completion times than HybridTE

- HybridTE has very modest requirements on the switching hardware
  - Very few wildcard entries are used
  - Never more than 400 elephants co-existed
    - Very low rate of flow_mods (in case not too many false positives are reported)

- HybridTE is resilient against false positives
  - External elephant detectors can be used which create uncertain information about elephants
Thank you for your attention.

Holger Karl
Strawman: Per-flow routing?

- Average flow size on Layer 2: 146 KB
  → ~20,000 new flows/sec to saturate a full-duplex 10G link
  → ~500,000 new flows/sec to fully utilize a 48-port switch

- NoviSwitch 2128 (available in Q2 2015) marketing figures:
  - Up to 1M flow entries
  - Up to 12K flow installations per sec

- Per-flow routing in data centers with OpenFlow not possible today
  - At least not with off-the-shelf OpenFlow switches
DCT$^2$Gen – Traffic Generator for Data Centers

- Takes statistical properties of Layer 2 traffic as input
- Outputs a schedule of TCP connections
  - When the schedule is executed in a data center, this results in Layer 2 traffic as given before
DCT$^2$Gen

**Traffic Properties (on Layer 2):**

- Distribution of
  - Flow sizes
  - Inter flow arrival times
  - # intra-rack communication partners per server
  - # inter-rack communication partners per server
  - Sizes of intra-rack traffic matrix entries
  - Sizes of inter-rack traffic matrix entries

MaxiNet is a framework for distributing Mininet emulations onto a cluster of servers (called workers)

MaxiNet partitions the virtual topology in N parts.

From each partition a new topology is built and emulated using Mininet at a dedicated worker.

Traffic across partitions is routed over the physical network.

Goal: minimize edge cut.
Communication over Partition Boundaries

Mininet @ Worker 1

Mininet @ Worker 2

Mininet @ Worker 3

s1

s2

s3

h1

h2

veth-pair

veth-pair
veth-pairs are only internal to the Linux kernel
Communication over Partition Boundaries

- veth-pairs are only internal to the Linux kernel

Diagram:
- Mininet @ Worker 1
  - s1
  - Mininet @ Worker 2
    - s2
    - h1
  - Mininet @ Worker 3
    - s3
    - h2
Communication over Partition Boundaries

- veth-pairs are only internal to the Linux kernel
Controlling an Experiment

Mininet @ Worker 1

Mininet @ Worker 2

Mininet @ Worker 3

s1

s2

h1

s3

h2

GRE Tunnel

GRE Tunnel
Controlling an Experiment

Specialized entity (Frontend Node) to coordinate experiments
Controlling an Experiment

- Specialized entity (Frontend Node) to coordinate experiments

Diagram shows:
- Mininet @ Worker 1
- Mininet @ Worker 2
- Mininet @ Worker 3
- Frontend Node
- Connections via GRE Tunnel and RPC
Specifying Experiments with MaxiNet

- MaxiNet has its own Python API
- This API is very close to the Mininet API
  - Basically Mininet API augmented with some MaxiNet specific commands
- Process of distribution and cluster management is (mostly) transparent to the user

→ Migrating code from Mininet to MaxiNet is very simple

- MaxiNet also supports dynamic changes to the topology at runtime
  - Link up/down
  - Change of link characteristics (packet loss, delay, etc.)
  - Dynamically adding / removing hosts and switches
Complete Python code to start a MaxiNet emulation:

```python
1 import maxinet
2 import TreeTopo from mininet.topolib
3
4 cluster = maxinet.MininetCluster("pc1","pc2","pc3")
5 cluster.start()
6
7 exp = maxinet.Experiment(cluster, TreeTopo(3,2))
8 exp.addController("192.168.0.1", "6633")
9 exp.setup()
10
11 print exp.get("h2").cmd("ping -c5 10.0.0.3")
12 print exp.get("h5").cmd("ping -c3 10.0.0.8")
13
14 exp.stop()
15 cluster.stop()
```
Speedups using MaxiNet

- **Fat Tree**: linear speedup, but no gain from adding a fourth worker.
- All traffic is going through the root of the tree → bottleneck
- More than linear speedup for Switch Pairs

![Graph showing speedup comparison between Fat Tree and Switch Pairs](image)