# Use of Devolved Controllers in Data Center Networks

Adrian S.-W. Tam Kang Xi H. Jonathan Chao

Department of Electrical and Computer Engineering
Polytechnic Institute of New York University
Email: adrian@antioch.poly.edu, kxi@poly.edu, chao@poly.edu

***Abstract*—In a data center network, for example, it is quite often to use controllers to manage resources in a centralized manner. Centralized control, however, imposes a scalability problem. In this paper, we investigate the use of multiple independent controllers instead of a single omniscient controller to manage resources. Each controller looks after a portion of the network only, but they together cover the whole network. This therefore solves the scalability problem. We use flow allocation as an example to see how this approach can manage the bandwidth use in a distributed manner. The focus is on how to assign components of a network to the controllers so that (1) each controller only need to look after a small part of the network but (2) there is at least one controller that can answer any request. We outline a way to configure the controllers to fulfill these requirements as a proof that the use of devolved controllers is possible. We also discuss several issues related to such implementation.**

## I. Introduction

Among recent years' literature on data center networking, using a centralized controller for coordination or resource management is a common practice [1], [2], [3], [4], [5], [6], [7]. In [5], for example, a master server is used to hold the metadata for a distributed file system. In another example, [1], a flow scheduling server is responsible for computing a new route for a rerouted flow at real time. In [3], a controller is also used to enforce a route for a packet so that its use of the network compliances with the policies. Using a centralized controller not only makes the design simpler, but also sufficient. In [5], the authors claim that a single controller is enough to drive a fairly large network and the problem of single point of failure can be mitigated by replication.

Nevertheless, the use of a centralized controller subjects to scalability constraints. Usually the scalability problems are solved by load balancing. For example, replicating the whole database to multiple servers is a common way to load balance MySQL servers [8]. However, if the scalability problem is caused by too much data stored in a controller so that its response time is degraded, balancing load by identical controllers cannot solve the problem. As the data center network grows larger and larger, we can expect to have such problem in the near future. Therefore, it is interesting to study an alternative solution to a *single* centralized controller.

We study the use of *devolved* controllers in this paper. They together function as a single logical centralized controller but none of them have the complete information of the whole data center network. This is beneficial, for example, when the controllers are supposed to provide real time computations and too much data would cause the computation slow.

We take the following flow route assignment as an example to see how we can use devolved controllers: Whenever a flow, identified by a source and a destination node in the network, is to be established, the sending node will query the controllers for the route it should use to avoid congestion. The controllers are therefore responsible to monitor the network to assist the route selection. If the network topology were too large, the response time would be too long to be useful. Thus instead of a single omniscient controller to cover the whole network, we use multiple 'smaller' controllers so that each of them covers a partial topology only. When a controller is asked for a route, it responds with the topology data it has.

Note that this paper is not about route optimality or routing protocols, but to show that an omniscient controller is not the only solution. The novelty of this paper is on the concept of *devolved controllers*, which eliminates the scalability problem of traditional omniscient controller.

Our work is on the control aspect of data center networks. In recent years, there are many literatures that focus on control plane design in networks. Examples are OpenFlow [9], NOX [10] and Ethane [3]. To address the scalability issue of the controllers in these designs, their developers proposed [11] to partition the controllers horizontally (i.e. replication of controllers) and vertically (i.e. each controller serve a part of the network). While horizontal partitioning is trivial, this paper explore into the ways of vertical partitioning.

In the rest of this paper, we describe an example on controller use in section II and provide heuristic algorithms in section III on how to configure the controllers. Evaluation is provided in section IV and discussion on the use of devolved controllers in section V.

## II. Problem statement

On a network represented by a connected graph $G = (V, E)$, a flow is identified by the ordered pair $(s, t)$ where $s, t \in V$. On such a network, there are $q$ controllers. Each of them is managing a portion of the network, represented by a subgraph of $G$. We say a controller that manages $G' = (V', E')$ *covers* a node $v \in V$ or a link $e \in E$ if $v \in V'$ or $e \in E'$, respectively. Upon a flow is going to be established, the source node $s$

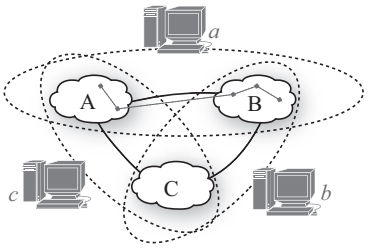


Fig. 1. A network managed by three controllers

queries the controllers for a route to destination $t$. Among the $q$ controllers, at least one of them responds with a path $p$ that connects $s$ to $t$, which is the route for this flow. TABLE I summarizes the terms used in this paper.

The controllers are supposed to respond to the flow route query in a very short time. Therefore, computationally intensive path-finding algorithms are not viable. Furthermore, we have to ensure at least one the $q$ controllers can provide a route for any source-destination pair $(s, t)$. One way is to have all routes pre-computed. Assume for any ordered pair $(s, t)$, we compute $k$ different paths $p_1, \ldots, p_k$ that join $s$ to $t$. We call the set $M = \{p_1, \ldots, p_k\}$ as a $k$-multipath. Then, we install the multipath into a controller. Upon the query is issued, the controllers will return the least congested one of the $k$ paths.

Fig. 1 gives an example of a network with three controllers. The part of the network that a controller covers is illustrated by a dotted ellipse. Precisely, there is a controller that covers all the routes between nodes in regions A and B as well as the routes within those regions; another controller covers that of regions A and C and yet another is for regions B and C. In this way, none of the controllers monitor every spot in the network but they together can respond to any request from any node. For instance, the route illustrated in Fig. 1 is entirely within the jurisdiction of controller $a$. So $a$ is the one to provide this route upon request. When the network grows, we can install more controllers to cover the network so that none of the controllers need to manage a region of too large in size.

Assume we have all the paths pre-computed, the immediate questions are then

1) How to optimally allocate the multipaths into the $q$ controllers? We define the optimality as the smallest number of unique links to be monitored by the controller. In other words, we prefer the controller to cover as small portion of the network as possible.
2) If no controller can monitor more than $N$ links on the network, what is the number of controllers needed?

## III. Approximate solution

We first consider the problem of optimal allocation of the multipaths into $q$ controllers.

For $n = |V|$ nodes on the network, there are $n(n-1)$ different source-destination pairs. This is also the number of multipaths to be pre-computed as mentioned in section II. It is a NP-hard problem to find the optimal allocation to

| | |
|---|---|
| $(s, t)$ | Source-destination pair |
| flow | A data stream from a source node to destination node |
| path/route | A path that connects two nodes |
| multipath | A set of paths that each of them connects the same pair |

TABLE I
Definition of terms used in this paper

$q$ controllers[1]. The size of the solution space for allocating $n(n-1)$ multipaths to $q$ controllers is given by the Stirling number of the second kind [13]:

$$\frac{1}{q!}\sum_{j=0}^{q}(-1)^j\binom{q}{j}(q-j)^{n(n-1)}$$

It becomes intractable quickly for a moderately large network.

### A. Path-partition approach

Instead of looking for a global optimal solution, we developed a heuristic algorithm to obtain an approximate solution. The algorithm is in two parts: Firstly it enumerates all the $k$-multipaths for all source-destination pairs $(s, t)$. Then, it allocates each multipath into one of the controllers according to a cost function. The multipaths are pre-computed with no knowledge of where they are to be allocated to the controllers. We call this the *path-partition approach*:

**Algorithm 1:** Path-partition heuristic algorithm

```
   Data: Network G = (V, E), q = number of controllers
1  foreach s, t ∈ V in random order do
       /* Constructing a k-multipath from s to t */
2      M := k paths joining s to t;
       /* Allocate into a controller */
3      for i := 1 to q do
4          c_i := cost of adding multipath M to controller i
5      end
6      Allocate multipath M to the controller j = arg min c_j
7  end
```

We implemented the multipath enumeration in algorithm 1 (line 2) after [14]. We find a path from $s$ to $t$ using Dijkstra's algorithm with unit link weight for each link in $E$. Then, the links used in this path have their link weights increased by an amount $\omega$. The Dijkstra's algorithm and link weight modification are repeated until all $k$ paths are found. It is a straightforward algorithm to find $k$ distinct paths from $s$ to $t$ by using Dijkstra's algorithm iteratively with modified link weight. The link weight increase is suggested in [14] to be $\omega = |E|$ to prefer as much link-disjointness as possible between paths. When a short route is preferred, however, $\omega$ should set to a small value. We use the latter approach.

Other methods to compute multipaths in algorithm 1 are available, such as [15] or [16]. The way the multipaths are found does not affect the discussion hereinafter.

[1]The problem in concern is an extended problem of graph partitioning. It is well-known in algorithmic graph theory that graph partitioning is NP-hard. Using heuristic algorithms such as Kernighan-Lin [12] is the standard way to solve graph partitioning problems.

The essential part of the heuristic algorithm is the lines 3–6. It allocates the multipaths to controllers one by one according to a cost function. The goal of the cost function is to allocate the multipath into controllers such that the maximum number of links to be monitored by a controller is minimized. With that in mind, we established the following heuristic:

1) A multipath shall be allocated to a controller if that controller already monitors most of the links used by that multipath; and
2) In an optimal allocation, the total number of links monitored by each controller shall be roughly the same.

Therefore, we define the cost function in line 4 as

$$c_i = \alpha\nu_i(M) + \mu_i$$

where $\mu_i$ is the number of links already monitored by controller $i$ at the moment and $\nu_i(M)$ is the number of links in the multipath $M$ that is not yet monitored by that controller, i.e. if $M$ is allocated to controller $i$, the total number of links monitored by controller $i$ would become $\mu_i + \nu_i(M)$. Parameter $\alpha$ adjusts their weight in the cost function. When $\alpha \approx 0$, we ignore the benefit of reusing existing links in a controller. When $\alpha \approx \infty$, however, we do not require the controllers to be balanced. This usually yields the result that almost all multipaths are allocated to the same controller, which can be explained by the Matthew's effect on the allocation process of lines 3–6. We empirically found that $\alpha$ between 4 to 8 gives a good result. We set $\alpha = 4$ in our experiments but the wide range of appropriate values for $\alpha$ suggests that it is not very sensitive.

### B. Partition-path approach

Another way to allocate multipaths into controllers, the *partition-path approach*, is available. Its idea is that, if a controller is already monitoring certain links, we can find the $k$-multipath between the source-destination pair $(s, t)$ that uses those links as long as it is possible. Therefore, in this approach, we first partition the links into $q$ controllers as their preferred links. Then the multipath connecting $s$ to $t$ is computed individually in each controller, with preference given to certain links. Algorithm 2 illustrates the idea.

The algorithm begins with a procedure to randomly partition the set of links $E$ into the $q$ controllers such that each controller $i$ covers a subset $\mathcal{E}_i \subset E$ preliminarily (lines 1–5). Then for each source-destination pair $(s, t)$, it finds a $k$-multipath on each controller $i$ with the preference to using links in $\mathcal{E}_i$. The same path-finding algorithm is used in Algorithm 2 as in Algorithm 1. Note that, the links in $\mathcal{E}_i$ affects the path-finding algorithm by changing the initial link weight only. In the other part of the algorithm, such as the cost function in line 10, it is not involved. The same cost function as in section III-A is used here.

## IV. Performance

We applied the heuristic algorithms on different topologies from the Rocketfuel project [17] to evaluate its performance. We use Rocketfuel topologies as there is no detailed data center topologies available publicly. We also evaluate the algorithm with a fat tree topology, which is likely to be used in data center networks, in section IV-D.

**Algorithm 2:** Partition-path heuristic algorithm

**Data**: Network $G = (V, E)$, $q$ = number of controllers
/* Partition links to controllers preliminarily */
1 **foreach** $i := 1$ **to** $q$ **do** $\mathcal{E}_i = \emptyset$
2 **foreach** $e \in E$ **do**
3 $\quad i :=$ random integer in $\{1, \ldots, q\}$
4 $\quad \mathcal{E}_i := \mathcal{E}_i \cup \{e\}$
5 **end**
/* Enumerate multipaths and allocate into controllers */
6 **foreach** $s, t \in V$ *in random order* **do**
7 $\quad$ **foreach** $i := 1$ **to** $q$ **do**
8 $\quad\quad$ Set link weight $w(e) = \begin{cases} 1 & \text{for all } e \in \mathcal{E}_i \\ \psi & \text{for all other } e \in E \end{cases}$
9 $\quad\quad M_i :=$ $k$ paths joining $s$ to $t$
10 $\quad\quad c_i :=$ cost of adding multipath $M_i$ to controller $i$
11 $\quad$ **end**
12 $\quad$ Allocate multipath $M_j$ to controller $j = \arg\min c_j$
13 $\quad \mathcal{E}_j := \mathcal{E}_j \cup \{e :$ for all links $e$ in $M_j\}$
14 **end**

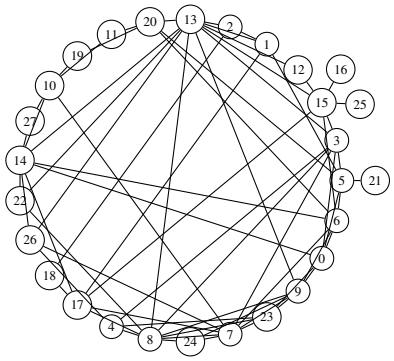


Fig. 2. Topology of an irregular network with 28 nodes and 66 links.

### A. Size of controllers

We use $q = 4$ controllers on a network of 28 nodes and 66 links. The topology is illustrated in Fig. 2. One configuration of the four controllers computed by algorithm 1 is depicted in Fig. 3, with each controller monitoring 45–47 links. From the figure, we found quite significant overlap on the nodes and links monitored by each controller. Some links appeared in all controllers, as they are critical links for the connection of the network. Some other links are less important and appeared in one controller only. The large number of overlap is unavoidable when devolved controllers are used. In fact,

*Theorem 1:* When devolved controllers are used, there is either a controller that covers all nodes, or any single node is covered by more than one controller.

Since for any node $v$, if it is covered by only one controller, then for any flow $(v, u)$ to be routable, node $u$ must also covered by that controller. Therefore that controller must cover all nodes on the network. □

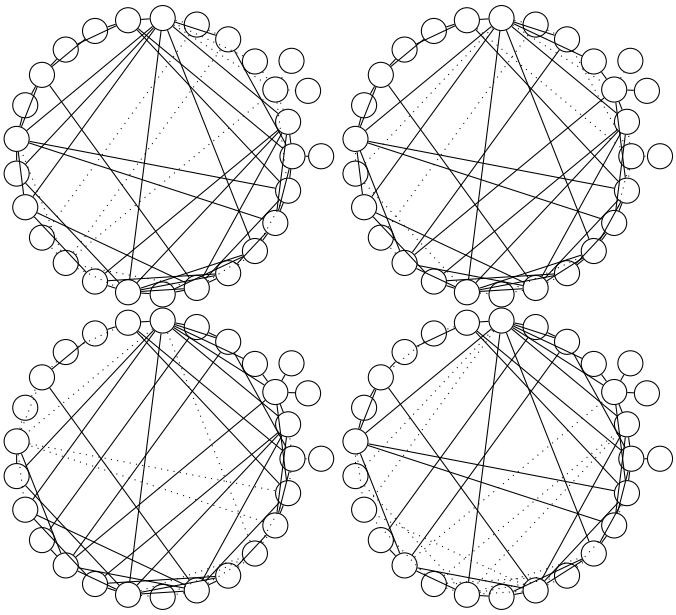

Fig. 3. Links monitored by each of the four controllers as suggested by algorithm 1. Critical links are more likely to be included in multiple controllers, whereas less important links are appeared in only one controller.

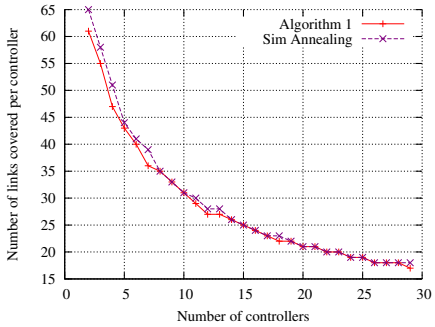


Fig. 4. Number of links covered per controller vs number of controllers in topology of Fig. 2, comparing Algorithm 1 and simulated annealing

| Topology | # nodes | # links | Algo. 1 | Sim. Annealing |
|---|---|---|---|---|
| 1 (Fig.2) | 28 | 66 | 47 (0.1s) | 51 (69.9s) |
| 2 | 108 | 141 | 114 (24.4s) | 140 (1387.4s) |
| 3 | 53 | 456 | 204 (1.1s) | 226 (301.6s) |
| 4 | 44 | 106 | 77 (0.5s) | 92 (186.6s) |
| 5 | 51 | 129 | 97 (0.9s) | 112 (239.9s) |
| 6 (Fig.7) | 45 | 108 | 83 (0.03s) | 83 (46.3s) |

TABLE II
COMPARING THE SIZE OF CONTROLLERS IN DIFFERENT TOPOLOGIES, WITH TIME TAKEN BY EACH SOLVER SHOWN IN BRACKETS.

In other words, there must be significant overlap if we want to reduce the scope of the network that each controller monitors. In fact, we can reduce the number of links monitored by each controller if we use Algorithm 2. Applying to the same topology in Fig. 2, each controller monitors only 29–31 links, which is significantly less. This better result, however, comes with the price that the route found by Algorithm 2 is longer. The average hop count of a path (the mean number of hops over all $kn(n-1)$ paths computed) is 3.5 in Algorithm 2 whereas that in Algorithm 1 is 2.6. The lengthened route may not be favorable in data center networks, however.

To compare the result, we computed a configuration with the same set of multipaths using the time-consuming *simulated annealing* process[2]. The result, presented in Fig. 4, turns out is no better than that obtained by the heuristic algorithms despite the longer time it took. Indeed, the heuristic algorithms often gives a slightly better solution than simulated annealing.

We also applied the algorithm on several different topologies of different number of nodes and links from Rocketfuel. Due to space limitation, we do not show their topologies here but TABLE II shows the maximum number of links covered by a controller resulted from the heuristic algorithm compared to that from simulated annealing. It confirms that the controllers covers around 60-80% of links on the network when $q = 4$ and the result provided by Algorithm 1 is at least as good as that obtained by simulated annealing.

[2]The simulated annealing process is to replace the loop in lines 3–6 only in Algorithm 1. The multipath $\mathcal{M}$ in the comparison are exactly the same for a fair contrast. It may seem counter-intuitive that the well-established simulated annealing technique does not produce better solution. Partly this can be attributed to the choice of parameters such as the cooling function used. More importantly, however, is because our algorithm work 'smarter' than simulated annealing as our cost function guides toward optimality whereas the latter is simply a brute-force search.

### B. Number of controllers and the effect on the size of coverage

In order to reduce the number of links covered by any controller, an intuitive way is to use more controllers. We applied Algorithm 1 with various $q$ to three different irregular topologies. Fig. 5 plots the result.

Obviously, $q = 1$ shows the total number of links in the network. In Fig. 5, the curves show a general decreasing trend. In fact, the curves are decreasing geometrically. This suggests that although we can reduce the size of a controller, there is an overhead: As $q$ increases, the average number of controllers monitoring a link also increases. This means the monitoring traffic, although small, also increases with $q$. This is the trade-off that we have to consider when devolved controllers are used in place of a single centralized controller.

### C. The effect on the number of paths

While the parameter $q$ affects the number of links covered by each controller, the parameter $k$, i.e. the number of paths to find between a source-destination pair, does not have a significant effect on it. This is shown by Fig. 6. The figure plots the number of links covered by a controller against the number of controllers using the topology of Fig. 2, but with the parameter $k$ varied. We examined with $k$ in ranges of 1 to 10 and also some larger values. The varied value of $k$ does not produce a significantly different result between each other. This is explained by the fact that each link on the network is reused for fairly large number of times in different flows $(s, t)$. When we pick a multipath from a controller, it is likely that every links on this multipath are also used by

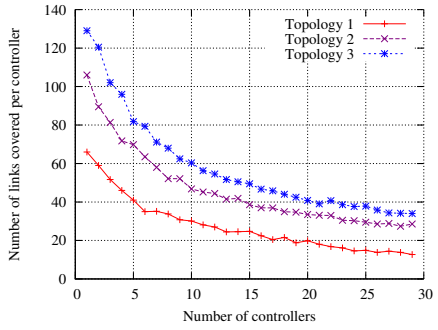


Fig. 5. Number of links covered per controller vs number of controllers in three different topologies from Rocketfuel

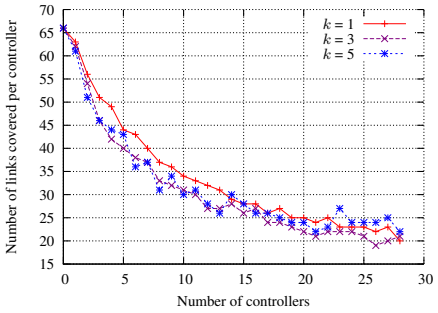


Fig. 6. Number of links covered per controller vs number of controllers in topology of Fig. 2 with different value of $k$

another multipath from the same controller. In this sense, if we increase the multiplicity $k$, the additional paths also likely using the links already covered by the controller.

### D. Using partition-path approach on regular networks

As mentioned in section IV-A, Algorithm 2 produces a better result because it has a path-finding algorithm that fits the path into the controller, in the expense of resulting in a longer route. This weakness of partition-path algorithm can be removed on regular topology networks such as fat tree, which according to [18], has been suggested to use in data center networks. It is because in a regular topology, we know a priori the length of an optimum route and it also provides enough number of distinct paths of the *same length* between a source-destination pair.

To illustrate the idea, we show a 3-layer fat tree network built with switches of 6 ports in Fig. 7. It is trivial to see that, given a pair of hosts in a different subtree, there are $(6/2)^2 = 9$ distinct paths (each pass through a distinct core switch) between them that passes through 5 switches. In such a network, no path that traverses more than 5 switches is optimal. With such knowledge, we can modify the path-finding algorithm used in Algorithm 2 (line 9) to enforce a solution of fixed-length path. Note that such modification only works on regular topologies like fat tree or Clos network. In fact, the modified path-finding algorithm can be used in place of that in line 2 of Algorithm 1 as well.

We also modified line 2 in Algorithm 2 slightly so that only the links connecting core and aggregation switches are partitioned into $\mathcal{E}_i$. This is a reasonable modification considering that in a fat tree network (see Fig. 7), we fixed the whole path between two nodes when we fixed the links that it uses connecting the core and aggregation switches.

Applying the path-partition and partition-path algorithms to the network in Fig. 7 with $q = 4$, we find the coverage per controller to be 83 and 56 links respectively over a total of 108 links. Both Algorithm 1 and Algorithm 2 yield an average hope count of 3.8, due to the modified path-finding algorithm.

## V. Discussions

### A. The communication between a server and controllers

According to the algorithms aforementioned, the multipath for each $(s, t)$ is installed in only one controller. Therefore, only one among the $q$ controllers can reply to a route request for $(s, t)$. When node $s$ is initiating a flow to $t$, it has to deduce which controller can answer its route request. There are two ways to solve this problem. Firstly, node $s$ can send its request to all $q$ controllers and let the one owns the data to reply. Trivially this solution incurs additional network traffic. The second solution is to have a mapping at node $s$: For each destination $t$, there is a table in $s$ tells which controller contains the route for $(s, t)$. This is a viable solution because the total number of nodes $|V|$ (and the number of destinations $t$ for any node $s$) is usually limited. Moreover, when we configured the controllers, storing the mapping information of $\{(s, t), \forall t \in V\}$ to $s$ is just one step further with the existing information.

### B. Path-partition vs partition-path

Section IV-A mention that path-partition algorithm is inferior to partition-path algorithm in terms of the size of controllers produced. However, only path-partition algorithm can guarantee shortest-path routes because the path-finding algorithm is not interfered by the configuration of controllers.

When regular topologies are used, such as fat tree networks, we can have a modified path-finding algorithm to ensure shortest-path routes are found. This makes the partition-path algorithm favorable. Therefore, it is interesting to see that path-partition algorithm is suitable for use with irregular networks while partition-path algorithm is good for regular networks.

### C. Precompute routes

Using pre-computed routes in this paper is intentional. Assume each controller covers only a part of the network and a flow's route is computed dynamically when the request is arrived. It is hard to guarantee that, among the $q$ controllers, there must be one can fulfill any route request. The role of pre-computed multipaths is therefore a verifier to guarantee a controller is responsible for any possible flow.

### D. Link failures

While we do not address the actual operation of devolved controllers in a network, it is expected that whenever there is a link failure, i.e. a topology change, something have to be done in the controllers to reflect this change. This could be disabling certain paths (among the $k$ multiple paths of

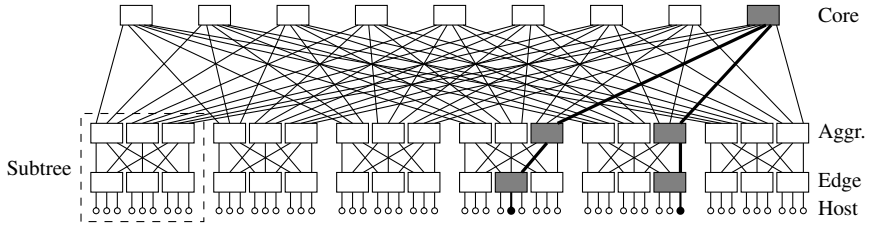


Fig. 7. A fat tree topology built with switches of 6 ports

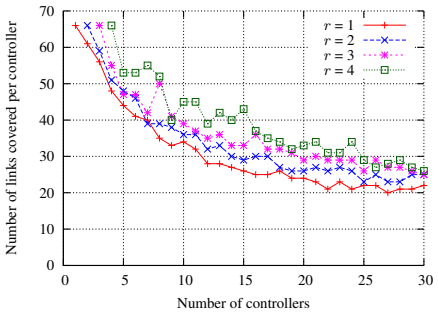


Fig. 8. Number of links covered per controller vs number of controllers in topology of Fig. 2 with different value of $r$

the same source-destination pair), or reconfiguration of the network. This overhead could be large and intensive. In order to provide a prompt reaction, therefore it is essential to keeping the number of links managed by a controller small. This justifies our objective in the optimization.

### E. Redundancy

While it is possible for more than one controller that can respond to a route request, the algorithms in section III does not guarantee this. One way to ensure redundancy is to modify line 6 of algorithm 1 or line 12 of algorithm 2, so that a path is added to $r > 1$ controllers. Usually $r = 2$ is sufficient for resilience. In Fig. 8, we plot the number of links covered by each controller in different values of $r$ in topology of Fig. 2, using the modified path-partition algorithm. Trivially, as the degree of redundancy $r$ increases, the number of links covered by a controller increases. The increment, however, is moderate due to the overlap of link coverage between controllers. In other words, we can have redundancy at just a small price. More details on the redundancy design, its overhead, and an example showing its mechanism would be in a future paper.

## VI. Conclusion

The focus of this paper is to see the possibility of using multiple small independent controllers instead of a single centralized omniscient controller to manage resources. We use flow routing as an example to see how we can use multiple controllers to assign routes to flows base on dynamic network status. The main reason to avoid a single controller is because of scalability concern. Therefore, we forbid our controllers to have the complete network topology information in run time, and introduced the concept of *devolved controllers*. Furthermore, we propose algorithms that aims to limit the network topology information stored in the controllers.

Our result shows that, devolved controllers are possible. We proposed two heuristic algorithms to limit the size of each controller. Although they do not seek for a globally optimal solution, their results are as good as simulated annealing solvers but much faster. The heuristic algorithms, path-partition and partition-path algorithms, are found to be suitable for irregular and regular networks respectively. Such difference is due to the fact that, in regular networks, we can easily estimate the length of route a priori.

In computer networks such as data center or compute clouds, controllers are often used, such as for security policy control, resource allocation, billing, and so on. This paper is a precursor to a new design direction on the use of controllers, such that they can scale out.

## References

[1] M. Al-Fares *et al.*, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010.
[2] F. Chang *et al.*, "Bigtable: A distributed storage system for structured data," in *Proc. OSDI*, 2006.
[3] M. Casado *et al.*, "Ethane: Taking control of the enterprise," in *Proc. SIGCOMM*, 2007.
[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th OSDI*, San Francisco, CA, Dec. 2004.
[5] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *Proc. ACM SOSP*, Bolton Landing, New York, USA, Oct. 2003.
[6] A. Greenberg, N. Jain, S. Kandula *et al.*, "VL2: A scalable and flexible data center network," in *Proc. SIGCOMM*, 2009.
[7] R. N. Mysore *et al.*, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. SIGCOMM*, 2009.
[8] B. Schwartz *et al.*, *High Performance MySQL*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2008.
[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM CCR*, vol. 38, no. 2, pp. 69–74, Apr. 2008.
[10] N. Gude *et al.*, "NOX: Towards an operating system for networks," *ACM SIGCOMM CCR*, vol. 38, no. 3, pp. 105–110, Jul. 2008.
[11] N. McKeown, Personal communication, 2010.
[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Sys Tech Journal*, vol. 49, pp. 291–307, 1970.
[13] N. L. Johnson and S. Kotz, *Urn Models and Their Application*. New York: John Wiley & Sons, 1977.
[14] J. Mudigonda *et al.*, "SPAIN: Design and algorithms for constructing large data-center ethernets from commodity switches," HP, Tech. Rep. 2009-241, 2009.
[15] E. Minieka, *Optimization algorithms for networks and graphs*. New York: M. Dekker, 1978.
[16] R. Guérin and A. Orda, "Computing shortest paths for any number of hops," *Trans. on Networking*, vol. 10, no. 5, pp. 613–620, Oct. 2002.
[17] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," in *Proc. SIGCOMM*, 2002.
[18] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. SIGCOMM*, 2008.