SWARM: The Power of Structure in Community Wireless Mesh Networks

Saumitra Das
Qualcomm Research Center
saumitra@qualcomm.com

Konstantina Papagiannaki
Intel Labs Pittsburgh
dina.papagiannaki@intel.com

Suman Banerjee[*]
University of Wisconsin at Madison
suman@cs.wisc.edu

Y.C. Tay
National University of Singapore
dcstayyc@nus.edu.sg

ABSTRACT

Community wireless networks (CWNs) have been proposed to spread broadband network access to underprivileged, under-provisioned and remote areas. Research has focused on optimizing network performance through intelligent routing and scheduling, borrowing solutions from mesh networks. Surprisingly, however, there has been no work on how to make efficient use of multiple channels in CWNs in the presence of multiple gateways, and a single radio per device. In fact, today's deployments in under-privileged areas are primarily single radio and do operate on a single channel [19]. Frequency selection in such CWNs is very complex because it does not only determine the nodes' channel of operation but also the gateway and the routing tree to the gateway - a rather computationally intensive task. In this paper, we propose, design, implement, and evaluate SWARM, a practical system that allows a CWN to make effective use of the available wireless channels in order to offer globally optimal performance. SWARM improves performance versus current single channel protocols by up to 7.7× in our experiments. Moreover, while we should be expecting performance gains due to channel diversity, we clearly demonstrate that up to 3.7 × improvement is attributed to the network organization into efficient traffic distribution structures.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network Topology; Wireless Communication

[*]Suman Banerjee was supported in part by NSF awards CNS-0639434, CNS-0627589, CNS-0627102, CNS-0520152, and CNS-0747177.

General Terms

Algorithms, Design, Management, Performance

Keywords

community wireless networks, self-organization

1. INTRODUCTION

Community wireless networks (CWNs) consist of a few Internet gateways providing access to homes using multi-hop wireless links between off-the-shelf wireless mesh routers (MRs) already deployed within homes. Such networks have low cost for deployment due to reduced wiring, and low maintenance.

While CWNs fall under the general category of wireless mesh networks, they have certain differentiating properties that create unique challenges, as well as opportunities. First, devices in CWNs tend to be single-radio devices of limited computational resources[1]. For the rest of the paper, we focus on such *single radio CWNs*. Second, CWN users tend to use the network to access the Internet. As a result, communication patterns across the network are no longer "any-to-any" but primarily "any-to-1" and "1-to-any", with the gateway node playing a fundamental role in the CWN.

Despite those differences, current CWN deployments [19] borrow solutions from ad hoc wireless networks. As a result, we often find CWNs utilizing a *single radio channel* across the entire network, even if they feature *multiple gateway nodes*. Measurements we performed on real CWN deployments by Netequality [19] were found to follow this single channel model. Even a network with 77 nodes and 5 gateways was found to use a single channel. Second, CWNs make use of mesh routing algorithms, such as OLSR [5], LQSR [9] and SrcRR [4]. The fundamental problem with such an approach is that these protocols necessitate the operation of all nodes in the same frequency and that the algorithm itself has a myopic view of interference across the network, even if ETT (Estimate Transmission Time) [9] or ETX (Estimate Transmission Count) [6] is the metric of choice; each node connects to the gateway with the shortest path metric

[1]This is because CWNs use current off-the-shelf 802.11 wireless routers that operate using a single radio interface, a design that also assists in keeping costs low.

ignoring the interference caused by the traffic of other nodes in the same tree. A direct consequence of the above choices, is that CWNs suffer from unnecessary interference, something which creates an obvious adoption barrier for users.

In this paper we look at mitigating interference across CWNs through their effective organization into efficient distribution structures. The configuration choices in a multi-gateway CWN can be summarized as i) the gateways need to identify their channel of operation, ii) each non-gateway node needs to select its channel of operation, the gateway it will route its traffic through, and the optimal path to that gateway. Upon those decisions, the network is organized in a number of distribution trees, each one rooted at each gateway. The selection of a channel by any node has a definite impact on the resulting routing trees and the overall network performance. The focus of this paper is to design algorithms that create and maintain a good set of trees (equal to the number of gateways), their shape, and their operating frequencies.

Clearly, the optimal solution to the above problem can be achieved through a joint optimization framework, where the clustering of nodes to trees, the organization of each tree, and the mapping of channels to trees, are all considered together. However, the multiplicity of possibilities across all these decisions involves significant computational complexity - exploring a subset of the design space on a small 12 node network took us more than two weeks in simulation! Hence, we focus on a solution that trades *some loss in optimality for practicality*. SWARM decomposes the overall problem into the following independent sub-problems: it (i) groups all nodes into difference clusters, one to each gateway, assuming that each cluster can operate independently, (ii) organizes these clusters into trees rooted at the gateways, and (iii) ensures near independent operation across trees using a channel hopping algorithm that leverages channel diversity.

Throughout our work we make the following interesting observations. First, the inherently centralized nature of CWNs (the network stops to function when the gateway node fails) allows for an efficient centralized solution with limited overhead. This choice is corroborated by current industry practices, e.g. Meraki [1], that tend to configure and manage CWNs from central locations living in the cloud. Second, the number of optimal configurations that a CWN can assume is only 1% of the possible configurations, emphasizing the importance of good algorithms for such a task. Third, the benefits one reaps from SWARM are not only limited to the gain arising from the use of multiple channels. We show that simple ways of introducing channel diversity are far inferior to SWARM. A principled approach to CWN organization can lead to 7 times improvements, and 3 times improvement can be attributed to the discovery of efficient distribution trees. Fourth, we show that the resulting performance benefits can be even higher for TCP that is severely affected by loss.

Our contribution in this paper is the definition, implementation and performance evaluation of SWARM, a *practical* system for the organization of CWNs into a number of frequency diverse distribution trees. Through a variety of tested scenarios SWARM is shown to identify the optimal organization of a 20 node CWN in less than 100 seconds[2] on a standard desktop machine (the workload is easily parallelizable and this time could be further reduced). In contrast, we find that the time between failures and substantial network condition changes in operational CWNs can be on the order of hours. SWARM employs mechanisms that allow it to trigger the re-organization of the network under significant changes, reacting quickly and accurately to network dynamics, while balancing overhead and goodness of the assumed network state.

[2]Note that 100 seconds correspond to the time that the network will operate in a sub-optimal state. Packets will still be delivered to the gateway if a routing path exists.

2. PROBLEM STATEMENT AND APPROACH

We first provide a high level overview of what SWARM *aims* to achieve and how it can be achieved. The basic problem statement we have is: Given a community wireless network with N participating mesh routers and K gateways, how should the nodes organize for optimal performance. Given that such networks are primarily used for Internet access, each gateway forms a distribution tree connecting a number of mesh routers to the Internet, an architecture widely used in mesh deployments today [19]. As a result our problem is defined as *how should the nodes organize themselves in efficient distribution trees*[3]*, rooted at each gateway, so as to maximize a fair notion of total throughput across the entire network*. We further develop such an optimization assuming that the network is primarily used for content downloads, today's prevalent workload.

If we leave *practical use* aside, then a brute force algorithm to achieve the goals of SWARM would need to do the following. For every assignment of nodes to gateways, it would need to form all possible trees, and quantify the performance of the overall network as a function of the performance of each individual tree. Then, it would return that configuration that optimizes the metric of choice. The inputs to such a process would be the network topology (N nodes and K gateways), the connectivity graph $G(V, E)$ with edges annotated with some link quality metric (e.g. ETT) and a conflict graph $C(V', E')$ with nodes denoting edges in $G(V, E)$[4]. The output, which we call a "configuration" in the rest of the paper, would specify which nodes should join which gateways, the routing paths for each node to the gateway and its channel.

Clearly brute force SWARM can do what we aim for: an automated self-organization for any arbitrary community mesh network topology. However, such an evaluation of all possible topologies is impractical and not scalable. First, we need to evaluate all possible node to gateway assignments, an operation with $O(K^N)$ complexity. Second, for each specific assignment of nodes to gateways we need to define all trees that can be constructed with that assignment for *each* gateway, an operation of $O(N^{N-2})$ complexity for *each* of the K trees in a configuration. Finally we need to model each tree analytically so as to quantify its performance. Then, the overall network performance is a function of the performance of the trees defined for each configuration. There is no existing work that models a single tree in a mesh network taking into account spatial reuse in a scalable manner. Most existing formulations that can

[3]We choose trees since they are a likely topology when each node maintains a next hop for its best gateway. Consideration of multi-path routing is left for future work.

[4]Two vertices in the conflict graph are connected via an edge if the links interfere.

be adapted to perform this modeling take a heavyweight linear programming approach. Modeling typical and realistic single path routing [12], instead of multi-path, requires even more intensive integer linear programs.

Thus the main design challenge for SWARM is to perform the above search for the "optimal" organization in a *practical* manner, i.e. a set of algorithms that may need to sacrifice absolute optimality to be practically deployable (in typical mesh routers) while remaining effective. SWARM comprises solutions for (1) node to gateway clustering, (2) interference aware high performance tree construction, (3) performance modeling of trees in a configuration, and (4) channel assignment for channel diversity.

Before we describe the design of SWARM, we first demonstrate the potential of using SWARM, i.e. automated self-organization in CWNs. Such an assessment is based on a 2-week long simulation experiment, throughout which we performed a near-exhaustive search of the configuration space.

A Motivating Example.

We use a randomly generated network topology with 12 nodes and 2 gateways in a 100m×100m area. Using QualNet[5] [21] we simulate a nearly exhaustive set of all possible network configurations. All possible assignments of nodes to gateways lead to 4095 configurations. Out of those configurations, 1248 lead to disconnected nodes and are thus ignored. For the rest, we use Dijkstra's algorithm on Estimated Transmission Time (ETT) to define the distribution tree sourced at each one of the gateways (a full enumeration is practically impossible - thus we call this exercise "nearly-exhaustive"). We then measure the throughput achieved by each client and compute the Aggregate Client Throughput (ACT), as the sum of download throughputs across all 12 clients in Kbps, and the Potential Delay (PD) [17], as the sum of the reciprocal of the achieved throughputs. Desirable configurations should lead to high values for ACT but low values for PD to minimize the level of unfairness. Given the formulation of PD any configuration that leads to a small set of nodes with high throughputs and a large set of nodes with small throughputs will lead to higher PD values than other more fair allocations. Figure 1 shows the ACT and PD value obtained for each one of the 2847 remaining configurations for TCP and UDP downloads.

Three important observations from this evaluation are: (1) A good configuration that organizes the nodes effectively can significantly improve performance even in such a small network size. The best configuration in our network provides a factor of 3 improvement in ACT for both TCP and UDP scenarios than the worst performing configurations. TCP performs comparatively worse than UDP due to its known problems with increased loss. (2) The best configurations are a small fraction (about 1%) of the possible configurations that the network might assume, based on the algorithm used for self-organization and routing. This provides strong motivation for the design of intelligent algorithms that result in one of the few good configurations that can effectively make use of the network resources. (3) When studying the results across both ACT and PD, the number of configurations that simultaneously maximize ACT and minimize PD are even smaller, with a good configuration being able to reduce PD by upto 5× for TCP and more than 100× times for UDP[6]! The great improvement in PD clearly demonstrates that particular configurations can lead to gross unfairness unless care is taken. This further motivates the need for intelligence in self-organizing the network to arrive at the elusive good configurations.

[5]QualNet is a widely used commercial network simulator which models in detail wireless propagation phenomena, interference, fading and the 802.11 PHY/MAC layers.

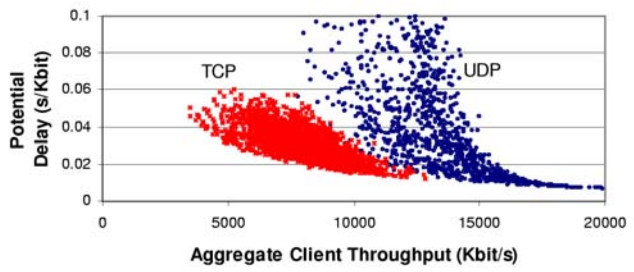


Figure 1: Network configurations vary in performance. Potential delay vs. ACT for each configuration.

3. SWARM: ASSUMPTIONS, MODEL AND ALGORITHMIC DESIGN

3.1 Basic Setup

Identifying the optimal organization of mesh nodes in frequency diverse trees requires knowledge about the connectivity of nodes, the quality of the links between them, and the conflict graph. All popular mesh network routing protocols collect the connectivity graph and the associated routing metrics through the operation of all nodes on a common channel. SWARM uses a similar setup. Each node in the network is capable of receiving advertisements from gateways, and other nodes on a default channel. Link metrics across mesh links are computed and propagated between neighboring nodes. Given that the entire network initializes on a common channel and all routing information is propagated across the network, gateways are in a natural position to construct a graph for the entire network, where each link is annotated with the respective routing metric, such as ETT or ETX. The gateways can further measure the network conflict graph, identifying those links that interfere with each other. A recent method [2] can measure such a conflict graph for WLAN infrastructure based networks in an accurate fashion in seconds. While this method requires synchronization which is hard in our setting (multi-hop), recent work on network-wide synchronization for wireless mesh networks [15] can be combined and potentially enable low overhead conflict graph construction in community wireless mesh networks as well. Upon the collection of the connectivity graph, the associated routing metrics, and the conflict graph, the gateways send all information to a SWARM server; an open computational platform in charge of searching for the optimal network configuration. That server could be collocated with one of the community mesh gateways or could be a separate network accessible entity.

[6]The worst performing configurations for UDP can have PD more than 1.0 due to severe unfairness and are not shown for graph clarity.

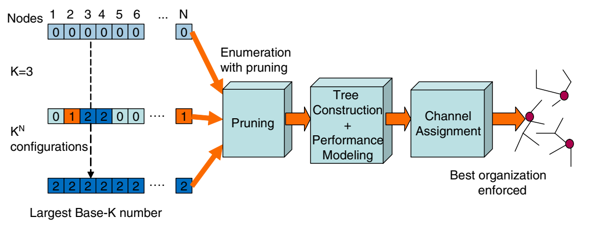


Figure 2: SWARM phases.

Note that the SWARM algorithm, as defined, is easily parallelizable and could thus be run in parallel by more than one community mesh gateways, each one exploring a different part of the configuration space. In the case when SWARM is a "standalone" service, one could envision mesh networks around the world using the SWARM servers to operate their network effectively.

A SWARM server computes the best network organization and communicates it back to the gateways. Subsequently each gateway decides to accept/reject association request from mesh routers according to the effective configuration and sets up forwarding state in the mesh routers accordingly. That best configuration will of course be specific to the network measurements collected at the time and the current network topology. Later in the paper, we discuss the network conditions that trigger re-measurement of the network state and/or recomputation of the best configuration.

The SWARM server searches for the optimal configuration in three separate phases described below, and presented in Figure 2.

3.2 Phase 1: Node Clustering

The first phase deals with how to assign nodes to gateways. This is performed through configuration enumeration and pruning. A configuration is defined by a unique mapping of mesh routers to gateways such that all mesh routers are associated with exactly one gateway. Thus in a network of N mesh routers and K gateways we need to find all possible ways of allocating N routers to K gateways. We can use base-K numbers to do this systematically. We first find the largest N-digit base-K number and iterate from 0 to this number. Each base-K number is a representation of a specific configuration. For example, 01200122 means that nodes 1,4,5 are with GW 0, nodes 2,6 are with GW 1 and nodes 3,7,8 are with GW3.

Thus, this phase enumerates K^N configurations from which we have to find the best performing. Due to the large number of configurations, we need to find heuristics that allow us to prune them before moving to tree construction and modeling, in order to save computation. These heuristics should only remove sub-optimal configurations.

Pruning the Search Space.

Through our exhaustive exploration of the design space in Section 2 we derive the following heuristics for pruning the search space.

Connectedness : A fundamental requirement in acceptable configurations is that all nodes should be connected to a gateway. In the first step, we prune those configurations that lead to disconnected nodes. In the sample scenario from Section 2 this step was able to prune 30% of the configurations, i.e. 1248 out of the 4095 total configurations.

Imbalance : This second heuristic is based on the intuition that poor overall performance can correlate with heavily imbalanced trees, where some trees have a very small number of nodes, whereas others have a significantly larger membership. This was verified using our sample scenario in Section 2 and other sample topologies we studied. For each configuration we calculate the difference in membership count of the most heavily and most lightly loaded tree. The configurations are ranked based on this metric and the bottom 75% are removed. Our motivating example in Section 2 showed that only the top 1-2% of the configurations are the best and thus 75% discard is generous. We verified across all our experiments that we do not lose any good configurations through this pruning.

Once the pruning steps are completed we are left with a set of configurations that cannot be ranked easily without further processing.

3.3 Phase 2: Tree Construction

The node clustering step outputs a set of configurations that need to be studied and ranked. Each configuration is a unique clustering of each node to one of the K gateways. Thus for each configuration we need to construct K trees (rooted at each gateway). Once these trees are built, the configuration performance can be modeled and ranked.

The typical algorithm used by state-of-the-art routing protocols such as LQSR is for each node to use Dijkstra's algorithm to find the path to each gateway and choose the one that minimizes the path metric. While Dijkstra's algorithm is convenient, it only provides a myopic view of the network to each node. If the link metric used is ETT, using Dijkstra's algorithm to construct a tree simply optimizes the ETT to reach each node *assuming no other nodes operate in the network*. Since ETT measurements tend to be taken in the absence of cross-traffic, they do not take interference from other nodes into account.

The other option which is definitely not scalable is to enumerate all possible trees in a configuration, model each of them and choose the best one [26]. For wireless networks, finding the best tree to optimize some overall performance metric taking into account interference is a difficult problem for which no scalable solutions appear to be available at this time. Therefore, it is necessary to develop heuristics.

We propose a greedy interference-aware heuristic tree construction technique for SWARM. The algorithm is shown in Table 1. Essentially the algorithm takes a set of nodes and a gateway, the link metrics between these nodes (annotated connectivity graph G) and their interference relationships (conflict graph C) and greedily constructs a tree to optimize performance. At each step the algorithm (i) tries to attach each remaining node, not part of the tree, to each node already part of the tree, (ii) chooses the best node *and* best point of attachment and (iii) makes this node permanent. A similar strategy of incremental construction was also used in [16]. This process is continued until all nodes are part of the tree. Critical to this algorithm is a way to determine which is the best node to add to the current tree which requires some way to model the performance of the tree, which we elaborate on next. This algorithm is superior to Dijkstra's algorithm for our application scenario because

```
INPUT: 1. $G(V,E)$: Graph of the network with link metrics.
2. $s$: id of the gateway node. 3. $C(V',E')$: Conflict graph.

CONSTRUCT_TREE($G$, $s$, $C$)
(1) for each vertex $v \in$ V [G] do $\pi$[v] ← NIL
(2) T ← T ∪ {$s$} //T contains nodes already added to the tree.
(3) Q ← V[G] - {$s$}
(4) while Q ≠ $\phi$
(5)     do $u$ ← EXTRACT_BEST_NODE($G$, $Q$, $T$, $\pi$)
(6)         T ← T ∪ {$u$}

EXTRACT_BEST_NODE($G$, $Q$,$T$,$\pi$)
(1) AttachmentPoint, BestThroughput, BestNode ← NIL
(2) for each vertex $v \in$ Q
(3)     do for each vertex $u \in$ T
(4)         do THR = MODEL($T$,$G$,$s$,$u$,$v$,$\pi$,$C$)
(5)         if THR > BestThroughput
(6)             then BestNode ← {$v$}, AttachmentPoint ← {$u$}
(7)                 BestThroughput ← THR
(8) $\pi$[BestNode] = AttachmentPoint
(9) return BestNode // return node to be added to tree.
```

Table 1: Phase 2: Greedy Tree Construction.

the tree is constructed by making current choices aware of the interference due to past choices.

Performance Modeling.

Performance modeling of a tree is an integral part of the tree construction technique. The predicted performance from the model for each of the K trees in a configuration can be combined to rank the overall goodness of the configuration. Each of the trees is modeled separately since we assume that each gateway is on a perfectly orthogonal channel. While a linear programming framework [12] can perform this modeling, execution of a linear program can be computationally expensive especially on embedded devices used in community wireless network gateways and even on higher-end servers. In addition, lengthy computations can be potentially irrelevant on completion due to network changes, such as failures or significant link quality changes. One of the contributions of SWARM is the use of a lightweight performance modeling method. Note, however, that at this stage we could actually plug in any performance evaluation algorithm, as long as it's capable of returning an answer within seconds. Our experience with the state of the art has been that typical times for the evaluation of such a problem tend to exceed 2 hours.

Our method models the throughput of a tree using an estimate of the amount of time that it will take for each mesh router in the tree to receive a packet, denoted by T_{cycle}. The model further incorporates the effect of both the ETT of a packet across an edge in the tree but also how often a specific mesh node will have the opportunity to send a packet to its children, thus taking into account self-interference. Given that we target a configuration where every node in the tree receives a packet, then the load on the tree edges can be easily computed from the number of each mesh node's descendants.

We denote the number of transmissions between mesh routers MR$_i$ and MR$_j$ by L_{ij}. We further set $L_{i0} = 1$ to denote that each leaf node in the tree will receive one packet per cycle, and will serve its children such that they receive equal throughput. $L_{ij} = 0$ if nodes MR$_i$ and MR$_j$ do not exchange traffic.

For each $i, j \neq 0$, we define T_{ij} to be the estimated transmission time for one data unit from MR$_i$ to MR$_j$. The amount of time it takes for a data unit to be transmitted from one mesh node to the other will also depend on the probability of successful access to the medium, negatively impacted by the way conflicts arise in the network. We denote by δ_{kij} the probability of node MR$_k$ sensing an ongoing transmission from MR$_i$ to MR$_j$. $\delta_{kij} = 1$ if a conflict is sensed and 0 otherwise. Then, the probability that a node MR$_k$ is going to be blocked by a transmission from MR$_i$ to MR$_j$ during T_{cycle} is given by:

$$p_{kij} = \delta_{kij} \frac{L_{ij} T_{ij}}{T_{\text{cycle}}} \tag{1}$$

Let W_{kij} be MR$_k$'s waiting time if blocked by a transmission from MR$_i$ to MR$_j$. We use a first-order residual-life approximation [13], namely

$$W_{kij} \approx \frac{1}{2} T_{ij} \tag{2}$$

Let W_k be the average blocked time for MR$_k$ to send 1 data unit at a time, and B_k the average busy time (blocked plus transmission) for MR$_k$ in a cycle. Then $B_k = \sum_h L_{kh}(W_k + T_{kh}) = T_k + W_k L_k$, where $T_k = \sum_h L_{kh} T_{kh}$ and $L_k = \sum_h L_{kh}$. $W_k = \sum_i \sum_j p_{kij} W_{kij}$, determining the average block time across all interfering transmissions. The duty cycle T_{cycle} in such a system is determined by the bottleneck node MR$_k$, lending $T_{\text{cycle}} = B_k$. From (1),

$$B_k = T_k + L_k \sum_i \sum_j \delta_{kij} \frac{L_{ij} T_{ij}}{B_k} W_{kij}.$$

This has solution

$$B_k = \frac{1}{2}\left(T_k + \sqrt{T_k^2 + 4L_k \sum_i \sum_j \delta_{kij} L_{ij} T_{ij} W_{kij}}\right)$$

The bottleneck node determines T_{cycle}, so $T_{\text{cycle}} = \max_k B_k$. By (2),

$$T_{\text{cycle}} \approx \max_k \frac{1}{2}\left(T_k + \sqrt{T_k^2 + 2L_k \sum_i \sum_j \delta_{kij} L_{ij} T_{ij}^2}\right)$$

The tree throughput (bit rate) is NS/T_{cycle} where N is the number of MRs and S is the number of bits in 1 data unit.

We have validated the accuracy of the presented model in *ranking the different possible network configurations* using extensive simulations, whose results are presented in the Appendix A. Based on that evaluation, the proposed model is highly accurate, despite its simplifications, while being able to explore the entire space of network configurations in a matter of seconds, compared to computationally intensive integer linear programs. Note that the model is able to quantify the throughput of the tree under fully saturated conditions. We feel this is an appropriate compromise since capturing workload across nodes across time would have a significant overhead. Our tree will be able to sustain the "worst-case" workload when all nodes request service continuously. Finally, the model presented ranks configurations in terms of their overall throughput and not potential delay. Note that grossly unfair configurations (with very high PD) will have already been disqualified from consideration when we prune the search space for imbalance.

3.4 Phase 3: Channel Diversity

All previous SWARM components operated under the assumption that the derived trees did not interfere. Introducing channel diversity would mean assigning each tree to an orthogonal frequency. However, there will be cases when there are not enough orthogonal frequencies. To avoid performance degradation due to overlap in the footprint of different trees, we use a distributed channel-hopping [18] scheme, i.e. each gateway selects a sequence of channels to hop through. The benefits of this approach are: (1) If the number of channels available are fewer than the number of gateways (and their potential area of operation) interfering in a given area, then channel hopping will perform better than even an optimal static channel assignment to the gateways [18]. (2) Channel hopping is robust to sources of external interference and jamming attacks [11]. (3) Channel hopping decouples the self-organization problem in SWARM from channel selection. This simplifies the self-organization algorithm considerably. Without this, a combined problem would have to consider the interference of a tree at one gateway on other trees which again is very complex.

Our algorithm works as follows: **(1)** Each gateway first obtains the hopping sequences of other *conflicting* gateways through the SWARM server. The notion of *conflicting* gateways defined in SWARM is different from that of interfering gateways: even if gateway A is not directly in communication range or interference range[7] of gateway B, the mesh routers that are part of the distribution tree of gateway A may interfere with gateway B or the nodes that are part of the distribution tree of gateway B. Essentially a gateway needs to obtain the hopping sequences of gateways whose operating "footprint" may interfere with its own operating "footprint". Once the best configuration is found using SWARM, the conflicting gateways can be simply determined by each gateway checking for a link in the conflict graph between itself or a single node in its tree with nodes in another gateway's tree (including the other gateway). **(2)** Each gateway on receiving hopping sequences from conflicting gateways performs the channel hopping algorithm from [18] shown in Table 2. We extended the original algorithm to appropriately define the set $N(x)$ (set of interfering nodes) for community wireless networks. The gateway x executing the algorithm chooses a hopping sequence that maximizes its throughput for each slot by choosing a channel that minimizes conflicts with other gateways (by minimizing η from Table 2), breaking ties randomly. The Initialize routine bootstraps the nodes and the Hop routine is executed every $N_s \cdot t_s$ seconds to compute a new hopping sequence. **(3)** Once a hopping sequence is decided upon by a gateway, each level of the tree rebroadcasts this sequence to trigger a channel switch according to the hopping sequence at every lower level of the tree.

3.5 CWN Reorganization

Community wireless networks are dynamic entities and their performance will vary according to the underlying conditions. There are two types of events where re-organization will be needed to rectify overall network performance: i) node failures, and ii) changes in link quality between nodes. The response of the network to those two types of events is likely to be different. In the case of link failures, recovery of network connectivity is of the utmost importance. As a result, our solution will aim to rectify connectivity and potentially operate in a slightly sub-optimal state until the new, optimal configuration has been identified. In the case of quality changes, one needs to study the way such changes manifest themselves in actual deployments before designing a solution. If changes happen too frequently, a complete re-execution of SWARM will prove impractical. In addition, in such extreme cases, tracking the network optimal will be associated with extreme cost, which may not make practical sense to start with. If, however, network dynamics can be well captured in medium timescales, greater than the amount of time required for the identification of the optimal configuration under SWARM, then one could possibly imagine a complete re-execution upon meaningful events.

[7]This is the the distance at which a node can interfere with the transmissions of another node although these two nodes cannot actually communicate.

```
INPUT: 1. $V$: set of gateways. 2. $x \in V$: executing gateway.
3. $N(x)$: gateways conflicting with $x$. 4. $N_s$: periodicity of
hopping sequence. 5. $K$: number of channels. 6. $\lambda_i$: represents
a hopping sequence. 7. $\rho(a,b)$: 1 if a==b, else 0. 8. $\eta(j)$: number
of conflicts if gateway $x$ used channel $j$. 9. $t_s$: duration of a slot
in the sequence.
OUTPUT: A channel sequence of $N_s$ entries.

INITIALIZE($x$)
(1) Set $\lambda_i(x)$ = random(1...$K$), for i=1...$N_s$
HOP($x$)
(1) for i = 1..$N_s$ do
(2)     for j = 1..$K$ do
(3)         $\eta(j) = \sum_{u \in N(x)} \rho(j, \lambda_i(u))$
(4)     $\eta_{min}$ = min($\eta(j)$), $\forall$ j=1...$K$
(5)     Let $C$ = {c:$\eta(c)=\eta_{min}$}
(6)     $\lambda_i(x)$ = random($C$)
```

Table 2: Phase 3: Channel Diversity.

Network	1	2	3	4
Nodes	77	20	14	10
Events	0	10	2	31

Table 3: Node failure/repair events seen in 4 operational mesh networks over 72 hours.

Mesh Router Failure/Recovery: We studied 4 operational mesh networks of various sizes deployed in low income communities in the pacific northwest (details about the networks shown in Table 3) to assess the rate of node failures. These networks are highly representative since they are used by home users for Internet access. For a period of 3 days, and at 2 minute intervals, we queried the service provider's online monitoring tool [19] to determine which nodes happened to be offline. Table 3 shows the number of node failure/recovery events in 72 hours. We notice that while network 1 did not experience any such events, network 4 had 31 failure/repair events over a period of 3 days. In all cases, a new event occurred more than 100 seconds after the previous one, and on average that time was on the order of hours. These events typically occur due to software resets, configuration errors, physical disruption etc.

Link Quality Changes : The next important cause of network dynamics is change in link quality, that could occur due to environmental changes (people moving, new obstacles, interference etc). Based on a 14 node wireless testbed in an office building we measure the value of ETT across all links every 60 seconds for a week. We then compute the amount of time between link quality changes of more than 50%. We find that significant link quality changes take a few 100s of seconds, with an average value close to a few hours. Similar results have been reported in [23]. While these were completely indoor deployments, they are partially representative of community mesh deployments where mesh routers may be placed in users' homes.

SWARM execution time: Using our implementation of SWARM on a network with 2 gateways and 13 nodes, we measured the time SWARM took to return the optimal configuration across a number of experiments. That time never exceeded 100 seconds on an Intel 1Ghz PC with 512MB RAM, while including i) the time needed for the detection of the network change (in our case node failure), ii) the computation of the new topology, and iii) its dissemination. Importantly, the SWARM workload is naturally parallelizable and could thus allow for even faster execution times, especially given that CWNs are unlikely to exceed a few 100s of nodes, due to their inherent performance limitations that arise with a large number of hops.

The aforementioned execution time did not include the time required for the measurement of the conflict graph. In fact, the conflict graph would not need re-measurement upon node failures. New node arrivals could be easily incorporated through partial measurement from the new node itself. However, the conflict graph could change in reaction to link dynamics. We believe that the overhead of conflict graph re-measurement and SWARM computation would make it impractical to react to network dynamics on the order of a few minutes. The network operator would be better off ignoring short term changes and set a minimum amount of time between re-organizations to amortize the associated overhead. Given the limited variability in our measured conflict graphs, we would recommend a re-organization at most once every 5 minutes or so. Note that most significant network changes take place on the order of hours anyway.

Given SWARM's low re-execution numbers one could advocate the complete re-organization of the entire network in the presence of failures or significant changes in network conditions as described above. While such a solution is likely to address most scenarios, we however need a solution to deal with node failures that may partition the network.

Re-execution does not mean dis-connection In the case of a node failure, the immediate children of the failed node cannot send data upstream and the parent of the failed node cannot send data downstream. Our solution to this problem has as follows. Under SWARM, each MR does not only know the best path to the rest of the nodes, but is aware of the entire tree and edge metrics, as computed by the SWARM server. Each MR caches these tree structures in memory. In the event of a failure in reaching the upstream parent MR or one of the children MRs, the affected MR uses Dijkstra's algorithm over these cached trees to compute an alternate next hop for the destination of each packet. It also notifies the gateway of the failed node which then recomputes a new network configuration via a request to the SWARM server. Thus, until a new network configuration is enforced taking into account the failure of a node, each MR can still provide service by using alternate next hops.

In summary, SWARM is able to react to node failures restoring connectivity if possible. Furthermore, it can adjust to medium term link dynamics through complete re-execution, given the limited required computation and measurement time.

4. EVALUATION AND EXPERIENCE

SWARM is implemented in C and works on Linux, FreeBSD and Windows to accommodate a variety of community network deployments. It has two main components: (1) A routing and monitoring component for the gateways and mesh routers, and (2) A server component that runs the 3 phases of SWARM.

On a mesh router the routing/monitoring component is a user-space daemon performing link metric measurements, reporting those to the gateways and responding to gateway commands regarding operating channels and next hops. It furhter reacts to packet forwarding failures by choosing alternate paths. On the gateway this daemon also communicates with the SWARM servers via a proprietary protocol. The SWARM routing protocol is built on top of the OLSR routing protocol code base. SWARM extensions are a few thousand lines of code while the entire protocol code is around 25000 lines.

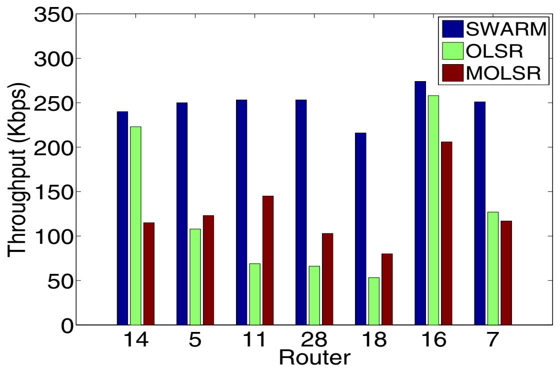
The SWARM server is a C++ application that accepts compute requests from gateways, finds the best topology and returns it to the gateways.

In what follows we provide a quantitative evaluation of SWARM's performance and contrast it to today's state of the art, i.e. OLSR and a channel-diverse version of OLSR that we developed and which we call MOLSR. More importantly, our evaluation follows a number of steps that allow us to comment on the efficiency of SWARM's building blocks, i.e. tree construction, clustering, and channel diversity, and report on their respective contribution to SWARM's performance.

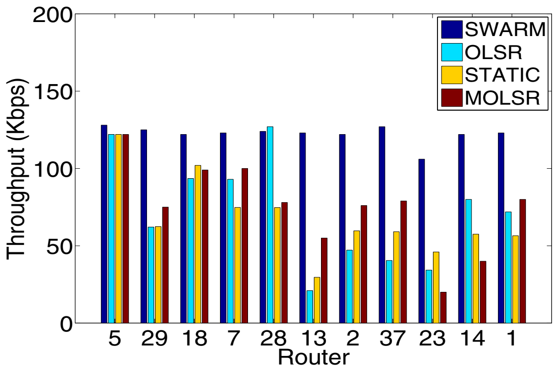
Our results are based on the deployment of SWARM on a 20 node wireless testbed, depicted in Figure 3. Each node features a Senao Prism 802.11b radio. Baseline results are reported on the latest version of OLSR, a protocol widely used in mesh networking with an open source implementation that has been well maintained, debugged and updated to include recent innovations such as measurement-based link metrics from SrcRR.

4.1 Single gateway: tree construction

When SWARM is applied on a single gateway network, one can study the efficacy of its tree construction component, i.e. what benefits a structured approach to organizing a set of nodes and a gateway provide over using state-of-the-art approaches. We deployed SWARM on 8 nodes (5,11,13,18,7,28,14,16) of the wireless testbed in Figure 3 and configured node 13 as a gateway. Each mesh router downloaded data using UDP at 256Kbps from the gateway node (some clients required multi-hopping) using `iperf` for 5 minutes. Figure 4(a) shows that SWARM allows this network to deliver a throughput of 256 Kbps to each mesh router unlike a state-of-the-art routing protocol (OLSR). Overall the sum of MR throughputs in SWARM is 1.92× that of OLSR and the sum of inverse of MR throughputs (Potential Delay) in SWARM is 2.64× better than OLSR due to interference aware tree construction. To verify that the gain over OLSR was not simply because of metric oscillation due to traffic [8], we ran an experiment with OLSR and froze the routing table entries after 5 minutes. This



(a) 1 gateway



(b) 2 gateways

Figure 4: SWARM performance with 1, and 2 gateways. SWARM is compared with OLSR, STATIC and MOLSR. SWARM results in higher throughput for clients, while maintaining fairness.

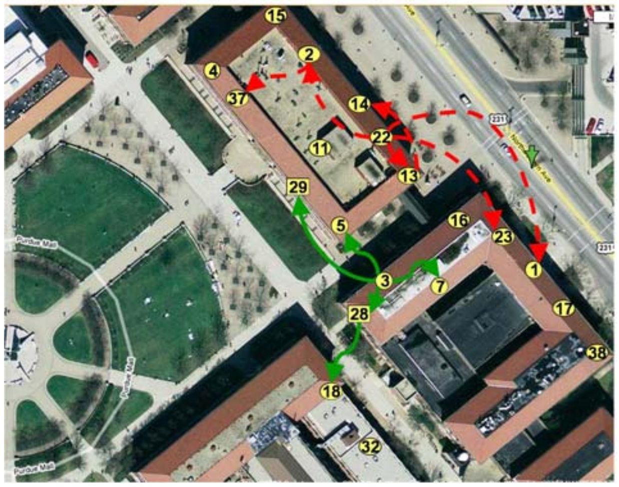


Figure 3: Wireless testbed used for SWARM evaluation.

version called ”STATIC” was also found to be worse than SWARM thus verifying that *SWARM performs better due to the choice of better network topologies.*

4.2 Multiple gateways

We now evaluate SWARM in the presence of multiple gateways, quantifying the benefit of both clustering and tree construction. We compare the performance of the topology selected by SWARM with that derived using OLSR and a multi-channel version of OLSR, which we call MOLSR. Upon convergence, OLSR has defined a set of trees, each one sourced at each gateway. MOLSR simply moves each one of those trees to its own, orthogonal frequency, thus giving OLSR the benefit of channel diversity, and offering a fair comparison point for SWARM. Notice that MOLSR is going to improve upon OLSR due to the use of additional frequencies. However, it will still fail to incorporate the effect of interference in constructing efficient distribution trees. Thus, it serves as a nice counter-example that goes to show how important it is to take a structured approach to addressing the issue of self-organization in CWNs.

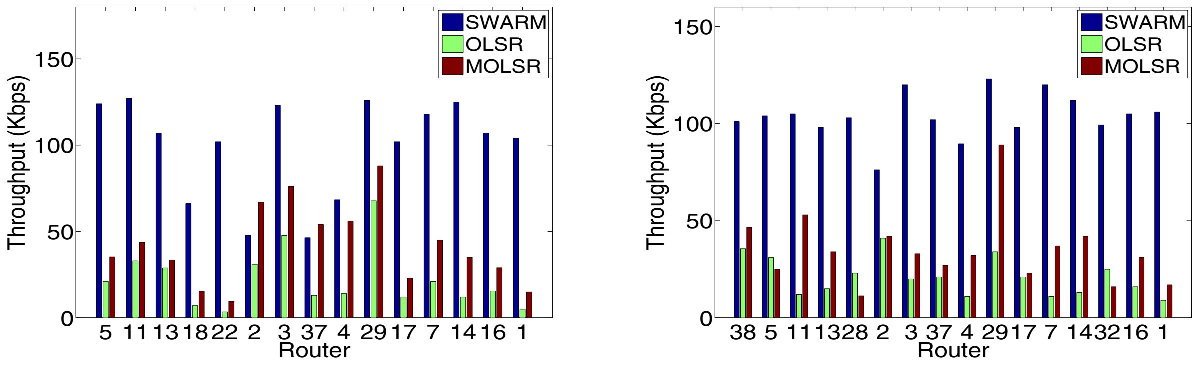
4.2.1 Two Gateways

We now evaluate a scenario with two gateways in the network. We deployed SWARM on 13 nodes (5, 13, 18, 22, 2, 3, 37, 29, 23, 7, 28, 14, 1) of the wireless testbed in Figure 3 and configured node 22 and 3 as gateways. Each node downloaded data using UDP at 128Kbps from its chosen (best) gateway node (some clients required multi-hopping). The throughputs achieved by the 11 mesh routers under the different protocols are shown in Figure 4(b). We find that the PD using SWARM is better than OLSR by 2.2×. SWARM allocates 6 MRs to one gateway and 5 MRs to the other one and operates them on different channels as well as performs interference-aware tree construction for each gateway. This reduces the interference in the network and improves performance. Interestingly, SWARM is also 2× better in PD than a multi-channel version of OLSR (MOLSR). This shows that *naive algorithms cannot effectively exploit the presence of multiple channels and the intelligent clustering and tree construction algorithms of SWARM are required to better utilize the network resources.*

In search of an optimal solution, we also experimented with the preliminary use of a mixed integer program using the model proposed in [12] (with extensions for multiple channels, multiple flows and single path routing) in CPLEX. For this two-gateway network, the LP did not converge to a solution after more than 2 hours which makes such an approach inappropriate for highly dynamic, network environments as those of CWNs.

4.2.2 Three Gateways

We now evaluate a more scaled up network scenario with 3 gateways. This is an interesting scenario as it stretches to the limit the channel separation possible with 802.11b, i.e. 3 orthogonal channels can be used by 3 gateways. We deployed SWARM on 18 nodes (1-5, 7, 11, 13-18, 22, 23, 28, 29, 37) of the wireless testbed in Figure 3 and configured nodes 15, 23 and 28 as gateways. Each node downloaded data using UDP at 128Kbps from its chosen (best) gateway node (some clients required multi-hopping). The results of this scenario are shown in Figure 5(a). In this scenario OLSR has a PD 7.75× higher than SWARM. Thus we can see that as more channels can be exploited and more interference is present in the network (due to a larger set of



(a) 3 gateways (b) 4 gateways

Figure 5: SWARM performance with 3 and 4 gateways. SWARM is compared with OLSR, STATIC and MOLSR. SWARM results in higher throughput for clients, while maintaining fairness.

nodes), the gain from SWARM becomes even more significant. Again, we see that exploiting multiple channels naively using MOLSR is not good enough and has a PD 3.25× higher than SWARM. The slight loss of throughput for some nodes is due to inter-channel interference, even when orthogonal channels are used.

4.2.3 Summary

The gains from SWARM can be significant in real network deployments and are summarized in Table 4(a). Notice that the PD gain is more significant than that of ACT which shows that *SWARM is particularly effective at finding configurations that provide a fair throughput to all mesh routers*. The gain is not exactly correlated with the number of channels that can be utilized since performance is a complex function of link quality, node density, gateway placement and interference. However, as the network size or the number of channels increases, the PD using SWARM is expected to also improve.

4.3 Multiple gateways: channel diversity

When the number of gateways exceeds the number of available, orthogonal frequencies, then SWARM's channel hopping feature becomes important in mitigating interference. We deployed SWARM on all nodes of the testbed in Figure 3 and configured nodes 15, 22, 23 and 18 as gateways. In this scenario, since there are only 3 orthogonal channels available for 4 gateways there is always some inter-channel interference in MOLSR. To minimize its impact, we configured gateways 15 and 18, as the ones farthest apart, to use the same channel. For SWARM, the channel hopping parameters chosen were: t_s=3 seconds and N_s=20. The results of this scenario are shown in Figure 5(b). We observe that in SWARM, the channel hopping helps to "share the suffering", resulting in a more equitable distribution of network resources and better performance for everyone. The throughput improvement is dramatic. As a result, the PD of OLSR and MOLSR is 5.9× and 3.7× worse than SWARM respectively. Thus, *SWARM is effective at utilizing channel diversity even when not enough channels are present in the network*.

4.4 Impact of Traffic Intensity

We now evaluate SWARM when the download rate from each MR is varied from 64Kbps to 512Kbps. We show the ratio of PD obtained by OLSR and MOLSR over SWARM in Figure 6(a). The results show that if the traffic intensity is low (64 Kbps) the gains are small. This is because there is enough network capacity to sustain the download rate despite the use of inefficient network topologies in OLSR and MOLSR. As the download rates increase and approach the network capacity, the gains from SWARM increase. For example, at 256 Kbps, the PD of OLSR and MOLSR are 3.04× and 2.85× worse than that achieved by SWARM. Increasing traffic demand leads to network resource scarcity, thus amplifying the need for intelligent network organization. Interestingly as we further increase the download rate the gains drop. This is because at this point the download rates cannot be sustained by the network and unfairness cannot be solved by good network organization alone. In general, we expect network operators to decide service rate plans for users [7] similar to other broadband services. In that case, we would expect SWARM to be able to accommodate more users or support higher rate plans.

4.5 Impact of variable traffic

Our evaluation so far quantifies the performance of SWARM when the routers' demands are equal and simultaneous - approximating fully saturated conditions. Here we would like to understand the gains of SWARM when the traffic across the network does not match this profile. To do that we introduce spatial diversity in user demands, i.e. different mesh routers request service at different points of time. More precisely, mesh routers are initiating their download at a random time in the 5 minute evaluation window. The result of such a setup is that the degree of interference and overall network load fluctuates with time as new flows are added.

Our results are presented in Table 4(b). The gains are slightly lower than in previous scenarios, they are still significant compared to the state of the art. The gains are lower because for some period of time, there is enough network capacity due to inactive mesh routers and the benefits of a good SWARM topology for those periods of time is less significant for performance (also shown in Section 4.4).

(a) Fixed Rate UDP Traffic

GWs	ACT Gain	PD Gain
1	1.9×	2.64×
2	1.6×/1.7×	2.0×/2.2×
3	2.4×/4.5×	3.3×/7.8×

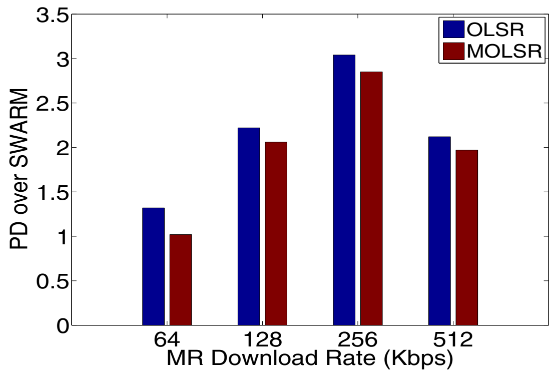
(b) Variable Rate UDP Traffic

GWs	ACT Gain	PD Gain
1	1.7×	2.1×
2	1.4×/1.5×	1.9×/2.0×
3	1.9×/4.0×	2.8×/5.4×

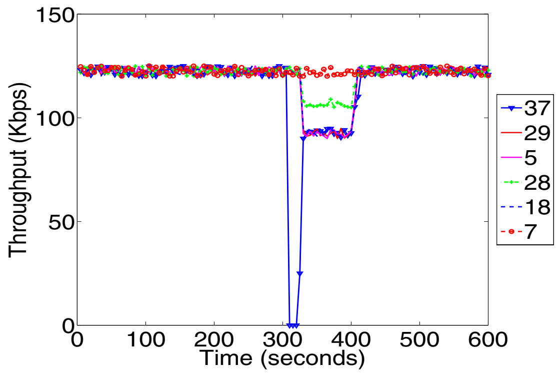
(c) TCP Traffic

GWs	ACT Gain	PD Gain
1	2.12×	2.73×
2	1.63×	2.66×
3	1.92×	3.63×

Table 4: Summary of performance comparison of SWARM against MOLSR/OLSR. ACT gain reflects SWARM's ACT over that of MOLSR/OLSR. PD gain shows the fraction of MOLSR/OLSR over SWARM, i.e. higher PD values indicate worse fairness.



(a) Varying Traffic Intensity



(b) Node Failure

Figure 6: SWARM performance (a) under different traffic intensities, and (b) in the presence of node failures, compared to OLSR and MOLSR. SWARM always outperforms OLSR/MOLSR while providing increased robustness to node failures.

4.6 Performance with TCP traffic

We used UDP traffic in our earlier experiments to explore the raw performance improvements possible via good network organization. We also perform TCP experiments using `iperf` in the same scenarios described earlier and show the gains observed using SWARM over MOLSR in Table 4(c). Note that due to TCP's congestion control, this experiment could feature arbitrary load. The results show that the PD obtained using MOLSR is significantly higher than that achieved using SWARM. In fact the reduction in potential delay (PD) achieved using SWARM is significantly higher using TCP than UDP. This is because TCP amplifies the unfairness between nodes far away and close to the gateway due to its sensitivity to packet loss which increases with hop count. In this case, good network topologies become even more important to sustain TCP window sizes and prevent starvation of some MRs. MOLSR results in a few nodes getting very low throughput consequently increasing the PD metric significantly.

4.7 Reorganization performance

An important goal behind SWARM's design was to efficiently re-organize the network under varying network conditions, i.e., node failures and link quality changes. To assess SWARM's ability to meet such a goal we carried out the following experiment. We use the same setup as that of the "dual gateway" experiment. The derived trees are shown in Figure 3. Clients initiate their downloads from their respective gateways but 5 minutes into the experiment we take down node 2. We then let the downloads run for another 5 minutes and measure how SWARM reacts to this event. We also show how this compares to the reaction of OLSR. The first node to notice the failure of node 2 is its child node 37. Node 37 uses its cached tree graph to find an alternate path to a gateway, and chooses path $37 - - > 29 - - > 3$. While using that path to route its traffic, it also notifies its gateway (node 3) of the failure. Notification of the gateway triggers the re-execution of SWARM until the new topology is computed. This entire process took approximately 100 seconds, starting from when node 2 fails, node 37 notices the failure (after 15 seconds) and notifies the gateway and SWARM returns a new topology (which takes 80 seconds). In the interim the alternate path is used to receive packets. Figure 6(b) shows the throughput of the affected nodes during the failure. Notice how node 37 falls to 0 and then recovers to around 90Kbps after discovering the failure and computing the alternate route. The nodes in the alternate route, i.e. nodes belonging to gateway 3 (refer to Figure 3) are affected by the join of the new node and some of their throughputs reduce accordingly. After the new topology is returned node 5 switches to gateway 22 and all nodes get 128Kbps again. The reaction of OLSR/MOLSR is similar in terms of failure detection since both use rapid link metric increase to deduce link failure. Subsequent to failure detection, OLSR/MOLSR find new routes and choose gateway 3. However, since both still use sub-optimal topologies, the PD of OLSR and MOLSR is still worse than SWARM by a factor of 1.98× and 2.14× respectively.

4.8 Limitations

The previous sections make a strong case for SWARM's ability to identify the few, elusive optimal configurations for channel diversity in CWNs. The performance gains achieved under both TCP and UDP traffic reach a 7 times improvement compared to the state of the art. However, SWARM's optimization relies on worst-case performance guarantees, where each mesh router is assumed to be actively using the network. While SWARM's optimization framework could incorporate different long-term workload estimates for the different mesh routers in the network, it would not be able to dynamically modify the topology based on instantaneous traffic demands. The associated overhead of such an action would be too great. Therefore, by design, SWARM trades true optimality for robust, practical deployment. Second, the SWARM optimization framework essentially partitions the problem into distinct phases, i.e. clustering, tree construction, distributed channel assignment. Such a choice makes the problem tractable while ensuring a fast enough execution to handle network dynamics. While it's not possible to assess how far from the optimal we are with today's technology, due to the computational complexity of the problem, we note that the benefits of our solution can provide dramatic improvement to today's installations making SWARM a valuable tool in the hands of CWN operators.

5. RELATED WORK

The problem of channel assignment in mesh networks has been a topic of interest in recent years. However, all work in that domain tends to address multi-radio nodes [10, 22, 24]. Our work is the first to address the issue of channel assignment and tree construction in community wireless networks, where nodes are found to feature a single radio, limited computation and traffic patterns primarily from and to the gateway.

Current performance models for mesh design typically use linear programming [3, 12, 14, 25]. We found such approaches heavyweight and too time consuming compared to the rate at which the network changes. In contrast, our modeling technique is lightweight and sacrifices accurate characterization for practical relevance.

Finally, we note that clustering in sensor networks [27] is fundamentally different from SWARM clustering in that it aims to optimize data fusion and energy consumption which are not very relevant metrics in mesh networking scenarios.

6. CONCLUSIONS

Current deployment of community wireless networks employ techniques from single-radio single-channel wireless mesh networks, ignoring the unique properties of their environment. A multitude of gateways motivates the need for self-organization algorithms that can structure the network into a number of frequency diverse distribution trees sourced at the gateways. While intuitively important, such a task involves great computational complexity. SWARM is a *practical* system for the organization of CWNs. Using experiments in a physical testbed we demonstrate that SWARM can lead to a fair distribution of traffic across clients that may improve performance by up to 7x! Avoiding complex linear programs SWARM is further capable of achieving such performance while keeping computational time to 100 seconds in our 20 node testbed. Notice that while the entire SWARM framework is specific to CWNs some of its components are general and have the potential for reuse in other wireless networking protocols. Notably, the tree construction and performance modeling components are certainly generalizable and could be reused in the context of mesh trees.

Acknowledgements

We would like to thank Dimitrios Koutsonikolas and Richard Gass for their help with the experimental setup.

7. REFERENCES

[1] Meraki networks. `http://meraki.com/`.

[2] N. Ahmed, U. Ismail, S. Keshav, and K. Papagiannaki. Online estimation of RF interference. In *Proc. of ACM CoNEXT*, 2008.

[3] M. Alicherry, R. Bhatia, and L. Li. Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. In *Proc. of ACM MobiCom*, 2005.

[4] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proc. of ACM MobiCom*, 2005.

[5] T. Clausen, P. Jacquet, C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot. Optimized link state routing protocol (OLSR). RFC 3626, Oct 2003.

[6] D. S. J. D. Couto, D. Aguayo, J. C. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proc. of ACM MobiCom*, 2003.

[7] S. M. Das, D. Koutsonikolas, and Y. C. Hu. Practical service provisioning for wireless meshes. In *Proc. of ACM CoNEXT*, 2007.

[8] S. M. Das, H. Pucha, K. Papagiannaki, and Y. C. Hu. Studying Wireless Routing Link Dynamics. In *Proc. of ACM Internet Measurement Conference*, 2007.

[9] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proc. of ACM MobiCom*, 2004.

[10] P. Dutta, S. Jaiswal, D. Panigrahi, and R. Rastogi. A new channel assignment mechanism for rural wireless mesh networks. In *Proc. of IEEE Infocom*, Apr. 2008.

[11] R. Gummadi, D. Wetherall, B. Greenstein, and S. Seshan. Understanding and mitigating the impact of rf interference on 802.11 networks. In *Proc. of ACM Sigcomm*, 2007.

[12] K. Jain, J. Padhye, V. N. Padmanabhan, and L. Qiu. Impact of Interference on Multi-hop Wireless Network Performance. *Wireless Networks Journal (WINET)*, 11(4):471–487, 2005.

[13] L. Kleinrock. *Queueing Systems, Vol. 1.* John Wiley, New York, NY, 1975.

[14] M. Kodialam and T. Nandagopal. Characterizing the capacity region in multi-radio multi-channel wireless mesh networks. In *Proc. of ACM MobiCom*, 2005.

[15] D. Koutsonikolas, T. Salonidis, H. Lundgren, P. LeGuyadec, Y. C. Hu, and I. Sheriff. Tdm mac protocol design and implementation for wireless mesh networks. In *Proc. of ACM CoNEXT*, 2008.

[16] S. Lee, S. Banerjee, and S. Bhattacharjee. The case for a multi-hop wireless local area network. In *Proc. of IEEE Infocom*, 2004.

[17] L. Massoulié and J. Roberts. Bandwidth sharing: objectives and algorithms. *IEEE/ACM Trans. Netw.*, 10(3):320–328, 2002.

[18] A. Mishra, V. Shrivastava, D. Agrawal, S. Banerjee, and S. Ganguly. Distributed channel management in

uncoordinated wireless environments. In *Proc. of ACM MobiCom*, 2006.

[19] NetEquality. `http://www.netequality.org/`.

[20] J. Padhye, S. Agarwal, V. Padmanabhan, L. Qiu, A Rao, and B Zill, Estimation of Link Interference in Static Multi-hop Wireless Networks. In *Proc. of IMC*, 2005.

[21] QualNet. http://www.scalable-networks.com.

[22] K. Ramachandran, E. Belding, K. Almeroth, and M. Buddhikot. Interference-aware channel assignment in multi-radio wireless mesh networks. In *Proc. of IEEE Infocom*, 2006.

[23] K. Ramachandran, I. Sheriff, E. Belding-Royer, and K. Almeroth. Routing stability in static wireless mesh networks. In *Proc. of PAM*, 2007.

[24] A. Raniwala, K. Gopalan, and T. Chiueh. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *ACM MC2R*, 8(2), April 2004.

[25] J. Tang, G. Xue, and W. Zhang. Maximum throughput and fair bandwidth allocation in multi-channel wireless mesh networks. In *Proc. of IEEE Infocom*, 2006.

[26] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides. On the Construction of Energy-Efficient Broadcast and Multicast Trees in Wireless Networks. In *Proc. of IEEE Infocom*, 2000.

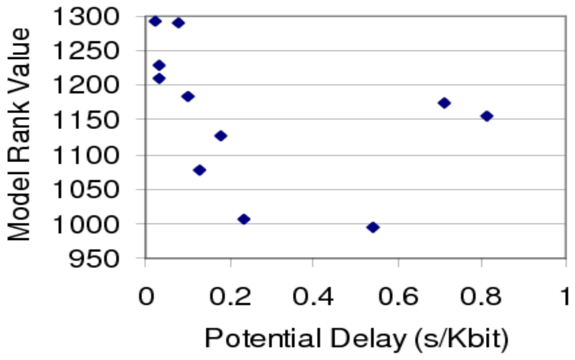
[27] O. Younis and S. Fahmy. Heed: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks. *IEEE Trans. on Mobile Computing*, 3(4):366–379, 2004.

APPENDIX

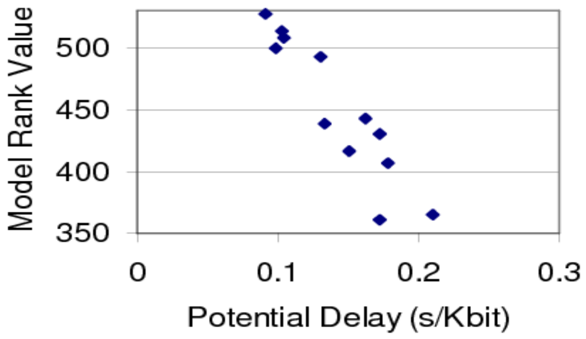
A. MODEL VALIDATION

While implementations can attest to the feasibility and the practicality of SWARM, simulations are the only method to assess the accuracy of SWARM's model and contrast its performance to a solution that would employ an exhaustive search of the entire configuration space. For that we used Qualnet [21]. Our simulations used the 802.11a MAC and PHY layers and auto rate fallback (ARF) for rate adaptation. The advantage of a commercial simulation environment, such as Qualnet, is that it incorporates a very realistic physical model. We used the two-ray propagation model, along with thermal noise and Rayleigh fading. The noise factor was set to 7dB. The Rayleigh fading model is appropriate for modeling CWN environments with many reflectors, e.g., trees and buildings, where the sender and the receiver are not in Line-of-Sight of each other. For each topology, we measured interference, similar to [20] and ETT between all nodes. Each node then reported the link metrics to each one of its neighbors to each nearby gateway.

In our experiment, we aim to evaluate the modeling accuracy of SWARM in different scenarios. Since the SWARM performance modeling phase aims to rank trees in terms of their expected performance, we simply use a single network topology of 12 randomly distributed nodes in a 800m×800m area and build, in turn, 12 trees each with a unique node as the gateway. For the same network topology we consider two scenarios: (1) We turn on Rayleigh fading which removes all spatial reuse opportunities in the networks (2) We turn off Rayleigh fading which allows several links in the network to experience spatial reuse.



(a) Spatial Reuse



(b) No spatial reuse

Figure 7: Accuracy of the SWARM performance model in ranking configurations, evaluated using Qualnet simulations.

For each one of the simulated network configurations we initiate traffic downloads from the gateway to each node in the tree. We measure the achieved throughput and compute the configuration potential delay, i.e. the sum of the inverse of the node throughputs, and the network throughput as defined by our model. We rank the different configurations according to increasing potential delay and present our results in Figure 7. Note that the model ranks configurations according to throughput. The results show that while there is not an exact one to one mapping between the ranks the model finds versus the actual performance of the topology, the model does find the best ones and assigns them a higher rank. We repeated these experiments for several different and larger network topologies and found that SWARM was able to construct the best tree through modeling. Moreover, it is effective in realistic propagation and interference environments and can take spatial reuse into account with acceptable computational complexity. This speed and accuracy of the modeling phase is critical when the configuration enumeration algorithm needs to rank thousands of trees quickly. Note that these accuracy experiment were mainly performed via simulation due to practical difficulties with testing out so many different topologies in a testbed. Tests on some toy topologies in a testbed provided similar findings.