Poster: Network Performance Upgrade by Cut Spanners

Guy Rozenberg and Michael Segal, *IEEE, Senior Member*
School of Electrical an Computer Engineering
Ben-Gurion University, Beer-Sheva, Israel

***Abstract*—In this paper, we introduce a new spanner algorithm which is based on computation of minimum cuts, and insertion of the edges crossing the cut to the spanner. The aim is to decrease the number of active links in the network while still maintaining the ability of the SDN (software defined networking) controller to perform load balancing. The spanner also can be used in order to reduce the running time of the SDN centralized routing algorithm to use. We present an algorithm to dynamically maintain the spanner under link insertion, deletion and changed weight. The analysis and simulation results show the superiority of our approach in many cases.**

***Index Terms*—spanner, centralized networking, minimum cut**

I. Introduction

Software-Defined Networking (SDN) technology is an approach of computer network that facilitates management and enables programmatic efficient network configuration to improve network performance and monitoring. It is based on centralizing network intelligence in one network component.

For centralized networks, several algorithms for increasing network utilization have been recently proposed, based on SDN architecture. All these approaches use the Multi-Commodity Flow (MCF) problem formalization to maximize the link utilization. The best known solution for the MCF problem is from [2], where the running time of the algorithm (for the explicit version) increases in a quadratic ratio to the number of edges. The aim is to decrease the number of edges in the network, by creating a spanner, while still maintaining the ability of the SDN controller to perform load balancing.

In the traditional spanner algorithm, the subgraph is constructed according to the edge's weight. The problem, with respect to sub-graph creation for the SDN routing algorithm to use, is that we would like to create a subgraph with the maximum flow (or minimum cut) between any two vertices and use the subgraph as an input to SDN routing algorithm.

In this manuscript, we propose to build a *min cut spanner*, using the fact that minimum cut guarantees a certain amount of bandwidth that can be used by the centralized routing algorithm. In order to cope with the dynamic nature of the network traffic, we also present an algorithm to dynamically maintain the spanner under edge insertion, deletion and changed weight.

II. Algorithm and theoretical results

A. Min cut spanner algorithm

We build the spanner using recursive calls of the minimum cut algorithm. At each iteration of the algorithm, we shall find the minimum cut of the graph, and add the edges crossing the cut to the spanner. After each iteration of finding the minimum cut, we increase the weights of the edges crossing the cut by $W = \sum_1^{\tilde{n}} w_i \cdot \frac{1}{\tilde{n}}$, where $w_1, \ldots, w_{\tilde{n}}$ are the weights of the edges crossing the cut and $\tilde{n}$ is their number. In this work the weights of the edges are based on the *Current-Flow Betweenness Centrality* $C_{CB}(e)$ of the edge e which is defined as the amount of current that flows through e averaged over all vertex pairs. The edge's weight in the graph G is $w(e) = \frac{1}{C_{CB}(e)}$ since we want that the recursive run of the minimum cut algorithm will first pick the edges with high value of betweenness.

Algorithm 1: *min cut spanner* algorithm

```
Input: undirected weighted graph
       $G = (V, E, W = \frac{1}{C_{CB}})$.
Output: $min\ cut\ spanner$, $min\ cut\ list$.
1 Build an empty graph $G'$ with all the vertices from $G$.
2 Remove all the vertices from $G$ with degree 1, and
    add their edges to $G'$.
3 Calculate the Minimum cut $\mathcal{C}$.
4 while $G'$ is not connected do
5     Add to $G'$ the edges crossed by the cut, that were
        not added to $G'$ before.
6     Add $\mathcal{C}$ and $w(\mathcal{C})$ (relative to the original graph) to
        $min\ cut\ list$ if at least one edge was added to $G'$.
7     Increase the weight of the edges in $G$ crossing $\mathcal{C}$
        by $W = \sum_1^{\tilde{n}} w_i \cdot \frac{1}{\tilde{n}}$.
8     Recalculate $\mathcal{C}$.
9 end
```

Running time: The algorithm to construct *min cut spanner* is composed of several iterations of the minimum cut algorithm run. Gomory and Hu [4] have shown that in a weighted graph $G = (V, E)$ with n vertices there are only $n - 1$ different minimum cut values. Thus, the maximum number of minimum cut iterations is $O(n)$. However, since we continue with the algorithm until the spanner graph is connected, the number of iteration would be much smaller. We get that the running time of the *min cut spanner* algorithm is $O(n)$ times the running time of the single minimum cut algorithm. If we use the algorithm of Gawrychowski et al. [3], with a running time of $O(m \log^2 n)$, the running time of the *min cut spanner* algorithm would be $O(nm \log^2 n)$.

B. Dynamic maintenance of the spanner

The next step is to find a way to maintain the *min cut spanner* under edge insertion, deletion or an update in one of the edge's weight. The *min cut spanner* is built by selecting minimum cuts in an increasing order according to their weights. If one of the edges is changed we could change only the cut associated with this edges. But a change in a single edge can cause a cascade of changes to the minimum cuts selected later. We now claim that all the cuts that were found during the run of the *min cut spanner* algorithm are part of the cut tree of the original graph G.

Theorem 4.1. Let $\lambda_{C_1}, \ldots, \lambda_{C_n}$ be the weights of the cuts that were chosen during the build of the *min cut spanner* in relation to the weights of the original graph G. Each one of the weights λ_i is equal to one of the edges in the cut tree of the original graph T_0.

Corollary 4.2. Each min cut $\mathcal{C}_0, \ldots \mathcal{C}_n$, chosen during the run of the *min cut spanner* algorithm, is a $minimum\ u{-}v\ cut$ for some $(u, v) \in V$ on the original graph G.

Reuse of cuts: We shall now present the algorithm which maintains the *min cut spanner* under edge insertion, deletion or weight change. In order to find the non-reusable minimum cuts after a change in the graph G, we need to determine in which order the reusable minimum cuts from the old spanner are inserted to the new spanner G', and where to assign the new cuts in the sequence of minimum cuts. For each case, insertion or deletion, the cuts from the previous cut list $\mathcal{C}$ are inserted to the new spanner G' in an non - decreasing order, according to the weight of the cut, and the weights in G' are increased accordingly, until the first cut $\mathcal{C}_i$ which can not be reused. Now, we need to find if the minimum cut of G_i, the graph after $i-1$ iterations of the *min cut spanner* algorithm, is a new cut, not part of the old spanner, or not and for that the weight of the minimum cut should be evaluated. To do so, the algorithm presented by Karger [5] is used, which finds the weight of the minimum cut within a $(1+\varepsilon)$ multiplicative in $O(m+n(\log^3 n)/\varepsilon^4)$ time. Moreover Karger showed a way to maintain an $O(\sqrt{1+2/\varepsilon})$ approximation of the cut's weight under insertion at a cost of $\tilde{O}(n^{\varepsilon})$ time. We shall repeat the algorithm of the static *min cut spanner*, but at each time we shall find the weight of the G_i minimum cut, λ_{G_i} first. If some of the weights of the reusable cuts are smaller then $\lambda_{G_i} \cdot O(\sqrt{1+2/\varepsilon})$, we shall add them to the spanner, and increase the weight of the graph accordingly, else we shall find the minimum cut of G_i itself. We continue with this procedure until the spanner G' is connected.

Edge insertion / weight increase:

Algorithm (2) depicts the dynamic spanner algorithm in the incremental scenario. Algorithm for edge deletion/weight decrease is similar and not presented here due to lack of space.

The runtime of the dynamic *min cut spanner* is $T = O(m+n(\log^3 n)/\varepsilon^4) \cdot N_{\mathcal{C}} + O(m \log^2 n) \cdot (N_{\mathcal{C}} - N_{reuse})$, where the number of cuts that were reused in the dynamic algorithm is N_{reuse} and total number of cuts in $minimum\ cut\ list$ is $N_{\mathcal{C}}$.

Algorithm 2: *min cut spanner* under insertion / weight increase

```
Input: undirected weighted graph $G = (V, E)$.
Changed weight of edge $e_{b,d}$.
$\{\mathcal{C}_i | 0 \le i \le L\}$ minimum cut list.
Output: updated min cut spanner and minimum cut list.
create new $minimum\ cut\ list$.
$i$ = 0.
Build an empty graph $G'$ with all the vertices from $G$.
Remove all the vertices from $G$ with degree 1, and add their edges to $G'$.
while $\mathcal{C}_i$ does not cross $e_{b,d}$ do
    Add to $G'$ the edges crossed by the cut $\mathcal{C}_i$.
    Add $\mathcal{C}_i$ and $w(\mathcal{C}_i)$ (relative to the original graph) to the new $minimum\ cut\ list$.
    Increase the weight of the edges in $G$ crossing $\mathcal{C}_i$ by $W = \sum_1^{\tilde{n}} w_i \cdot \frac{1}{\tilde{n}}$.
    $i = i + 1$.
end
Remove from the $minimum\ cut\ list$ all the cuts that cross $e_{b,d}$.
while G' is not connected do
    Find $\lambda \cdot O(\sqrt{1+2/\varepsilon})$ - the approximated weight of the minimum cut of $G$, $\lambda$, using [5].
    if $\lambda \cdot O(\sqrt{1+2/\varepsilon}) > \lambda_{\mathcal{C}_i}$ then
        add $\mathcal{C}_i$ to $G'$ and to the new $minimum\ cut\ list$.
        $i = i + 1$.
    end
    else find the minimum cut of the graph $G$, add it to the spanner $G'$ and to $minimum\ cut\ list$ ;
    Increase the weight of $G$ by $W = \sum_1^{\tilde{n}} w_i \cdot \frac{1}{\tilde{n}}$ according to the added cut.
end
```

III. Empirical results

We simulated the algorithms described above using three useful data center network topologies - dragonfly, hypercube and torus with different topology sets parameters. For each test and topology set we made 100 repetitions, every time randomly selecting the edge's parameters.

A. Number of edges and the average stretch factor

First we compared the number of edges and the average stretch factor with a spanner built using the classical spanner algorithm presented by Althöfer et al. [1], both of which with the same stretch factor. In order to find the average stretch factor we introduced a new concept called *local stretch factor* t_x, which is the stretch factor of a vertex v. The average stretch factor is then the average over the lower half of the *local stretch factors* of every vertex. The following graphs in Fig. 1 present the average stretch factor in the x-axis versus the number of edges in the y-axis, for the two spanners.

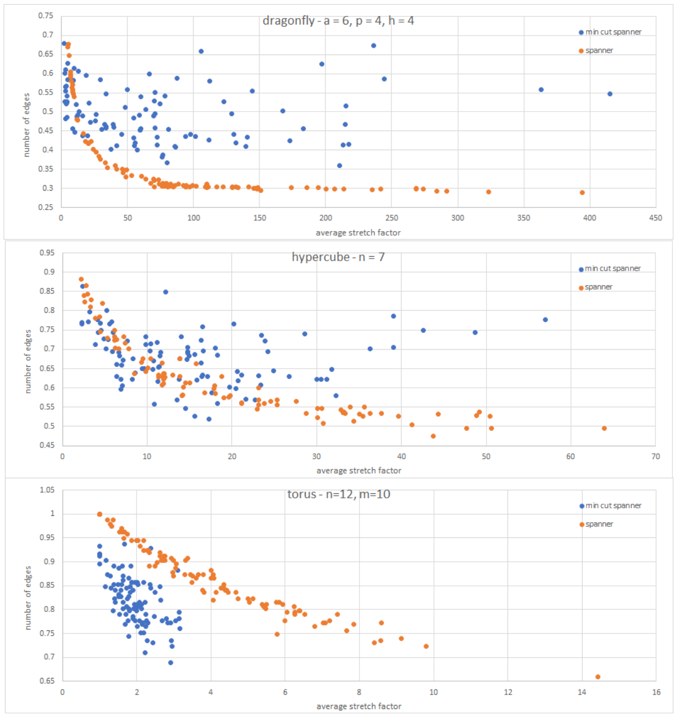

Fig. 1. sparseness test results.

Topology	Topology parameters	classic spanner average flow	min cut spanner average flow
dragonfly	a = 5, p = 3, h = 3	7.812	31.886
	a = 6, p = 3, h = 3	7.512	28.316
	a = 6, p = 4, h = 4	3.416	19.675
hypercube	n = 5	8.995	14.764
	n=6	3.449	8.216
	n=7	1.430	5.604
torus	n = 10, m = 10	0.838	1.432
	n = 10, m = 12	0.573	1.195
	n = 12, m = 10	0.664	1.31

TABLE I
AVERAGE FLOW COMPARISON

Topology	Topology parameters	reused cuts $\Delta > 0$	First unusable cut for $\Delta > 0$	reused cuts $\Delta < 0$
Dragonfly	a = 5, p = 3, h = 3	0.89	0.433	0.043
	a = 6, p = 3, h = 3	0.91	0.471	0.034
	a = 6, p = 4, h = 4	0.91	0.46	0.067
Hypercube	n = 5	0.861	0.42	0.067
	n = 6	0.904	0.43	0.034
	n = 7	0.933	0.438	0.017
Torus	n = 10, m = 10	0.93	0.444	0.079
	n = 10, m = 12	0.943	0.465	0.096
	n = 12, m = 10	0.952	0.414	0.108

TABLE II
NUMBER OF REUSED CUTS

As seen from the results, the average stretch factor of the *min cut spanner* is at least as good as, and most of the times better than, the stretch factor of the classical spanner algorithm. However, a variation is noticeable between the topologies in the number of edges. This can be explained by the average degree of each topology. In the Dragonfly topology, the degree of each vertex is $d = a + h - 1$, which according to the chosen topology parameters, is much bigger than the degree of the hypercube, $d = n$, and the degree of the torus $d = 4$.

B. The flow of the spanner

Our next test was to compare the amount of flow made available by the *min cut spanner*, and compare it with the classical spanner. Instead of calculating $\frac{n(n-1)}{2}$ maximum flows (*or s-t minimum cut*) between any source and a target, we computed the Gomory-Hu tree for each spanner. The following Table (I) presents the average flow for each spanner in each topology set. As seen from the table, the average flow in the *min cut spanner* is bigger, for all topologies, from the classical spanner algorithm.

C. Number of reuse cuts

The number of reused cuts were tested on the topologies mentioned above. At each iteration the weight of one edge was increased or decreased randomly and the spanner was dynamically updated. At Table (II) there is a number of reusable cuts divided by the total number of cuts for each incremental and decremental scenario.

When the change of edge weight $\underline{\Delta > 0}$ the majority of cuts can be reused. For the $\underline{\Delta < 0}$ case, only the cuts crossing the changed edge can be reused, which in our experiments was one or two. A noticeable variation is seen between the dragonfly and hypercubes topologies and between torus topology. This occurs because of the sizes of the partition induced by the cuts selected during the construction of the *min cut spanner* algorithm. The minimum cuts selected in the hypercube and dragonfly topologies tend to induce an uneven partition, which has only one or two vertices on one side that leads to no reused cuts. The torus topology has a uniform structure, so the minimum cuts partition create a more even partition and if $(b, d) \subseteq V/U$ and $|U| \approx \frac{n}{2}$ there is a larger probability that $\mathcal{C}_i \subseteq U$.

IV. CONCLUSION AND FUTURE WORK

In this work we gave an algorithm to construct a spanner using the minimum cut algorithm called *min cut spanner*, which can guarantee a certain amount of bandwidth that the centralized routing algorithm can use. We also gave a method to maintain the *min cut spanner* under edge insertion, deletion or an update in one of the edge's weight.

REFERENCES

[1] Althöfer, I., G. Das, D. Dobkin, and D. Joseph. "Generating sparse spanners for weighted graphs", In *SWAT*, pp. 26-37, 1990.

[2] Karakostas, George. "Faster approximation schemes for fractional multicommodity flow problems", *ACM Transactions on Algorithms*, 4, no. 1, pp. 1–17, 2008.

[3] Gawrychowski, Paweł, Shay Mozes, and Oren Weimann. "Minimum Cut in $O(m \log^2 n)$ Time", arXiv preprint arXiv:1911.01145, 2019.

[4] Gomory, Ralph E., and Tien Chung Hu. "Multi-terminal network flows", *J. of the Soc. for Ind. and App. Math.*, 9, no. 4 (1961): 551-570.

[5] Karger, David R. "Using Randomized Sparsification to Approximate Minimum Cuts." In *SODA*, vol. 94, pp. 424-432. 1994.