

# Two-Aggregator Network Topology Optimization with Splitting

Soham Das and Sartaj Sahni

Department of Computer and Information Science and Engineering

University of Florida

Gainesville, USA

Email: {sdas, sahni}@cise.ufl.edu

**Abstract**—We consider the problem of data aggregation using two aggregators. We assume that the source racks can split the data they need to send to the aggregators across multiple paths. We show that obtaining a topology that minimizes aggregation time is NP-hard for  $k = 2, 3, 4$ , where  $k$  the degree of ToR (top-of-rack) switches. We also show that an optimal topology can be computed in polynomial time for  $k = 5$  and 6 and conjecture this to be the case when  $k > 6$  as well. Experimental results show that, when  $k = 6$ , our topology optimization algorithm reduces the aggregation time by as much as 83.3% and reduces total network traffic by as much as 99.5% relative to the torus heuristic, proposed by [1].

**Keywords**—Data center networks; software defined networking; big data applications; map-reduce tasks

## I. INTRODUCTION

Large data centers are comprised of thousands of racks that are interconnected via top-of-rack (ToR) switches. Software defined networking (SDN) may be used to dynamically reconfigure the data center network to improve the performance of big-data applications. The reconfiguration time is negligible when compared to the total execution time of the applications. The problem of data aggregation is encountered, for example, in the aggregation phase of a MapReduce operation. In this phase, data residing in several racks are to be aggregated into one or more specified racks called aggregators. This paper focuses on the construction of tree topologies that minimize aggregation time when there are two aggregators. The degree of each node, where a node represents either an aggregator or source rack, in the aggregation topology is constrained by the number  $k$ ,  $k \geq 2$ , of optical links at each ToR switch. Wang et al. [1] have observed that the aggregation time in many big-data applications is a dominant component of overall execution time. Hence the need to minimize aggregation time.

Consider the case when data is to be aggregated from 8 source racks to 2 aggregator racks. Figure 1(a) shows a possible aggregation tree topology that may be used for this purpose by each of the 2 aggregators. In this figure,  $A_1$  and  $A_2$  are the aggregator racks and  $X(d)$ ,  $X \in \{B, C, D, E, F, G, H, I\}$ , denotes a source rack that has  $d$  units of data that are to be aggregated into the corresponding aggregator. Among the source racks,  $B$  and  $C$  have data to be sent to both aggregators and so, these racks are present in both the aggregation trees  $U_1$  and  $U_2$ . Racks  $D$ ,  $E$  and  $F$  have data to be sent to only  $A_1$  and racks  $G$ ,  $H$  and  $I$  have data to be sent to only  $A_2$  and so, these racks are present in only one aggregation tree. The edges denote optical links between pairs of ToR switches. The shown aggregation trees can be realized when  $k \geq 6$  as rack  $B$  uses 3 optical links in each tree.

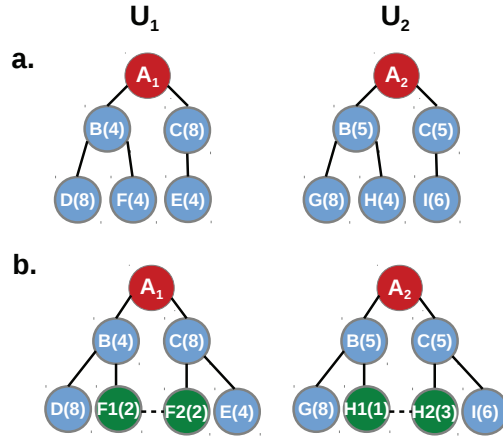


Fig. 1: Example aggregation tree topologies

For the data aggregation, each node in the aggregation tree sends its data along with the data aggregated from its children to its parent, which in turn sends the aggregated data to its parent and in this way data from all the sources reaches the aggregator at the root. Thus each node sends its data using the unique path from that node to the aggregator, which is at the root. Assume that the data is packetized and nodes can send and receive packets at the same time. Moreover nodes can receive packets through multiple links at the same time. If each optical link has a bandwidth of 10 Gbps, the data from  $B$ ,  $D$ , and  $F$  can be aggregated into  $A_1$  in  $(4 + 8 + 4)/10 = 1.6s$  and that from  $C$  and  $E$  can be aggregated in  $(8 + 4)/10 = 1.2s$ . Since  $A$  receives data from  $B$  and  $C$  in parallel, the aggregation time is  $\max\{1.6, 1.2\} = 1.6s$ . Similarly, the data from  $B$ ,  $G$ , and  $H$  can be aggregated into  $A_2$  in  $(5 + 8 + 4)/10 = 1.7s$  and that from  $C$  and  $I$  can be aggregated in  $(5 + 6)/10 = 1.1s$ . Since  $A_2$  receives data from  $B$  and  $C$  in parallel, the aggregation time is  $\max\{1.7, 1.1\} = 1.7s$ . So the total aggregation time is  $\max\{1.6, 1.7\} = 1.7s$ .

Figure 1(b) shows an alternative way to aggregate when  $k = 6$ . In  $U_1$ , rack  $F$  has been split into two nodes  $F1$  and  $F2$  (we use a broken line to connect nodes that represent the same rack; solid lines denote optical links). So, rack  $F$  uses two paths ( $F1$  to  $A_1$  and  $F2$  to  $A_1$ ) to route its data to the aggregator; each path carries 2 units of  $F$ 's data. The path from  $F1$  to  $A_1$  goes through  $B$  and that from  $F2$  goes through  $C$ . So the aggregation time for  $A_1$  reduces to  $\max\{(4 + 8 + 2)/10, (8 + 2 + 4)/10\} = \max\{1.4, 1.4\} = 1.4s$ . Similarly in  $U_2$ , rack  $H$  has been split into  $H1$  and  $H2$ ,  $H1$  sends 1 unit

of data through  $B$  to  $A_2$  and  $H_2$  sends 3 units of data through  $C$  to  $A_2$ . So the aggregation time for  $A_2$  reduces to  $\max\{(5+8+1)/10, (5+3+6)/10\} = \max\{1.4, 1.4\} = 1.4s$ . Hence, the overall aggregation time reduces to  $\max\{1.4, 1.4\} = 1.4s$ .

A strategy that is often used in practice to reduce overall aggregation time is to distribute the tasks randomly across racks during the Map phase of a MapReduce. However, dynamic reconfiguration of the network using SDN is far more effective. A random distribution only enhances expected performance and does not improve worst-case performance. For example, in Figure 1(a), no distribution of the tasks can reduce aggregation time to  $1.4s$  (without splitting) as there is no partition of tasks B-F in  $U_1$  whose size is 14. The reduction to  $1.4s$  is made possible by splitting racks as shown in Figure 1(b).

The *total network traffic* is the sum of the amount of data moved through each optical link. For the topology of Figure 1(b) the total traffic in  $U_1$  is 14 (link  $A_1B$ ) + 14 (link  $A_1C$ ) + 8 (link  $BD$ ) + 2 (link  $BF_1$ ) + 2 (link  $CF_2$ ) + 4 (link  $CE$ ) = 44Gb. Similarly the total traffic in  $U_2$  is 14 (link  $A_2B$ ) + 14 (link  $A_2C$ ) + 8 (link  $BG$ ) + 1 (link  $BH_1$ ) + 3 (link  $CH_2$ ) + 6 (link  $CI$ ) = 46Gb. So total network traffic is 90Gb. If we swap nodes  $B$  and  $D$  in  $U_1$  and  $B$  and  $G$  in  $U_2$  in Figure 1(b), the total network traffic reduces to 83Gb and the aggregation time still remains  $1.4s$ .

We use the acronym TANTOS to refer to the two-aggregator network topology optimization problem with data splitting. In this problem, we are to construct tree topologies  $U_1$  and  $U_2$  for aggregators  $A_1$  and  $A_2$ , respectively, that minimize the overall aggregation time (i.e., the larger of the aggregation times of  $U_1$  and  $U_2$ ), given the degree  $k$  of ToR switches, set of source racks  $S_1$  that send data to only  $A_1$ , set of source racks  $S_2$  that send data to only  $A_2$ , and set of source racks  $C$  that send data to both  $A_1$  and  $A_2$  along with the corresponding amounts of data to be sent. Each source rack may appear as several nodes in an aggregation tree and thereby split its data traffic utilizing different optical links.

The main contributions of this paper are:

- 1) We prove that TANTOS is NP-hard for  $k = 2, 3$  and 4.
- 2) We prove that TANTOS is polynomially solvable for  $k = 5$  and 6.
- 3) We compare our algorithm for  $k = 6$  with the torus heuristic proposed by Wang et al. [1] experimentally. Experimental results show that our algorithm reduces the aggregation time by up to 83.3% and the total network traffic by up to 99.5%.

The remainder of this paper is organized as follows. Related work is reviewed in Section II. TANTOS is shown to be NP-hard for  $k = 2, 3$  and 4 in Sections III, IV and V, respectively. TANTOS is shown to be polynomially solvable for  $k = 5$  and 6 in Section VI. Our experimental results are presented in Section VII and we conclude in Section VIII.

## II. RELATED WORK

This paper focuses on the two aggregator variant of the problem addressed by us in [3]. In this prior work, we examined single-aggregator network topology optimization with splitting (SANTOS). Consequently, much of the related-work

section of [3] has been reproduced here and we have added in a summary of the new results obtained by us in [3].

A lot of work has been done in configuring the topology of data center networks to enhance the performance of applications [6]-[10]. Our model and work are most closely related to that of Wang, Ng and Shaikh [1]. In [1], Wang et al. describe an “integrated network control for big data applications” that comprises of “OpenFlow-enabled top-of-rack (ToR) switches”. They propose heuristics for the single as well as multiple aggregators variants of the network topology optimization problem.

In [2], we address single aggregator network topology optimization (SANTO) under the constraint that each rack appears as exactly one node in the aggregation tree (i.e., nodes cannot split their data over multiple paths). The problem is shown to be NP-hard and that the approximation ratio of the algorithm of Wang, Ng, and Shaikh [1] is  $(k+1)/2$ , where  $k$  is the degree of ToR (top-of-rack) switches. We propose a SANTO algorithm that is based on the longest processing time (LPT) scheduling rule. The approximation ratio for this algorithm is  $(4/3 - 1/(3k))$  [4], [5]. Further, we show that the aggregation time using the LPT method is never more than that using the algorithm of [1]. Moreover, if the LPT algorithm is used to partition the source racks into  $k$  groups and the racks within each group organized into a subtree of the aggregation tree using the algorithm of Wang et al. [1], then total network traffic is minimized in that subtree.

In [3], we propose two classes of algorithms for SANTOS (single aggregator topology optimization with splitting). The complexity of one of these is  $O(n)$  and that of the other is  $O(n \log k)$ . We develop  $O(n \log n)$  heuristics to reorganize the nodes in any solution to SANTOS in an attempt to minimize total network traffic. Through extensive experiments, we demonstrate the benefit of permitting racks to use multiple data paths to the aggregator (i.e., to split their data). The aggregation time reduces by up to 99% using data splitting.

## III. TANTOS IS NP-HARD WHEN $k = 2$

We observe that when  $k = 2$ , the aggregation tree  $U_1$  ( $U_2$ ) consists of the aggregator  $A_1$  ( $A_2$ ) as root plus at most two subtrees each of which is a chain. So,  $U_1$  and  $U_2$ , each have at most 2 leaves. Further, each intermediate node of a subtree chain requires 2 links while each leaf requires 1 link. Since each rack of  $C$  has to be in both  $U_1$  and  $U_2$ , each rack of  $C$  can be assigned only 1 link in each tree and hence must be a leaf in both trees. Consequently,  $U_1$  ( $U_2$ ) can accommodate at most 2 racks from  $C$ . Hence, when  $|C| > 2$ , data aggregation is not possible using a single pair of trees  $U_1$  and  $U_2$ . Instead, we must do some of the aggregation using one pair of trees and the rest using another. For example, we could do the aggregation for  $A_1$  using a single tree assigning 2 links in  $U_1$  for every rack in  $C$  and then do aggregation for  $A_2$  using another tree  $U_2$  in which all racks of  $C$  are assigned 2 links each. We note that some racks will use only 1 of the 2 links assigned to them.

Since our TANTOS model requires the use for a single pair of trees to concurrently aggregate for both  $A_1$  and  $A_2$ , data aggregation under the TANTOS model is infeasible when  $k = 2$  and  $|C| > 2$ . The following theorem shows that minimizing aggregation time, under the TANTOS model, when  $k = 2$  and  $|C| = 1$  or 2 is NP-hard.

**Theorem 1.** *TANTOS is NP-hard when  $k = 2$  and  $|C| = 1$  or 2.*

*Proof:* We use the partition problem, which is known to be NP-hard. Let  $s_i, 1 \leq i \leq n$  be an instance  $P$  of the partition problem. Without loss of generality, we may assume that  $\sum s_i$  is even as otherwise there can be no partition.

From  $P$ , we create an instance  $T$  of TANTOS with  $k = 2$ ,  $S_1 = S_2 = \{s_i\}$ ,  $C = \{c_1 = \{c, c\}, c_2 = \{c, c\}\}$ , and link bandwidth 1. From the discussion preceding this theorem, it follows that, when  $k = 2$ , in every  $U_1$  and  $U_2$  for  $T$ ,  $c_1$  and  $c_2$  are leaves, the  $s_i$ s are intermediate nodes; and no rack is split. From this, it follows that  $T$  can be aggregated in  $\sum s_i/2 + c$  time iff  $P$  has a partition and that when  $P$  has no partition, the aggregation time is larger. Hence, TANTOS is NP-hard when  $k = 2$  and  $|C| = 2$ .

To prove the theorem for the case when  $|C| = 1$ , we create the TANTOS instance  $T$  with  $k = 2$ ,  $S_1 = S_2 = \{s_i\} - s_1$ ,  $C = \{c_1 = \{s_1, s_1\}\}$ , and link bandwidth 1. Once again, when  $k = 2$ , in every  $U_1$  and  $U_2$  for  $T$ ,  $c_1$  is a leaf, all but at most one of the  $s_i$ s in  $S_1$  and  $S_2$  are intermediate nodes; and no rack is split. Hence,  $T$  can be aggregated in  $\sum s_i/2$  time iff  $P$  has a partition; when  $P$  has no partition, the aggregation time is larger. Hence, TANTOS is NP-hard when  $k = 2$  and  $|C| = 1$ . ■

#### IV. TANTOS IS NP-HARD WHEN $k = 3$

Our proof employs the 3-way partition problem, which we first define and prove to be NP-hard.

**3-way partition problem:** Given  $n$  positive integers,  $s_i, 1 \leq i \leq n$ , partition them into 3 subsets such that each sums to  $\sum s_i/3$ .

**Theorem 2.** *3-way partition is NP-hard.*

*Proof:* Consider any instance  $P2$  of the standard 2-way partition problem. Let  $s_i, 1 \leq i \leq n$  define this instance. We may assume that  $\sum s_i$  is even as otherwise  $P2$  has no partition. To this instance, add the positive integer  $\sum s_i/2$  to obtain an instance  $P3$  of 3-partition. It is easy to see that  $P3$  has a 3-way partition iff  $P2$  has a (2-way) partition. Hence, the 3-way partition problem is NP-hard. ■

In the following, we use the term *terminal* to refer to a point on a rack to which an optical link connects. Each rack has exactly  $k$  terminals and each optical link connects exactly 2 terminals, which are the end points of the optical link.

**Lemma 1.** *Let  $A$  be an aggregation tree that has  $N$  nodes (a node represents the aggregator or a data rack; a split rack appears as two or more nodes). The number of terminals used in  $A$  is  $2(N - 1)$ .*

*Proof:* An  $N$  node tree has  $N - 1$  edges/links and each edge uses two terminals. So,  $2(N - 1)$  terminals are used in  $A$ . ■

**Lemma 2.** *When  $k = 3$ , the number of unutilized/free terminals in an aggregation tree for  $n$  source racks is  $n + 3$ , when no rack is split, and is less than  $n + 3$ , when a rack is split.*

*Proof:* The number of nodes in the aggregation tree is  $n + 1$ , when no rack is split, and is more than  $n + 1$ , when a rack is split. So, the number of terminals in use is at least  $2n$  (Lemma 1). The total number of terminals in the  $n$  source

racks is  $3n$  and that in the aggregator is 3. So, the number of free terminals is at most  $3n + 3 - 2n = n + 3$ . ■

**Theorem 3.** *TANTOS is NP-hard when  $k = 3$ .*

*Proof:* Let  $s_i, 1 \leq i \leq n$  be an instance  $S$  of the 3-way partition problem. We may assume that  $\sum s_i$  is a multiple of 3 as otherwise, a 3-way partition cannot exist. Consider the following instance  $Q$  of TANTOS:  $k = 3$ ,  $S_1 = S_2 = \{t_1, \dots, t_n\}$ ;  $t_i = s_i * (2n + 7)$ ,  $C = \{(1, 1)X(2n + 6)\}$ , and link bandwidth 1. Note that  $|C| = 2n + 6$  and that each rack in  $C$  sends 1 unit of data to each of the 2 aggregators. We shall show that  $Q$  can be aggregated in less than  $\sum t_i/3 + 2n + 7$  time, iff  $S$  has a 3-way partition. Hence, TANTOS is NP-hard when  $k = 3$ .

If  $S$  has a 3-way partition  $A, B, C$  then the sum of the  $s_i$ 's in each partition is  $M/3$ , where  $M = \sum s_i$ . For this 3-way partition, the sum of the  $t_i$ 's in each partition is  $M * (2n + 7)/3$ . Put  $A, B$ , and  $C$ , respectively, in the 3 subtrees of each of the aggregation trees  $U_1$  and  $U_2$ . Note that the aggregator  $A_1$  is the root of  $U_1$  while  $A_2$  is the root of  $U_2$ . Arbitrarily assign one terminal to  $n + 3$  of the racks in  $C$  in  $U_1$  and 2 terminals for the remaining  $n + 3$  racks. The racks of  $C$  that are assigned 1 terminal in  $U_1$  are assigned 2 in  $U_2$  and those assigned 2 in  $U_1$  are assigned 1 in  $U_2$ . The  $n + 3$  racks of  $C$  with 2 terminals may be distributed to any of the 3 subtrees of  $U_1$  in any fashion and the remaining  $n + 3$  become leaves using the  $n + 3$  free terminals (Lemma 2). Note that when a rack with 2 terminals is added to the aggregation tree, the number of free terminals is unchanged. Since the number of racks of  $C$  added to a subtree of either  $U_1$  or  $U_2$  is  $2n + 6$ , the aggregation time for both  $U_1$  and  $U_2$  is less than  $M * (2n + 7)/3 + 2n + 7$ .

Next, suppose that  $Q$  can be aggregated in less than  $M * (2n + 7)/3 + 2n + 7$  time. Consider any pair of aggregation trees  $U_1$  and  $U_2$  for  $Q$  that accomplishes this. When no rack is split in either  $U_1$  or  $U_2$ , the number of terminals used in each is  $2(n + 1 + 2n + 6 - 1) = 6n + 12$  (Lemma 1). Together,  $U_1$  and  $U_2$  use  $12n + 24$  terminals. The total number of available terminals on the  $4n + 6$  source racks and the 2 aggregators in the instance  $Q$  is  $3(4n + 8) = 12n + 24$ , which is sufficient to meet the needs of  $U_1$  and  $U_2$  when no rack is split. When a rack is split, the number of nodes in an aggregation tree is more than  $3n + 7$  and the number of terminals needed in an aggregation tree with a split rack is more than  $6n + 12$  (Lemma 1). Hence, when a rack is split, the  $4n + 6$  source racks and 2 aggregators of  $Q$  do not have enough terminals to form the aggregation trees  $U_1$  and  $U_2$ . So, we may assume that no rack is split in either  $U_1$  or  $U_2$ .

Now, since (a) the aggregation time for  $U_1$  is less than  $M * (2n + 7)/3 + 2n + 7$ , (b) each  $t_i$  is a multiple of  $2n + 7$ , (c) the sum of the  $t_i$ 's in  $U_1$  is  $M * (2n + 7)/3$ , (d) there is no split rack in  $U_1$ , and (e) there are exactly  $2n + 6$  racks of  $C$  each with unit data in  $U_1$ , the sum of the  $t_i$ 's in each subtree of  $U_1$  must be exactly  $M * (2n + 7)/3$  and the sum of the corresponding  $s_i$ 's must be  $M/3$ . Hence,  $S$  has a 3-way partition. ■

#### V. TANTOS IS NP-HARD WHEN $k = 4$

**Theorem 4.** *TANTOS is NP-hard when  $k = 4$ .*

*Proof:* For this proof, we use the standard (2-way) partition problem that is known to be NP-hard. Let  $\{s_1, \dots, s_n\}$



with  $\sum s_i = 2M$  be an instance  $P$  of the partition problem. Consider the following TANTOS instance  $Q$ :  $k = 4$ ,  $S_1 = S_2 = \emptyset$ ,  $C = \{(M+2, M+2), (M+2, M+2), (s_1, s_1), (s_2, s_2), \dots, (s_n, s_n)\}$ , and link bandwidth 1. Let  $LB = M+1$ . We shall show that  $Q$  has an aggregation time of  $LB$  iff  $P$  has a partition; the aggregation time is more than  $LB$  when  $P$  does not have a partition. Hence, TANTOS is NP-hard when  $k = 4$ . It is easy to see that an aggregation time less than  $LB$  is not possible for  $Q$ .

Suppose that  $P$  has a partition. We use the partition to construct 4 subtrees  $T_1, \dots, T_4$  for each of the aggregators  $A_1$  and  $A_2$ . The total number of nodes in  $T_1, \dots, T_4$  is  $n+4$  (excluding the aggregator) and the first two racks of  $C$  are split once each. The number of links associated with each source rack in each of  $U_1$  and  $U_2$  is 2 and each  $T_i$  is a chain of degree 2 nodes terminating in a degree 1 leaf.  $T_1$  ( $T_2$ ) is assigned  $M+1$  units of data from the first (second) rack of  $C$  while  $T_3$  ( $T_4$ ) is assigned the remaining 1 unit. These racks are the leaves of  $T_1, \dots, T_4$ . The racks corresponding to one partition of  $P$  are assigned to  $T_3$  and those to the other partition to  $T_4$ . The aggregation time for  $A_1$  and  $A_2$  is  $LB$ . Suppose there is an aggregation tree  $U_1$  ( $U_2$ ) for  $A_1$  ( $A_2$ ) with aggregation time  $t_1$  ( $t_2$ ) and  $LB = \max\{t_1, t_2\}$ . Note that since  $\min\{t_1, t_2\} \geq LB$ ,  $t_1 = t_2$ . In  $U_1$  ( $U_2$ ) the first two racks of  $C$  must be split exactly once each. Note that if one of these is split twice in (say)  $U_1$ , at least 3 links must be assigned to it in  $U_1$ , which means it has at most 1 link in  $U_2$  and so cannot be split in  $U_2$ . This means that  $t_2 > LB$ . Hence  $U_1$  ( $U_2$ ) has the first two racks of  $C$  occupying 4 leaf positions. In the absence of degree 3 nodes,  $U_1$  has exactly 4 leaf positions because the aggregator degree is 4.

Now, suppose that  $U_1$  has  $q$  source racks with 3 links assigned (note that it can have no source rack with 4 links as this would leave no link for the source rack to use in  $U_2$ ). Suppose that  $j$  of these source racks are split at least once in  $U_1$ . This creates a demand for at least  $j$  additional (i.e., in addition to the 4 from the first two racks of  $C$ ) leaf positions in  $U_1$ . The number of additional leaf positions created by the 3-link source racks that are not split is  $q-j$ . This means that  $q-2j$  leaf positions of  $U_1$  are occupied by racks with 2 links that are each split once or by racks with one link. So, the number of source racks in  $U_1$  with exactly 1 link is at most  $q-2j$ .

Now, in  $U_2$  the  $q$  3-link racks of  $U_1$  have only 1 link each. Since  $U_1$  has at most  $q-2j$  1-link racks,  $U_2$  has at most this many 3-link racks and, hence, at most  $q-2j$  additional (over and above the 4 it has when  $q=0$ ) leaf positions. So, to accommodate the 1-link racks of  $U_2$ ,  $j$  must be 0. In other words, no 3-link rack is split in either  $U_1$  or  $U_2$ .

Next, suppose that  $m$  2-link racks are split in  $U_1$ . This creates a demand of  $2m$  additional leaf positions in  $U_1$ . So, the number of 1-link racks in  $U_1$  is at most  $q-2m$ . Hence, the number of 3-link racks in  $U_2$  is at most  $q-2m$  and so  $U_2$  has at most  $q-2m+4$  leaf positions. But, the number of 1-link racks is  $q$ , which requires  $q$  added (i.e., in addition to the 4 required for the first 2 racks of  $C$ ) leaf positions. So,  $m=0$  and no 2-link rack is split either. Therefore, in  $U_1$  ( $U_2$ ) no rack other than the first 2 of  $C$  is split. Let  $T_1, \dots, T_4$  be the subtrees of  $U_1$ . Let  $T_1$  and  $T_2$  ( $T_3$  and  $T_4$ ) be the subtrees that have the two split portions of the first (second) rack of  $C$ . The sum of the data with the racks in  $T_1$  and  $T_2$  is

$2LB = 2M+2$ . Hence the source racks in  $T_1$  and  $T_2$  other than the first 2 of  $C$  define a partition of  $P$ . ■

## VI. TANTOS WITH $k > 4$

Consider an instance  $T$  of TANTOS. Let  $s_i^1$  be the amount of data the  $i$ th rack of  $S_1$  has to send to  $A_1$  and let  $c_i^1$  be the amount that the  $i$ th rack of  $C$  has to send to this aggregator.  $s_i^2$  and  $c_i^2$  are similarly defined for  $S_2$ ,  $C$ , and  $A_2$ . We assume that the link bandwidth is 1. If the  $i$ th rack of  $C$  is assigned  $j$  links in  $U_1$ , then it can be assigned at most  $k-j$  in  $U_2$  and the time to get  $c_i^1$  ( $c_i^2$ ) data out of rack  $C$  in  $U_1$  ( $U_2$ ) is at least  $\lceil c_i^1/j \rceil$  ( $\lceil c_i^2/(k-j) \rceil$ ). So, the aggregation time for  $T$  is at least  $LB1 = \max_i \{ \min_j \{ \lceil c_i^1/j \rceil, \lceil c_i^2/(k-j) \rceil \} \}$

We may arrive at another lower bound,  $LB2$ , on aggregation time by noticing that  $\sum s_i^1 + \sum c_i^1$  data has to flow into  $A_1$  using at most  $k$  links. So,

$$LB2 = \max \{ \lceil (\sum s_i^1 + \sum c_i^1)/k \rceil, \lceil (\sum s_i^2 + \sum c_i^2)/k \rceil \}$$

Combining these two lower bounds, we obtain the lower bound  $OPT = \max\{LB1, LB2\}$ . We note that  $OPT$  may be computed in  $O(n+km)$  time, where  $n$  is the total number of racks in  $S_1$  and  $S_2$  and  $m$  is the number of racks in  $C$ .

Let  $m_1$  and  $m_2$ , respectively, be the number of racks in  $C$  that send more than  $OPT$  data to  $A_1$  and  $A_2$ . Without loss of generality, we may assume that  $m_1 \geq m_2$  and so  $k > m_1 \geq m_2 \geq 0$ . Hence, the number of  $(m_1, m_2)$  combinations is  $k(k+1)/2$ . Also, from the definition of  $OPT$ , it follows that  $c_i^r \leq (k-1)OPT$ , where  $r \in \{1, 2\}$ .

As we shall see below, when  $m_1 = m_2 = 0$  it is possible to aggregate in  $OPT$  time and the required aggregation trees are polynomially computable. For other values of  $m_1$  and  $m_2$  it may not be possible to aggregate in this much time. For  $k=5$  and 6, we have examined all possible cases for  $m_1$  and  $m_2$  and derived provably optimal aggregation trees. For  $k > 6$ , we conjecture that a similar case analysis can be done resulting in polynomial time algorithms to construct optimal aggregation trees for every  $k > 6$  as well.

### A. $m_1 = m_2 = 0$

We use the depth-first algorithm of [3] to assign the racks of  $S_1$  and  $C$  to the  $k$  subtrees of  $U_1$ . This algorithm assigns racks one-at-a-time in a prespecified order. First, racks are assigned to subtree 1, then to subtree 2, ..., and finally to subtree  $k$ . If the rack currently being assigned causes the total data assigned to the subtree under consideration to exceed the bound  $OPT$ , then the rack is split making the total data assigned to the current subtree equal to  $OPT$  and then continuing the assignment of the current rack into the next subtree. A rack with  $d$  amount of data is split across at most  $\lceil d/OPT \rceil + 1$  subtrees. Within a subtree, racks are placed into a chain top-to-bottom with the last assigned rack being a leaf. Since  $c_i^1 \leq OPT$  for all  $i$ , each  $c_i$  needs at most 3 links in  $U_1$  (1 when it is a leaf and up to 2 when it is the first node of a chain; 2 when it is an intermediate node of a chain). The racks in  $S_1$  need up to  $k$  links each.

When  $k \geq 6$ , each rack of  $C$  is left with at least 3 links that can be used in  $U_2$  and each rack of  $S_2$  has  $k$  links for use in  $S_2$ . Hence, using the same depth-first algorithm, the racks in  $S_2$  and  $C$  can be assigned to the  $k$  subtrees of  $U_2$  so that no subtree is assigned more than  $OPT$  amount of data, each

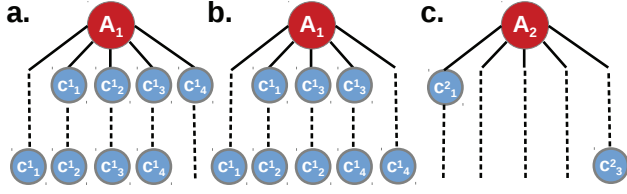


Fig. 2: The case  $m_1 = m_2 = 0$ ;  $k = 5$

rack of  $C$  needs at most 3 links, and each rack of  $S_2$  needs at most  $k$  links. The constructed aggregation trees  $U_1$  and  $U_2$  complete the aggregation in  $OPT$  time.

When  $k = 5$ , a minor variation of the above algorithm yields  $U_1$  and  $U_2$  that satisfy the link constraints and complete the aggregation in  $OPT$  time. We examine the  $U_1$  that results using the depth-first algorithm as above. In the worst-case, 4 of the  $c_i^1$ s are split (see Figure 2(a)). We reorder the racks in the 6 subtree chains so that at most 1 of the split  $c_i^1$ s uses 3 links and at most 1 uses 4 links. This is shown in Figure 2(b) for the case when 4  $c_i^1$  are split in the initial  $U_1$ . The reordered  $U_1$ , for this case, requires 3 links for  $c_1^1$ , 2 for  $c_2^1$ , 4 for  $c_3^1$ , and 2 for  $c_4^1$ . Now, when using the depth-first algorithm to construct  $U_2$ , we begin by ordering the racks in  $S_2 \cup C$  so that  $c_1^2$  is the first and  $c_3^2$  is the last. The depth-first algorithm assures that  $c_1^2$  and  $c_3^2$  are not split and that  $c_3^2$  is a leaf (see Figure 2(c)). So,  $c_1^2$  needs at most 2 links and  $c_3^2$  needs exactly 1; the remaining racks of  $C$  need at most 3 and those in  $S_2$  need at most  $k$ . Hence we have a  $U_1$  and  $U_2$  that satisfy the link constraints and result in an aggregation time of  $OPT$ .

We note that the above construction uses at most 5 links per rack when  $k = 5$  and at most 6 when  $k \geq 6$ . The unutilized links could be used to transform the subtrees of  $U_1$  and  $U_2$  from chains into more bushy subtrees and thereby reduce total network traffic as was done by us in [3].

### B. Other Cases

As mentioned earlier, we have examined the remaining cases when  $k = 5$  and 6 and for each case been able to construct aggregation trees that complete the aggregation in  $OPT$  time or shown that this is not possible; for the latter cases, we have determined the minimum increase  $\Delta$  in  $OPT$  needed to guarantee aggregation in  $OPT + \Delta$  time and then constructed the aggregation trees to aggregate in this much time. This examination leads to polynomial time algorithms for  $k = 5$  and 6. These algorithms construct the optimal aggregation trees by determining which  $(m_1, m_2)$  case (and possibly subcase) the current instance falls into. The complete case-by-case analysis for  $k = 5$  and 6 appears in [11].

Because of the large number of cases involved, we have not done the analysis for larger values of  $k$  but we believe that, in principle, such an analysis could be done for each  $k$ ,  $k > 6$  and we would then obtain a polynomial time algorithm for each such  $k$ .

**Theorem 5.** *TANTOS is polynomially solvable for  $k = 5$  and 6.*

## VII. EXPERIMENTS

We benchmarked our optimal algorithm for  $k = 6$  against an extension of the 2D torus heuristic proposed in [1] for

$k = 4$ . The torus heuristic was originally proposed for a data-shuffling pattern in which each source rack sends data to each aggregator rack. When the number of aggregators is 2, this data shuffling corresponds to  $S_1 = S_2 = \emptyset$ . As a result, the test data used by us also has  $S_1 = S_2 = \emptyset$ . We also note that the data shuffling heuristic of [1] does not do data splitting. In our extension of the torus heuristic of [1], a 3D torus is employed; each rack uses all 6 of its links. The description of the heuristic is vague in [1] and we make some reasonable assumptions to remove ambiguities. For each aggregator, we sort the sources, based on the amount of data they have to send to the corresponding aggregator. WLOG, we start by placing  $A_1$  at  $(0, 0, 0)$  and  $A_2$  at  $(x/2 - 1, x/2 - 1, x/2 - 1)$ , where  $x$  is the length of a side of the torus. We construct three rings around  $A_1$  along the three axes, using source racks from its sorted list. In the first iteration, we place the first six sources at  $(1, 0, 0)$ ,  $(0, 1, 0)$ ,  $(0, 0, 1)$ ,  $(x - 1, 0, 0)$ ,  $(0, x - 1, 0)$  and  $(x - 1, 0, 0)$  respectively. We continue placing the remaining sources along the 3 rings, moving away from  $A_1$  in corresponding iterations, until we fill the rings. We build three rings around  $A_2$  with sources from its sorted list in the similar way, ignoring sources which have already been placed around  $A_1$ . The remaining racks are placed at the remaining positions randomly. Routing is done using two minimum spanning trees; one for each aggregator. We use the Prim's algorithm to construct the Minimum Spanning Tree for each aggregator. When a link  $(u, v)$  is selected in the tree, we increase the weight of all the edges starting from  $v$  all the way up to the aggregator by the data from  $v$ .

In addition to  $S_1 = S_2 = \emptyset$ , our test data have  $m_1 = m_2 = 0$ . So, we programmed only the  $m_1 = m_2 = 0$  case for our algorithm. Although the construction of Section VI-A requires at most 3 links per rack of  $C$  in  $U_1$  and  $U_2$ , we assigned 3 links to each such rack in each of  $U_1$  and  $U_2$  and used the Depth First Decreasing (DFD) algorithm of [3] to organize the racks assigned to each subtree of  $U_1$  and  $U_2$  so as to reduce total network traffic while aggregating in optimal time.

The distribution of the amount of data, available at each source rack, is application specific. The distribution may vary widely depending on the particular application involved. For example, if, say, we are scanning a big text corpus distributed in the network and returning frequencies of a set of keywords for each document, we have the same amount of data to aggregate from each source rack. On the other hand, if we are executing a search query on the same text corpus and returning the documents matching the keywords, the amount of data to be sent to the aggregators from each source rack may vary to a large extent. So, in order to assess the performance of our algorithm in such real scenarios, we used the following data sets:

- *Gaussian 1.* The  $c_i^r$ s are drawn from a truncated Gaussian distribution with mean 500 and standard deviation 1000 truncated in [200, 800].
- *Gaussian 2.* The  $c_i^r$ s are drawn from a truncated Gaussian distribution with mean 500 and standard deviation 1000 truncated in [400, 600].
- *Uniform.* The  $c_i^r$ s are drawn from a uniform distribution with values in [50, 100].

The link bandwidth is assumed to be 1 unit so that the data transmission time equals the amount of data transmitted

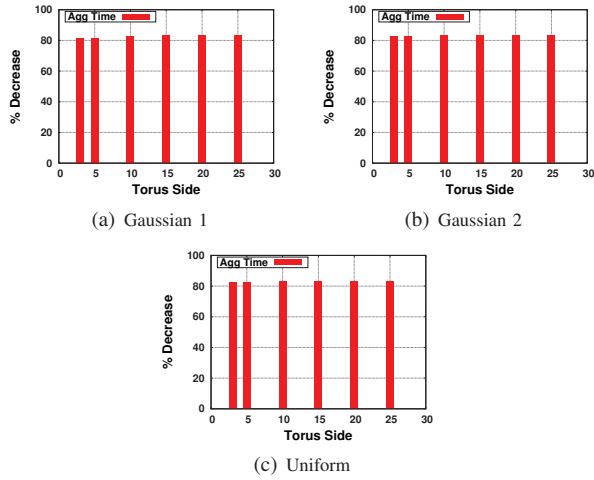


Fig. 3: Average percentage change in aggregation time

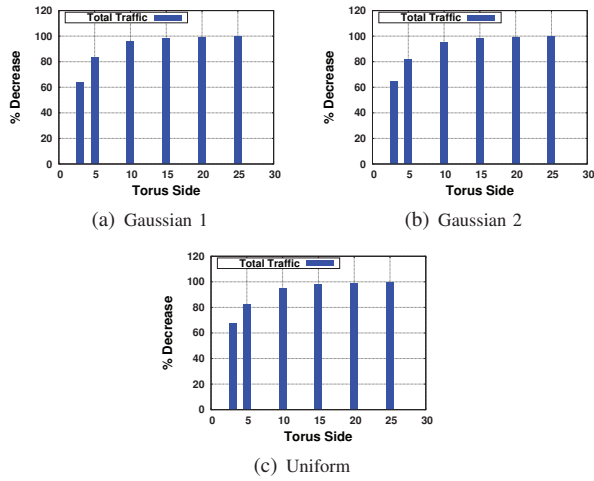


Fig. 4: Average percentage change in total network traffic

through a link. In our experiments, we varied the length of a side of the torus from 3 to 25 nodes; if a side of the torus has  $x$  nodes, the total number of racks in the topology is  $x^3$ , which includes the 2 aggregators. So our experiments involve up to  $25^3 - 2 = 15623$  source racks. For each choice of  $x$  and data distribution, we run the algorithms on 10 randomly generated instances. Our experiments were conducted on a 64-bit PC with a 2.80 GHz AMD Athlon(tm) II X2 B22 processor and 8GB RAM.

#### A. Aggregation Time

Figure 3 shows the average change in aggregation time when our  $k = 6$  algorithm is used versus the 3D torus extension of [1]. The average reduction in aggregation time (relative to the torus) for any  $n$  was up to 83.32%, 83.31% and 83.32% for the Gaussian 1, Gaussian 2 and the Uniform data-sets, respectively. Similar high percentages were reported for all network sizes with the minimum being 81.51%. We observe that the three distributions show quite similar percentage reductions. Although the distributions have significantly different PDF functions, they have a similar impact on data aggregation for  $m_1 = m_2 = 0$ .

#### B. Total Network Traffic

Figure 4 shows the average change in total network traffic using our  $k = 6$  algorithm versus using the 3D torus extension

of [1]. The average reduction in total network traffic (relative to the torus) for any  $n$  was up to 99.5%, 99.4% and 99.4% for the Gaussian 1, Gaussian 2 and the Uniform data sets, respectively with the minimum being 63.4%.

### VIII. CONCLUSION

In this paper, we have focused on the Two Aggregator Network Topology Optimization with Splitting (TANTOS) problem. We have proved that TANTOS is NP-hard for  $k = 2$ ,  $k = 3$  and  $k = 4$  using reductions from the standard 2-way Partition and a newly formulated 3-way Partition problems. Here  $k$  is the degree of a ToR switch. We have also shown that TANTOS can be solved in polynomial time when  $k = 5$  and 6 and conjecture this is so when  $k > 6$ . Through extensive experiments, we illustrate the improved performance obtainable using our optimal algorithm for  $k = 6$  versus a 3D torus extension of the 2D torus heuristic proposed by Wang et al. in [1] for  $k = 4$ . Using our algorithm, the data aggregation time and total network traffic reduce by up to 83.3% and 99.5%, respectively.

### ACKNOWLEDGMENT

This research was supported, in part, by the National Science Foundation under grant CNS0905308.

### REFERENCES

- [1] Wang, Guohui and Ng, T.S. Eugene and Shaikh, Anees, *Programming your network at run-time for big data applications*, Proceedings of the first workshop on Hot topics in software defined networks HotSDN '12, 2012.
- [2] Soham Das and Sartaj Sahni, *Network topology optimization for data aggregation*, IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid), 2014.
- [3] Soham Das and Sartaj Sahni, *Network topology optimization for data aggregation with Splitting*, IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), 2014.
- [4] R. L. Graham, *Bounds on Multiprocessing Timing Anomalies*, SIAM JOURNAL ON APPLIED MATHEMATICS, 1969, volume 17, number 2, pages 416–429.
- [5] Coffman, Jr., E. G. and Sethi, Ravi, *A generalized bound on LPT sequencing*, Proceedings of the 1976 ACM SIGMETRICS conference on Computer performance modeling measurement and evaluation, SIGMETRICS '76, 1976.
- [6] Greenberg, Albert and Lahiri, Parantap and Maltz, David A. and Patel, Parveen and Sengupta, Sudipta, *Towards a next generation data center architecture: scalability and commoditization*, Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow, PRESTO '08, 2008.
- [7] Al-Fares, Mohammad and Loukissas, Alexander and Vahdat, Amin, *A scalable, commodity data center network architecture*, Proceedings of the ACM SIGCOMM 2008 conference on Data communication, SIGCOMM '08, 2008.
- [8] Guo, Chuanxiong and Wu, Haitao and Tan, Kun and Shi, Lei and Zhang, Yongguang and Lu, Songwu, *Dcell: a scalable and fault-tolerant network structure for data centers*, Proceedings of the ACM SIGCOMM 2008 conference on Data communication, SIGCOMM '08, 2008.
- [9] Das, S. and Yiakoumis, Y. and Parulkar, G. and McKeown, N. and Singh, P. and Getachew, D. and Desai, P.D., *Application-aware aggregation and traffic engineering in a converged packet-circuit network*, Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference, 2011.
- [10] Webb, Kevin C. and Snoeren, Alex C. and Yocum, Kenneth, *Topology switching for data center networks*, Proceedings of the 11th USENIX conference on Hot topics in management of internet, cloud, and enterprise networks and services, Hot-ICE'11, 2011.
- [11] <http://www.cise.ufl.edu/~sahni/papers/2aggregatorWithSplit.pdf>