Abstract—In this paper, we study topology control methods aiming at achieving high throughput for wireless mesh networks. Based on the analysis of the existing topology control methods, such as the shortest path tree method, the load balancing method and the greedy method, we propose a Relative-Closest Connect-First topology control algorithm. In our proposed method, each node has a distance vector to keep the hop-distance to all gateway nodes, one component for each gateway. A node that has a closest relative distance to a gateway has the highest priority to be connected to the subtree rooted from the gateway. Our proposed topology control method combines the merits of the shortest path tree, the load balancing and the greedy methods. We also present a distributed implementation of the proposed topology control method. Simulation results have shown our proposed method achieves better performance than the existing methods under various distributions of nodes and gateways, and the distributed method achieves a similar performance to the centralized one. 1

I. INTRODUCTION

In this paper, we study the problem of constructing a network topology that achieves high throughput for Wireless mesh networks (WMNs) [1]. Supposing each router has a traffic demand, our task is to construct a forest (a set of trees rooted from the gateway nodes) and route the traffic from mesh nodes to the Internet via the gateway nodes along the forest, such that the per-node end-to-end throughput of the network is maximized. The traffic of each node is assumed to be unsplittable and routed to the Internet through a single path.

Some previous works, such as the shortest path tree (SPT) method, the load balancing method and the greedy method have already been proposed for constructing topologies for wireless mesh networks. However, the SPT method often produces unbalanced traffic load among gateway nodes, which results in poor network throughput. The load balancing method tries to balance the traffic load among gateway nodes. But, it does not minimize the max collision load in the system, which does not lead to a good performance in throughput. The greedy method, in each round, greedily chooses a node and a link to be added into the topology, such that the max collision load in the system is minimized. Again, this method may be trapped in some local optimum, because the network topology formed at the early stage may cause bad effect to the links to be added later.

In this paper, we present a topology control method, called Relative-Closest Connect-First (RCF) topology control. In each round of the method, each gateway selects a candidate (node) that has the closest relative distance to this gateway. Among all the candidates selected by gateways in that round, the one with minimal max collision load will be added to the topology. The method combines the merits of the SPT, the load balancing and the greedy methods, which achieves good performance in various mesh node distributions. Besides the centralized algorithm, we also propose a distributed implementation, which is efficient and achieves the similar throughput as the centralized method according to simulation results.

The rest of this paper is organized as follows. We discuss the related work in Section II. In section III, we describe the system model and formulate the problem we concern. The Relative-Closest Connect-First topology control is presented in Section IV. We investigate the performance of our solution through simulations and the results are analyzed in Section VI. Finally, Section VII concludes this paper.

II. RELATED WORK

Some researchers believe that the interference level is an indication of the system throughput, and thus proposed topology control algorithms to minimize the maximal or average interference level [2], [8], [5], [9]. However, since the traffic is not averagely distributed in the wireless mesh network, minimizing the interference level does not necessarily lead to a high throughput topology, as shown in [7].

There are some topology control methods proposed for wireless mesh networks, or applicable to wireless mesh networks. The work in [3] constructed a load balancing forest for wireless networks, in which each tree carries almost the same traffic load. In [4], the authors defined the nominal capacity of wireless mesh networks as one over the maximal collision domain load. The collision domain load for a link in their concern is the total load of its interfering links. Two similar greedy methods are proposed in [6] and [7] to construct a topology with maximal collision domain load. They are shown to achieve higher nominal capacity, compared with methods that use distance or interference as the performance metric.
The mesh network discussed in this paper consists of a set of mesh routers, some of which are gateway routers that have wire connection to the Internet. We simply call gateway routers gateways and non-gateway routers nodes. The physical network topology is represented as $G(V,E)$, where $V$ is a set of mesh routers (including gateways). A physical link exists between $u$ and $v$, i.e., $(u,v) \in E$, if and only if $u$ and $v$ are in each other’s transmission range. The communication link is bi-directional. All traffic of a node is destined to or originating from the Internet. We assume all nodes have the same traffic demand $\lambda$. Let $\alpha$ denote the traffic delivery ratio per node, $0 \leq \alpha \leq 1$. That is, for each node, only $\alpha \lambda$ of end-user’s traffic is guaranteed to be delivered to Internet due to the limited bandwidth of the system. We consider a common traffic delivery ratio for all nodes, because maximizing the throughput is equivalent to maximizing the total traffic delivery ratio, as shown in [10]. All nodes use the same channel, and the antennas on routers are omni-directional.

Our goal is to construct a logical topology that allows each node to route their end-users’ traffic to the Internet via the gateways, such that the amount of traffic delivered per node is maximized. Since we only consider a single path routing from a node to a gateway, the routes of all nodes to gateways form a forest and each tree in the forest is rooted from a gateway. The end-users’ traffic is merged at the tree nodes who further forward the traffic to their parent nodes towards the gateway. Let $T(v_i)$ denote the subtree rooted at node $v_i$. And let $l_{v_i}$ denote the link connecting subtree $T(v_i)$ to its parent node, and $L_{v_i}$ the traffic on link $l_{v_i}$, as shown in Fig. 1. Thus

$$L_{v_i} = \sum_{v_j \in T(v_i)} \alpha \lambda = |T(v_i)| \alpha \lambda,$$

where $|T(v_i)|$ is the size of $T(v_i)$, which is the number of nodes in $T(v_i)$, including $v_i$ itself.

We adopt the protocol interference model in this paper. Suppose the transmission range is $r_t$. Two nodes interfere with each other when their distance is within the interference range $q r_t$, where $q$ is a constant usually between 2 and 4. Since we assume communication links are bi-directional, two links $l_1$ and $l_2$ interfere with each other if and only if one end-node of $l_1$ interferes with one end-node of $l_2$. For link $l_{v_i}$, we define its collision set as a set of links that interfere with it, including $l_{v_i}$ itself, and denote this collision set as $I(l_{v_i})$. We define the (normalized) collision load of $l_{v_i}$ as the total carried traffic demand of links in the collision set of $l_{v_i}$, which is

$$L_I(l_{v_i}) = \sum_{l_{v_j} \in I(l_{v_i})} |T(v_j)| \alpha \lambda.$$

In wireless communication, two links that interfere with each other cannot be active at the same time due to the signal interference. We consider the most conservative case that no two links in the same collision set can be active at the same time in order to guarantee successful transmissions. That is, the collision load of any link, say $l_{v_i}$, cannot exceed the bandwidth of a channel. Thus, we have:

$$\sum_{l_{v_j} \in I(l_{v_i})} |T(v_j)| \alpha \lambda \leq C,$$

where $C$ is the channel bandwidth, and its unit is $\text{bps}$.

From ineq. (3), we have

$$\alpha \leq \frac{C}{\sum_{l_{v_j} \in I(l_{v_i})} |T(v_j)| \alpha \lambda} = \frac{C}{L_I(l_{v_i})}.$$

The nominal capacity of our concern is the maximal possible throughput. It is the maximal possible value of $\alpha$ that meets ineq. (4) for all node $v_i$. That is,

$$\text{Cap} = \max_{v_i} \alpha = \min_{v_i} \frac{C}{L_I(l_{v_i})} = \frac{C}{\max_{v_i} L_I(l_{v_i})}.$$

Let $L_{\text{max}} = \max_{v_i} L_I(l_{v_i})$, which is the max collision load of the system. The capacity can be re-written as

$$\text{Cap} = \frac{C}{L_{\text{max}}}.$$

From the above definition of the nominal capacity in eq. (6), we can see that maximizing the throughput is equivalent to minimizing the max collision load $L_{\text{max}}$ of the system.

### IV. ALGORITHM DESIGN

In this section, we first analyze the drawbacks of the previous topology control methods for wireless mesh networks. Then we present our Relative-Closest Connect-First topology control method based on our analysis.

#### A. Drawbacks of previous topology control methods

In this subsection, we analyze some of the typical topology control methods for wireless mesh networks, including the shortest path tree method, the load balancing method, and the greedy method.

The shortest path tree method constructs a forest such that each node is connected to its closest gateway by shortest path (in terms of hops). When there are many nodes around one gateway and few nodes around others, this method may lead to a serious load unbalancing, which results in a high max collision load.

The load balancing method tries to assign the same number of nodes for trees rooted at each gateway. However, because links have different traffic load, the balancing of traffic load does not mean the balancing of collision load.
Let $h$ be a wireless mesh network with multiple gateways as follows.

### C. Centralized method

We will start with a centralized method first, and thus present our Relative-Closest Connect-First topology control method.

#### B. Relative-Closest Connect-First topology control

As shown in last subsection, the topology with the minimal max collision load does not necessarily connect nodes to their closest gateways, nor connect balanced number of nodes to each gateway. In order to achieve maximal throughput, an algorithm should aim at minimizing the max collision load, rather than at any other metric.

The greedy method starts with a forest that initially consists of only gateways. Each time a link that minimizes the current max collision load is added into the forest. This operation is repeated until all nodes are eventually added into the forest. When the gateways are not uniformly placed in the network, the greedy method could lead to a rather bad performance, as illustrated in Fig. 2. Due to its top-down fashion, the nodes that are enclosed in each dashed circle may be connected to that gateway after several rounds. Next, all the rest nodes on the right hand must be connected to gateway $G_2$. In this case, there will be too many nodes connected to $G_2$, which reduces the network throughput.

Fig. 2. Bad performance of the greedy method.

**B. Relative-Closest Connect-First topology control**

As shown in last subsection, the topology with the minimal max collision load does not necessarily connect nodes to their closest gateways, nor connect balanced number of nodes to each gateway. In order to achieve maximal throughput, an algorithm should aim at minimizing the max collision load, rather than at any other metric.

The greedy method aims at minimizing the max collision load while constructing the topology. However, from the example of the greedy method shown in Fig. 2, we can see the performance is greatly affected by the order that nodes are connected to the forest. In Fig. 2, each time only the neighboring nodes of the forest are considered to be added into the forest, and two subtrees growing up from the two gateways will equally divide the nodes located in between the two gateways and connect them without seeing the group of extra nodes located at the far right-hand side of $G_2$. This results in a topology with two unbalanced subtrees. If we give the nodes at the right-hand side of $G_2$ a higher priority than the nodes at the left-hand side of $G_2$ to be connected to $G_2$, it would leave more nodes located in between $G_1$ and $G_2$ connecting to $G_1$, which makes the two subtrees more balanced. Notice that for nodes at the right-hand side of $G_2$, their distance to $G_2$, relative to the distance to $G_1$, is shorter than the nodes at the left-hand side of $G_2$. They should have a higher priority to be connected to the subtree of $G_2$ than the nodes at the left-hand side of $G_2$. This observation leads to our Relative-Closest Connect-First topology control method.

We will start with a centralized method first, and thus present a distributed version, based on the similar technique with the centralized one.

**C. Centralized method**

We define the relative distance from a node to a gateway in a wireless mesh network with multiple gateways as follows. Let $h_{u,v}$ be the number of hops between node $u$ and $v$, $u \in V$, $v \in V$, in $G(V,E)$. Suppose there are $M$ gateways in the system, denoted by $W = \{g_1, g_2, \ldots, g_M\}$. For each node $u$, we define a vector of hop-numbers between $u$ and all the gateways, denoted as $H_u$:

$$H_u = \langle h_{u,g_1}, h_{u,g_2}, \ldots, h_{u,g_M} \rangle. \quad (7)$$

For each gateway $g_i \in W$, the vector of hop-counts to all gateways is denoted as $H_{g_i}$:

$$H_{g_i} = \langle h_{g_i,g_1}, h_{g_i,g_2}, \ldots, h_{g_i,g_M} \rangle, \quad (8)$$

where $h_{g_i,g_i} = 0$.

We define the relative distance between a node $u$ and a gateway $g_i$ as the distance between $H_u$ and $H_{g_i}$, denoted by $d(u, g_i)$:

$$d(u, g_i) = \sum_{g_j \in W} (h_{u,g_j} - h_{g_j,g_i})^2. \quad (9)$$

Based on the above definition, a smaller $d(u, g_i)$ indicates that the distance vector from $u$ to all gateways has a better match of the distance vector from $g_i$ to all other gateways, i.e., $u$ is relatively closer to $g_i$ than to other gateways. It means $u$ is more urgent, compared with other nodes whose relative distance to $g_i$ is longer than $u$, to be connected to the subtree of $g_i$. By connecting $u$ to the subtree of $g_i$ first, it allows the nodes who are also located nearby $g_i$ (in terms of Euclidean distance) but relatively closer to other gateways (e.g., the nodes located at the left-hand side of $G_2$ in Fig. 2) to connect to the other gateways for load balancing purpose at the later stage of the topology construction. If connecting these nodes to $g_i$ first as the SPT method does, later it would either force node $u$ to take much more hops to connect to other gateways (for load balancing), or simply connect $u$ to $g_i$ but causing load unbalancing.

The topology construction method works as follows. Initially, the forest consists of all the gateway nodes as roots of subtrees. In each round, each gateway $g_i$ finds the node $u$ whose $d(u, g_i)$ is the smallest among all the nodes, and computes the minimal max collision load if connecting node $u$ to the subtree of $g_i$. After all gateways find their most suitable candidates to be connected into their subtrees, the candidate that results in the smallest minimal max collision load will be selected and connected to the corresponding subtree. The process is repeated until all nodes are eventually connected to the subtrees (of the forest). The details of our method are shown in Fig. 3.

Our topology control method combines the idea of the greedy, the SPT, and the load balancing methods:

1) our method aims at minimizing the max collision load when connecting each node into the forest, which is a direct optimization of the system throughput;

2) the node with the closest relative distance to the gateway has the highest priority to be connected to that gateway, which makes the routing more efficient;

3) our method can produce more balanced subtrees than the traditional SPT method, because we connect nodes to the gateways that are relatively closest to them first, leaving nodes
that have the similar distances to multiple gateways to be connected later, which gives more flexibility to allocate these nodes to subtrees for load balancing purpose.

V. DISTRIBUTED IMPLEMENTATION

In this section, we present a distributed implementation of the RCF topology control method proposed in Section IV-C. The distributed algorithm consists of two phases: the first phase is Information Collection, and the second phase is Topology Construction.

During the first phase, each gateway broadcasts a topology-query packet to the entire network. At the completion of this phase, each node \( u \) will receive at least one topology-query packet from each gateway, and based on the number of hops recorded in these packets, it knows the number of hops to all gateways, i.e., \( H_u \). And each gateway \( g_i \) also knows the number of hops to all gateways, i.e., \( H_{g_i} \). Through message exchanges with 1-hop neighboring nodes, we assume each node knows its 1-hop neighbors and their distance vectors to gateways.

In the topology construction phase, each gateway chooses a node from the neighbors of its subtree that has the shortest relative distance, and finds the max collision load if connecting this node to this subtree. All the gateways then, coordinating with each other, decide to connect which node that will produce the minimal amount of max collision load to the topology constructed so far. The details of the method are as follows.

Each gateway \( g_i \) keeps a list of its subtree nodes, denoted by \( T_{g_i} \), and a set of neighboring nodes of the subtree, denoted by \( N_{T_{g_i}} \). Each node \( v \) in \( N_{T_{g_i}} \) contains the ID of \( v \) and its distance vector to gateways, i.e., \( H_v \). Initially, \( T_{g_i} \) contains only \( g_i \) and \( N_{T_{g_i}} \) contains the 1-hop neighbors of \( g_i \). Gateway \( g_i \) chooses a node from \( N_{T_{g_i}} \), say \( u \), that has the smallest \( d(u, g_i) \) defined in eq. (15), and sends an invitation packet \( Inv \) to \( u \) through the shortest path along the subtree constructed so far. Here, we decide the parent node of \( u \) if \( u \) is connected to the tree (through the shortest path from \( g_i \) to \( u \) along the tree), which is different from the centralized algorithm. This simplification comes from our observation of the simulation results that in almost all the cases a node is connected to the tree via the shortest path to the gateway (of the tree).

Upon receiving the \( Inv \) packet, node \( u \) sends a join packet \( Join \) back to the gateway of this subtree, hop by hop. The \( Join \) packet contains the ID of \( u \) and its traffic demand \( \lambda_u \). In order to collect the new collision load information by adding \( u \) into the tree, the \( Join \) packet is transmitted by using a larger transmission power such that all nodes in the interference range of any node along the tree-path from \( u \) to \( g_i \) can hear the \( Join \) packet. When a node in the tree-path from \( u \) to \( g_i \) receives the \( Join \) packet, it computes the collision load of the link to its parent by adding the new traffic \( \lambda_u \) and continues to forward the packet up towards the root node \( g_i \) of the tree. When a node, say \( v \), in the interference range of the tree-path over hears this \( Join \) packet, it will also compute the new collision load of the link linking to its parent, i.e., \( L_j(\nu_c) \) defined in eq. (2), if it is already in the forest constructed so far. Since \( v \) may be in the interference range of several nodes in the tree-path as the \( Join \) packet is forwarded up, to avoid the transmission of its collision load multiple times, we let \( v \) wait for a certain period of time. When this timeout expires, \( v \) sends a \( Coll_load \) packet that contains its most up-to-date collision load to the node from which it hears the last \( Join \) packet. Upon receiving the \( Coll_load \) packet, the node in the tree-path aggregates the collision load in the packet together with the collision load it received from other nodes and of its own, and forwards the largest one up towards the gateway \( g_i \) along the tree-path.

When gateway \( g_i \) receives the \( Join \) packet from \( u \), it waits for a period of time to collect the \( Coll_load \) packets from its subtree. After the timeout, \( g_i \) knows the max collision load of its subtree by connecting \( u \) to it. Gateway \( g_i \) then exchanges the max collision load information with other gateways to find out which gateway and node that produce the minimal max collision load. Suppose gateway node \( g^*_i \) and node \( u^* \) are such a pair that produces the minimal max collision load.

At the next step, \( g^*_i \) sends an \( Acpt \) packet to \( u^* \). Upon receiving the \( Acpt \) packet, \( u^* \) propagates a confirm packet \( Conf \) along the tree-path back to \( g^*_i \) hop by hop. The \( Conf \) packet contains the non-tree neighbors of \( u^* \) and their distance vectors to gateways. Same as the \( Join \) packet, the \( Conf \) packet uses the transmission power large enough to reach all nodes in the interference range. When a node receives (or over hears) the \( Conf \) packet, it updates the collision load of the link linking to its parent by adding the traffic demand of \( u^* \). When \( g^*_i \) receives the \( Conf \) packet from \( u^* \), it updates the tree-node list \( T_{g^*_i} \) and tree-neighbor list \( N_{T_{g^*_i}} \), including \( u^* \) and \( u^* \)'s non-tree neighbors, respectively. The above operation is repeated until all nodes are added into the set of subtrees rooted from gateways.

\[
T_{g_i} = \{g_i\} \text{ for } g_i \in W \\
N_{T_{g_i}} = \{\text{ neighbors of } g_i \} \text{ for } g_i \in W \\
N = \{\text{all non-gateway nodes}\}
\]

while \( N \neq \emptyset \) do
  for all \( g_i \in W \) do
    find \( u_i \in N_{T_{g_i}} \) with \( \min d(u_i, g_i) \)
    find \( p_i \in T_{g_i} \) with \( \min L_{\max} \) if connecting \( u_i \) to \( p_i \)
  end for
  connect \( u_i^* \) to \( p_i^* \)
  if \((u_i^*, p_i^*)\) achieves \( \min L_{\max} \) if connecting \( u_i^* \) to \( p_i^* \)
  \( T_{g_i} = T_{g_i} + \{u_i^*\} \)
  \( N_{T_{g_i}} = N_{T_{g_i}} - \{u_i^*\} + \{\text{non-tree neighbors of } u_i^* \} \)
  \( N = N - \{u_i^*\} \)
end while

output \( \{T_{g_i}, g_i \in W\} \) as the network topology.

Fig. 3. Centralized Relative-Closest Connect-First topology control
VI. PERFORMANCE EVALUATION

The simulations are conducted in a 300m × 100m region, with 100 non-gateway nodes. The channel capacity is set to 54 MB/s and the traffic demand of each router is 1 MB/s. All gateways and nodes have the same transmission range and the ratio of interference range to transmission range, \( q \), is 2.

We compare the performance of our method, namely RCF and its distributed version, with the SPT, the load balancing, and the greedy methods. Each simulation runs 50 times and their average is taken as the result.

![Topology](image)

(a) Topology

![Nominal capacity](image)

(b) Nominal capacity

Fig. 4. Three-gateways network.

The existing methods, the SPT, the load balancing and the greedy would perform satisfactorily when node distribution is uniform and the gateway placement is random. Our method addresses the weakness of the existing methods in some special circumstances. We first consider the scenario that nodes are unevenly distributed but gateways are evenly placed. Then, we consider the scenario that nodes are uniformly distributed but gateway placement is unbalanced. They are the common scenarios that the existing methods usually perform badly. Fig. 4(a) shows an example of the first scenario. The whole area is evenly divided into four subarea A, B, C and D. Three gateway G1, G2, and G3 are placed along the top boundary of the area, where G1 and G3 are at the two sides and G2 at the middle. The gateways are placed at the boundary to ensure there exist some long paths, that is, paths of hop-numbers more than 2, to ensure that there are some nodes outside the interference area of some gateways. The optimal topology control strategy is not trivial in this scenario, as how to connect those nodes that at least 3-hops away from any gateway can significantly affect the throughput. The number of nodes in each subarea decreases from left to right, which is 40, 30, 20, 10 nodes for subarea A, B, C and D respectively, and nodes are placed randomly in each subarea. In such a node distribution, the SPT will create an unbalanced forest, because more nodes are closer to G2 than G1 and G3. The load balancing method tries to assign the same number of nodes to each of the three trees of the gateways. Since there are more nodes close to gateway G1, the average end-to-end hops of nodes in the tree of G1 is smaller than that of G2 and G3. The average load of links in the tree of G1 is smaller than that of G2 and G3. The load balancing method will construct a forest of unbalanced collision load, which results in a poor throughput. Since both the greedy method and our RCF aim at minimizing the max collision load, we expect that they perform better than SPT and the load balancing method in this scenario.

As shown in Fig. 4(b), the RCF and its distributed version outperform the other algorithms. RCF’s distributed version has a slightly lower performance than the RCF. This is because the distributed version skips the selection of a best tree-node that a node is connected to and it simply links the node to a tree-node that has the minimal number of hops to the gateway. Thus, the distributed version does not optimize the max collision load in choosing a parent node for a candidate. Nevertheless, it still performs better than the other methods in comparison, because it adds nodes to the forest in an order according to the relative-distance, which is a more important factor that affects the performance. From the performance curves, we observe that the load balancing method performs the worst among all methods in most of the cases. This confirms our observation that the balancing of tree nodes does not necessarily balance the collision load. We observe that the RCF method outperforms the greedy method. This is because the greedy method connects nodes to the trees in the order according to their Euclidean distance to the gateways. So, it may connect all the nodes near the boundary between subareas C and D to the subtree of G2, which leaves no chance for G3 to connect nodes in subarea F. This results in unbalanced trees. But in the RCF method, G2 will connect nodes in the region below it before the nodes at its right or left sides, because the nodes below G2 are relatively closer to G2 compared with the nodes at its sides. Therefore, it leaves more nodes near the boundary between C and D to be connected to G3, which results in a more balanced forest.

From Fig. 4(b), we observe that the capacity generally increases as the increase of transmission range. This is because the increase of transmission range would reduce the average end-to-end hops, and thus reduce the relay traffic. Therefore, the link load, particularly for the links close to the gateways, is reduced. When transmission range is beyond a certain threshold, the capacity may decrease as the further increase of transmission range. This is because the links that interfere with multiple gateways become the links of the max collision load in the system. The collision load of those links includes load destined to several gateways, which is quite large. Such an increase in collision load out weights the benefit brought by the reduction of number of hops to gateways. When transmission range keeps on increasing, the capacity eventually increases, since the relay load will keep on decreasing, and no more gateways are included in the collision set. Notice that, since load balancing method does not aiming at the capacity directly, its curve does not necessarily follow the relationship between capacity and transmission range as we mentioned in this paragraph.
Now we consider the second scenario that nodes are uniformly distributed but gateways are unevenly placed. As shown in Fig. 5(a), the whole area is evenly divided into three subareas, $A$, $B$, and $C$. Two gateways are placed at the top boundary, where gateway $G_1$ is placed at the left corner and gateway $G_2$ is placed at the center of subareas $B$. 60 nodes are uniformly distributed in subarea $B$, and all the other 40 nodes are uniformly placed in the subareas $A$ and $C$. This scenario is similar to the case we discussed in section IV-A.

As shown in 5(b), the RCF performs better than other methods. Its distributed version achieves a similar performance. This is because, in this scenario, according to the RCF method, most of nodes are connected to the gateways via shortest paths. Therefore, in most cases, the RCF and its distributed version make same decision to select the tree-node that a node is connected to. Due to the local greedy decision, the greedy method even does not perform as good as the SPT. This illustrates that the pure greedy is not a robust method. The greedy method may let $G_2$ connect to some nodes in subarea $A$, which prevents $G_1$ from connecting nodes in subareas $B$ and $C$, even part of nodes in subarea $A$, to gateway $G_1$. In contrast, the SPT connects all nodes in subarea $A$ to $G_1$. This is why the greedy method connects too many nodes to $G_2$, and results in an unbalanced structure that is even worse than the SPT.

Finally we simulate the case that both nodes and gateways are uniformly distributed. We place 100 non-gateway nodes and $K$ gateways uniformly in a $300m \times 100m$ region., where $K$ varies from 3 to 9. The transmission range is 30 meters. We observe in Fig. 6 that the capacity increases as the number of gateways, because both the number of nodes assigned to trees rooted at each gateway and relay traffic decrease as the increase of number of gateways. As shown in Fig. 6, the RCF and its distributed version perform better than other methods. However, its performance gain is marginal. This confirms our intuition that, when nodes and gateways are evenly distributed in a mesh network, the performance of the SPT, the greedy method, and the RCF would be very close to each other. This is because all these three methods in this scenario would connect nodes to their closest gateways and assign same number of nodes per gateway.

![Fig. 5. Two-gateways network.](image)

![Fig. 6. Nominal capacity of $K$-gateways network.](image)

VII. CONCLUSION

In this paper, we investigated topology control methods that construct network topologies to achieve high throughput for wireless mesh networks. We first studied the drawbacks of the previous works, such as the shortest path tree method, the load balancing method and the greedy method. Then we proposed our method, Relative-Closest Connect-First (RCF) method, that combines the idea of of the SPT, the load balancing and the greedy. We also proposed a distributed implementation of the RCF method. The simulation results have shown that our proposed RCF method achieves higher throughput than the existing methods in various scenarios of node distributions and gateway locations, and the distribution implementation achieves a similar performance as the centralized one.

REFERENCES