A Distributed Constraint Optimization Algorithm for Dynamic Load Balancing in WLANs *

Shanjun Cheng[1], Anita Raja[1], Jiang(Linda) Xie[2], and Ivan Howitt[2]

[1]Department of Software and Information Systems
[2]Department of Electrical and Computer Engineering
The University of North Carolina at Charlotte
Charlotte, NC 28223
{scheng6,anraja,jxie1,ilhowitt}@uncc.edu

Abstract. The traffic load of Wireless local area networks (WLANs) is often distributed unevenly among access points. In addition, interference from collocated wireless devices operating in the same unlicensed frequency band may cause WLANs to become unstable, leading to temporarily failures of access points. This paper addresses how to dynamically balance the load and how to quickly respond to instability in WLANs. We propose a new decentralized load balancing framework for WLANs based on multi-agent systems, which maps the WLAN load balancing problem into a distributed constraint optimization problem. A distributed optimization algorithm, DLB-SDPOP, is designed to solve this problem and dynamically balance WLAN load in a fully distributed way. It focuses on pseudo-tree[1] repair instead of pseudo-tree reconstruction each time when instability problems occur. We empirically show that the proposed distributed constraint optimization algorithms improve WLAN load balancing performance significantly.

Key words: Distributed Constraint Optimization, Pseudo-tree Repair, Multi-agent Systems

1 Introduction

Wireless local area networks (WLANs) have become one of the most popular wireless technologies due to their low cost, simple installation, and great capability to support high speed data communications. However, research studies on operational WLANs have shown that the traffic load is often distributed unevenly among access points (APs) [1].

* This work is supported by the U.S. National Science Foundation (NSF) under Grant No. CNS-0626980.

[1] A pseudo-tree of a graph G is a rooted tree with the same vertices as G and has the property that adjacent vertices from the original graph fall in the same branch of the tree [8].

In a dynamic operational environment of WLANs, interference may significantly impact the signal quality of WLANs, and hence, impact the network management decision-making. The ability of the system to deal with such dynamic changes and move from a previous stable state to a new optimal stable state quickly is a critical issue.

In this paper, we map the WLAN load balancing issue to a distributed constraint optimization problem (DCOP) [7] and develop a decentralized framework based on multi-agent systems. We present a Dynamic Load Balancing-Distributed Pseudo-tree Optimization Procedure (DLB-DPOP), that is a variation of the DPOP algorithm [8] geared towards the WLAN problem. In Section 3, DLB-DPOP is shown to be complete (i.e., guaranteed to find the optimal solution). We then develop a distributed optimization algorithm, DLB-SDPOP, that finds initial assignments to reach steady state and uses *self-stabilizing* [5] pseudo-tree repair mechanisms to dynamically maintain load balancing criteria. The main contributions of DLB-SDPOP are (a) a self-stabilizing pseudo-tree repair mechanism that repairs the affected nodes in the original pseudo-tree instead of reconstructing the entire pseudo-tree each time a perturbation needs to be handled. (b) It can solve complicated load balancing situations with up to 50% agents simultaneously failing. (c) DLB-SDPOP has the complexity of $\mathcal{O}(dom^w)$, where dom bounds the domain size and $w =$*induced width* along the particular pseudo-tree chosen. The induced width is the maximum number of parents of any node in the induced graph [3]. In the worst case, the complexity converges at $\mathcal{O}(|dom^*|^{w^*})$, where dom^* =the set of all APs in the WLAN, w^* =induced width of the whole pseudo-tree.

The rest of the paper is organized as follows. In Section 2, the WLAN load balancing problem is described and mapped to a DCOP model. In Section 3, the proposed DCOP algorithm is explained. A motivating example is provided to illustrate the self-stabilizing mechanism in the developed DLB-SDPOP algorithm in Section 4. The performance results of the developed algorithm are presented in Section 5, followed by the conclusions in Section 6.

2 Problem

The goal of the WLAN load balancing problem is to dynamically assess the associations of mobile stations (MSs) at time t_k, find the optimal set of MSs under each AP based on the estimates of the states of the MSs at time t_{k+1}, and change the associations of specific MSs from one AP to another neighboring AP and finish handoffs by t_{k+1}. The *neighborhood* of a certain AP is the set of those APs with whom it has frequent interactions. We define $t_{delay} = t_{k+1} - t_k$, as the maximum time required to handoff one MS from one AP to a neighboring AP. We formulate the load balancing issue as an optimization problem. The optimization satisfies the following criteria:

Criterion I: The received signal strength (RSS) of each MS associated with an AP is above the minimum received power threshold γ. In this paper, γ is set to be -82 dBm.

Criterion II: Maximize the minimum received power by each MS in order to minimize the likelihood of packet loss. In this paper, we implement this criterion as:

$$\max_{i,j} \min(R_i^j(t_k)) \tag{1}$$

where $R_i^j(t_k)$ denotes the reward of MS_i being handed off to AP_j at time t_k.

Criterion III: Distribute the load amongst viable APs in order to increase fairness as well as the overall network-wide resource utilization. In this paper, we assume each MS has the same load to each AP and implement this criterion as:

$$\min\{\max \sum_{j,l,j\neq l} \mid MS_j^{Num}(t_k) - MS_l^{Num}(t_k) \mid\} \tag{2}$$

where $MS_j^{Num}(t_k)$ denotes the number of MSs assigned to AP_j at time t_k.

Note that Criterion II and III may not be met simultaneously. Therefore, it is important to trade-off between these two criteria based on network conditions. Under low network load, Criterion II could be given higher priority in order to decrease the likelihood of ping-pong effect, while Criterion III could be given higher priority when the network load is higher and load balancing is more critical to maintain capacity availability across the network. Since the focus of this paper is load balancing, we give Criterion III higher priority than Criterion II.

2.1 DCOP Model

We map the WLAN load balancing problem to a DCOP [7] model in the following way. The model is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$, where

- $\mathcal{A} = \{A_1, ..., A_n\}$ is the set of agents interested in the optimal solution; in the WLAN context, each access point AP_j is assigned an agent.
- $\mathcal{X} = \{X_1, ..., X_m\}$ is the set of variables; in the WLAN context, each AP_j has a variable X_i for MS_i, which represents the new associated AP after a handoff.
- $\mathcal{D} = \{d_1, ..., d_m\}$ is a set of domains of the variables, where each domain d_j is a set of APs in AP_j's neighborhood.
- $\mathcal{R} = \{r_1, ..., r_p\}$ is a set of relations where a relation r_i is a utility function which provides a measure of the value associated with a given combination of variables. In WLAN, $\mathcal{R}$ represents objective functions, which are the three criteria we defined for load balancing.

The goal of our algorithm is to find a complete instantiation X^* for the variables X_i that maximizes the sum of the utilities of individual relations in the multi-agent system, in other words, to find which AP each MS should be associated with so that all the criteria for load balancing can be achieved.

2.2 Related Work

Distributed algorithms like DSA/DBA [11] and distributed complete algorithms such as ADOPT [7], DPOP [8] have been proposed to solve problems modeled as distributed constraint optimization (DCOP). These algorithms have been applied to problems such as graph coloring and meeting scheduling. However, there are only few attempts to address real world scenarios using this formalism, mainly because of the complexity associated with these algorithms [6]. Choxi and Modi [2] proposed an approach to manage WLAN connectivity by optimizing node positioning using (static) DCOP. Among these algorithms, DPOP is a complete algorithm based on dynamic programming. It is a utility-propagation method that extends tree propagation algorithms to work on arbitrary topologies using a pseudo-tree structure. It can generate only a linear number of messages. DPOP has mostly been used in environments (sensor assignment and meeting scheduling) where assignment decisions have to be made up front and then the application remains mostly static.

Self stabilization in distributed systems [4] is the ability of a system responding to transient failures, eventually reaching a legal state, and maintaining it afterwards. This makes such systems particularly interesting because they can tolerate faults and are able to cope with dynamic environments [9]. SDPOP [9] deals with dynamic problems, where variables and constraints can be added/deleted at runtime. SDPOP has a scheme for fault containment and fast response time upon low impact failures. SDPOP has been implemented to solve the meeting scheduling problem with up to 10% of the agents having simultaneous perturbations.

3 Solution

We have developed a multi-agent-system (MAS) based decentralized approach for WLAN load balancing. A distributed load balancing (DLB) agent is located inside each AP. Each DLB agent cooperates with other DLB agents in its neighborhood to ensure load balancing across the entire WLAN. A DLB agent's neighborhood consists of those DLB agents with whom it has frequent interactions. DLB agent interaction is initiated by two event triggers: (1) a handoff event and (2) the need for load balancing among APs. A handoff event occurs when the RSS of one MS begins to drop below the threshold. When a certain AP is over-loaded, some MSs associated with this AP need to be handed off to its neighboring APs so as to increase fairness as well as the overall resource utilization of the WLAN. These interactions include the distribution of information and control-decisions. The DLB agent's decision process involves determining which MS needs to be handed off to which AP.

Upon receiving a handoff event trigger, the DLB agent in the associated AP initiates agent interactions within its neighborhood by sending request messages to inform its neighbors. It has the view of the pseudo-tree which is made up of its neighborhood and itself. Similarly, The neighboring agents also have partial views of the whole pseudo-tree separately. They send back response messages to

announce the possible new assignments based on local evaluations of the three criteria in Section 2. The associated agent calculates the utility value of each possible AP assignment, and runs the DLB-DPOP algorithm using the local utility values as initial inputs. After an optimal assignment is found, the handoff decisions are sent to the target agents.

3.1 DLB-DPOP Algorithm

The DLB-DPOP includes 3 phases:

Phase 1-DFS (Depth-First Search) Traversal: DLB-DPOP performs a distributed depth-first traversal of the network to establish a pseudo-tree structure [3]. This is similar to the pseudo-tree creation phase in DPOP.

We have defined two heuristic functions here: $Num(v)$ and $Low(v)$. For each node v, we call its preorder number $Num(v)$. $Low(v)$ is defined as:

$$Low(v) = \min\{Num(v), \min\{Num(w), \forall back - edge(v,w)\},$$
$$\min\{Low(w), \forall tree - edge(v,w)\}\} \quad (3)$$

$Low(v)$ is the minimum value of the preorder number of node v, the lowest preorder number of node w from all the back-edges connecting node v to node w and the lowest $Low(w)$ from all the tree-edges connecting node v to node w. This definition ensures that all the nodes in the same pseudo-tree "loop" (A pseudo-tree "loop" is formed by tree-edges and back-edges as a close circle.) are assigned the same value of $Low(v)$. They help us efficiently represent and identify the triggered AP when a handoff trigger happens, thus obviating the need to traverse the DFS tree in search of the triggered AP. Building the DFS tree takes $\mathcal{O}(|E| + |V|)$ time, where $|E|$ and $|V|$ are the number of edges and vertices in the pseudo-tree, respectively.

Phase 2-Utility Propagation: DLB-DPOP propagates utility messages (called $UTIL$ messages) which contain utility vectors sent bottom-up along the pseudo-tree starting from the leaves, only through tree edges. This step too is similar to DPOP, except that (a) the propagation of DLB-DPOP acts only in part of the whole pseudo-tree which is made up of the neighborhood of the triggered DLB agent. This is because handoff changes will mostly have local effects on the pseudo-tree, thus obviating the need for DPOP type of global propagation of $UTIL$ messages. (b) the values propagated in the $UTIL$ messages of DLB-DPOP are the reward values. We use the function $R_i^j(t_k)$ to denote the reward of MS_i being handed off to AP_j at time t_k, which can be expressed as:

$$R_i^j = U_i^j(t_k) - C_i^j(t_k) \quad (4)$$

$U_i^j(t_k)$ is the normalized signal strength above the threshold of MS_i from AP_j per unit time:

$$U_i^j(t_k) = \frac{\sum_{t_s}^{t_e} P_{ratio}}{t_e - t_s} \quad (5)$$

where $P_{ratio} = P_{receive} - P_{thres}$, $P_{receive}$ denotes the received power (dBm) of MS_i from AP_j at a certain time unit, P_{thres} denotes the threshold value (If $P_{receive}$ of MS_i from AP_j is lower than P_{thres}, MS_i should be handed off to another powerful AP so as to remain working. In our WLAN problem, P_{thres} is set to be -82 dBm.), P_{ratio} measures how much $P_{receive}$ is above P_{thres} at a certain time unit. t_s and t_e denote the earliest and last time at which the signal goes above P_{thres}. $C_i^j(t_k)$ is the estimated handoff cost function:

$$C_i^j(t_k) = \begin{cases} \frac{C_h}{t_e - t_s} \text{ if } t_e - t_s > t_{delay} \\ C_{max} \text{ otherwise} \end{cases} \tag{6}$$

C_h and C_{max} are constant values (dBm) representing the handoff cost. If the time duration of good signal is not long enough ($t_e - t_s \leq t_{delay}$), the handoff decision would be unnecessary. The reward value $R_i^j(t_k)$ is used to reduce the possibility of reaching myopic solutions.

Phase 3-Optimal VALUE Propagation: The optimal value assignments are then propagated top-down from the root node [8]. The root agent chooses the optimal assignment and sends $VALUE$ message to its children agents (a $VALUE$ message represents this assignment). Each child agent determines its optimal assignment based on the messages from Phase 2 and the $VALUE$ message and repeats the propagation process. When all the nodes finish choosing an assignment, the algorithm is completed.

In order to prove that our proposed DLB-DPOP algorithm is complete, we have to prove its correctness and liveness. We use the two heuristics $Num(v)$ and $Low(v)$ to generate a pseudo-tree. $Num(v)$ helps to sort the nodes and $Low(v)$ is used to distinguish all the nodes to different groups that each group forms a pseudo-tree "loop". Adding each node according to $Num(v)$ and $Low(v)$ leads to a compatible pseudo-tree. We extend the optimality proof for DPOP [8] to DLB-DPOP. A pseudo-tree has no cycles. This implies that all messages come from unrelated parts of the tree and these messages are hence accurate evaluations of the utility that can be obtained by the sub-trees belonging to each sender node for each value of the node. The upper bound of the utility obtained from the whole problem at a node can be accurately computed by summing up the messages, for each possible value of the node. The value that produces the maximum utility is then assigned to the node. This shows correctness of the algorithm. In addition, both the absence of cycles in the pseudo-tree and the fact that all the leaves initiate the message propagation guarantee that each node will eventually receive $m-1$ messages (with m as the number of neighbors) and hence it will be able to send its mth message. This also means that each node will receive a message from its last neighbor, thus terminating the algorithm. This proves liveness. Hence, the proposed DLB-DPOP algorithm is complete.

3.2 DLB-SDPOP Algorithm

DLB-DPOP regenerates a new pseudo-tree (Phase 1) excluding the fault nodes each time perturbations occur. It is obvious that changes in the pseudo-tree

structure will adversely affect the performance of DLB-DPOP since some of the $UTIL$ messages will have to be recomputed and retransmitted. Therefore, it is desirable to maintain as much as possible the current DFS tree. In addition, in the $UTIL$ protocol in DLB-DPOP, upon a perturbation all $UTIL$ messages on the tree-path from the fault node to the root are recomputed and retransmitted. This is wasteful sometimes, since some of the faults have limited, localized effects, which do not need to propagate through the whole problem.

Algorithm 1 DLB-SDPOP

DLB-SDPOP $(\mathcal{X}, \mathcal{D}, \mathcal{R})$
Each agent X_i executes:

Phase 1:DFS Traversal
$root \leftarrow electedleader$
assignNum($root$)
assignLow($root$)
afterwards, X_i knows $Num\,(X_i)$ and $Low\,(X_i)$.

Phase 2:Self-stabilizing Utility Propagation
store all new $UTIL$ messages $\left(X_i, UTIL_i^j\right)$
if any perturbation is detected in WLAN **then**
  case 1: agent X_i stops working
    for all constraints R_i^k of X_i in the pseudo-tree **do**
      deleteEdge$\left(X_i, R_i^k\right)$
    delete the single agent X_i from the pseudo-tree
  case 2: agent X_i resumes working
    connect X_i to existing agent X_j
    send_Message$\left(X_i, UTIL_i^j\right)$
    for all original constraints R_i^k of X_i **do**
      addEdge$\left(X_i, R_i^k\right)$

Phase 3:Optimal Value Propagation
$X_i \leftarrow v_i^* = choose\_Optimal\ (agent\_view)$
send $VALUE_i^l$ to all $X_l \in C\,(X_i)$
END ALGORITHM

We develop DLB-SDPOP by incorporating a self-stabilizing mechanism. DLB-SDPOP dynamically modifies/repairs the affected nodes in the original pseudo-tree retaining the topology and states of unaffected nodes when inconsistency is detected (e.g., one or several APs stop working for a moment or previously fault APs resume functionalities). We define the following variables: $UTIL_i^j$ is the $UTIL$ message that X_i sends to X_j; R_i^k is the constraint relationship between X_i and X_k; $VALUE_i^k$ is the $VALUE$ message that X_i sends to X_k; $Sep(X_i)$ is the set of ancestors of X_i in the pseudo-tree; $P(X_i)$ is the parent of X_i (the single node higher in the hierarchy of the pseudo-tree that is connected to X_i directly through a tree-edge.); $C(X_i)$ is the children of X_i (the set of nodes lower in

the pseudo-tree that are connected to X_i directly through tree-edges.); $PP(X_i)$ is the pseudo-parents of X_i (the set of nodes higher in the pseudo-tree that are connected to X_i directly through back-edges.); and $PC(X_i)$ is the pseudo-children of X_i (the set of nodes lower in the hierarchy of the pseudo-tree that are connected to X_i directly through back-edges.). The DLB-SDPOP algorithm is described in Algorithm 1. It starts with a DFS traversal. After this phase, each node is assigned $Num()$ and $Low()$. The Self-stabilizing Utility Propagation process ($line\ 6-16$, Algorithm 1) is initialized and then run continuously. DLB-SDPOP ends with the propagation of optimal values to each node.

Main functions in DLB-SDPOP are described in Procedure 1. The functions $assignNum(vertex)$ and $assignLow(vertex)$ respectively generate $Num(v)$ and $Low(v)$ for each node v in the pseudo-tree. In Phase 2 of DLB-SDPOP, deleting interfered agents and adding back previous agents cleared of interference are two main concerns. $deleteEdge(X_i, R_i^k)$ deletes a relation/constraint R_i^k depending on the type of the edge (whether they are tree-edge or back-edge), $addEdge(X_i, R_i^k)$ adds a new relation/constraint R_i^k between two existing agents based on their relative position (whether they are *ancestor-descendant* or *siblings*).

As mentioned in Section 1, DLB-SDPOP has the complexity of $\mathcal{O}(dom^w)$. The variables dom and w increase with the propagation of the neighborhood size of the whole WLAN. The increase of neighborhood size would bring in a larger set of involved APs and a local view of pseudo-tree for each AP with more nodes, leading to a larger domain size and an equal or larger induced width separately.

4 Motivating Example

We now describe the self-stabilizing pseudo-tree repair mechanism in DLB-SDPOP by illustrating a motivating example in a WLAN scenario.

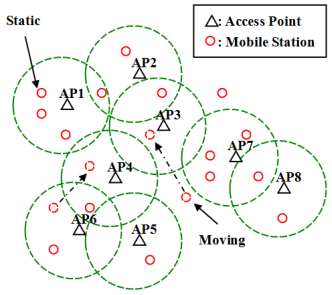


Fig. 1. 8-APs WLAN scenario.

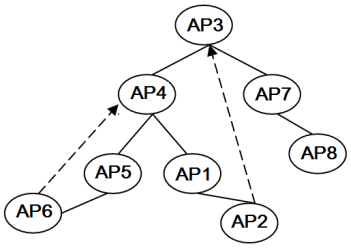


Fig. 2. Pseudo-tree for the WLAN scenario in Fig. 1.

Fig. 1 is a WLAN scenario with 8 APs and several MSs. All the APs are static. Some MSs are static and some are moving (as shown in Fig. 1). Each AP

has a coverage radius of 80 meters (shown as the dashed circles) which means the effective distance to associate a MS is 80 meters. If two APs have overlapping coverage areas, they are defined as *neighboring APs* (e.g., AP_5 and AP_4 are neighboring APs, while AP_5 and AP_3 are not.). APs can only hand off MSs to their neighboring APs.

Fig. 2 is the whole pseudo-tree generated by Phase 1($line\ 2-5$, Algorithm 1) as a global view for AP_3. All the neighboring APs in Fig. 1 are connected by tree-edges (solid lines) or back-edges (dashed lines) so that they can send messages to each other.

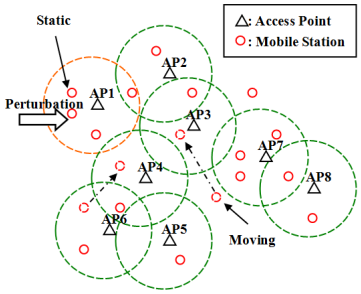


Fig. 3. Perturbation at AP_1.

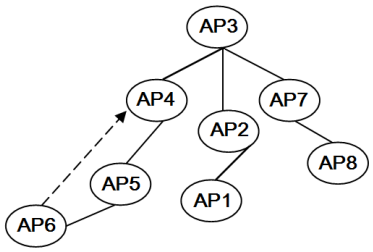


Fig. 4. Repaired pseudo-tree ($AP_1 - AP_4$ is deleted).

Suppose at time t_k, a new source of perturbation causes AP_1 to stop working (Fig. 3). AP_1 should be deleted ($line\ 8-11$, Algorithm 1) from the pseudo-tree in Fig. 2. AP_1 has two neighboring agents: AP_4 and AP_2 . We first consider removing tree-edge $AP_1 - AP_4$ ($line$ 24, Procedure 1). Removing the edge $AP_1 - AP_4$ does not disconnect the problem, but disrupts the structure of the pseudo-tree ($line\ 31-36$, Procedure 1). The nearest ancestor of AP_1 and AP_4 in the pseudo-tree is AP_3 ($line$ 32, Procedure 1). Thus the pseudo-tree repair begins from AP_3 and proceeds as follows ($line\ 34-36$, Procedure 1): $AP_3 \rightarrow AP_2 \rightarrow AP_1 \rightarrow AP_4 \rightarrow AP_3$. The result is depicted in Fig. 4. We should notice the role changes:AP_2 and AP_1 have switched parent/child roles. The $UTIL$ message between AP_2 and AP_1 has to be recomputed as well as that between AP_4 and AP_3, while other $UTIL$ messages can be reused. Then we consider removing tree-edge $AP_1 - AP_2$ ($line\ 26-30$, Procedure 1). AP_1 becomes a single node ($line$ 28, Procedure 1) that can be deleted directly, and AP_2 begins a new $UTIL$ propagation by re-computing its $UTIL$ message which does not include the previous message that sent from the sub-tree of AP_1 ($line$ 30, Procedure 1).

5 Empirical Evaluation

All experiments in this section were performed using Matlab optimization toolbox for simulation, running in an Intel Centrino Core Duo with 1.6GHz, 1G RAM

memory, under Windows XP. Values reported here are averages over at least 10 repetitions of the simulation. In the experiments, our baseline is DPOP, the existing state of the art. We show the computational advantage of DLB-SDPOP over DPOP and effectiveness of self-stabilization in DLB-SDPOP.

5.1 Metrics for Evaluation

We evaluate the performance of DLB-SDPOP and DLB-DPOP based on the following variables:

- # *of APs* - the total number of APs in the scenario.
- *Neighborhood size* -the number of neighboring APs of each associated AP.
- #*-changes* - the number of both pseudo-tree nodes which are added and deleted simultaneously.
- *Repair-cost* - the cost to reconstruct the pseudo-tree (measured by the number of messages transmitted to redefine the topology of the new pseudo-tree).
- # *of MS/AP* - the average number of MSs associated with each AP in the scenario.

We compare DLB-SDPOP and DLB-DPOP by measuring the metrics *Messages* and *Solving time* when perturbation occurs. *Messages* is defined as the total number of messages exchanged between DLB agents to respond to the perturbations and stabilize in a state corresponding to the optimal solution. *Solving time* is defined as the time (sec.) it takes for the DLB agent architecture to recover from the perturbations and stabilize in an optimal solution.

5.2 Scenario Setup

For the experiments reported here, we use four different scenarios or grids with different sizes ($3 \times 3, 4 \times 4, 6 \times 6$ and 9×9). Thus the variable # *of APs* is set to 9, 16, 36, and 81. Each grid in our simulation is wrapped around, i.e., if a MS moves out of one boundary of the simulation scenario, it moves into the scenario through the opposite boundary. The distance between APs is 80 meters. The link quality is based on the expected received power over a transmission distance of $d_{ij}(t_k)$ between AP_j and MS_i at time t_k given by $P_R(d_{ij}(t_k)) = P_T - (20\log_{10} f_c + 10\kappa \log_{10}(d_{ij}(t_k)) - 28)(dBm)$ [10]. In our simulation, 10% of MSs under each AP are randomly chosen to move in a random direction with a constant speed 0.5 m/s (MSs mobility percentage = 10%). We conduct simulation experiments with increasing *Neighborhood size* of 5, 9, 16 and 25. We set the #*-changes* to be 1, 2, 4, 5, 8, 16 and 36 to see the trends of both algorithms (We set the perturbation areas in the scenario randomly to influence the functionalities of APs) and set # *of MS/AP* to be 5 and 30 to simulate the scenario where each AP has few MSs and many MSs respectively. $C_h = 30dBm$ and $C_{max} = 25dBm$. Fig. 5 shows the simulation scenario.

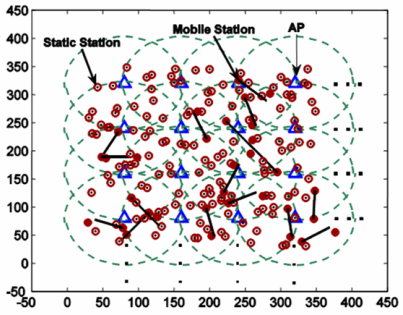


Fig. 5. Simulation Scenario.

# of APs	Algorithm	*Solving time*(sec.)	*Messages*
9	DPOP	0.053 ± 0.008	38 ± 5
	DLB-SDPOP	0.024 ± 0.006	72 ± 9
16	DPOP	0.109 ± 0.012	96 ± 27
	DLB-SDPOP	0.028 ± 0.009	147 ± 41
36	DPOP	0.458 ± 0.037	162 ± 42
	DLB-SDPOP	0.041 ± 0.013	253 ± 72
81	DPOP	8.522 ± 0.122	384 ± 83
	DLB-SDPOP	0.079 ± 0.034	540 ± 127

Fig. 6. DLB-SDPOP *vs.* DPOP (*Neighborhood size*= 5, *# of MS/AP*= 5, *#-changes*= 1)

5.3 Discussion

Fig. 6 provides the performance comparison between DPOP and DLB-SDPOP. DLB-SDPOP performs significantly ($p < 0.05$) better than DPOP on *Solving time* in all cases which shows the efficiency of pseudo-tree repair in DLB-SDPOP. Meanwhile, DLB-SDPOP consumes more *Messages* than DPOP which mainly occur in the $VALUE$ and $UTIL$ initiation processes (Phase 2, Algorithm 1). Given the importance of fast response time to perturbations in real-time WLAN scenarios, DLB-SDPOP is preferable.

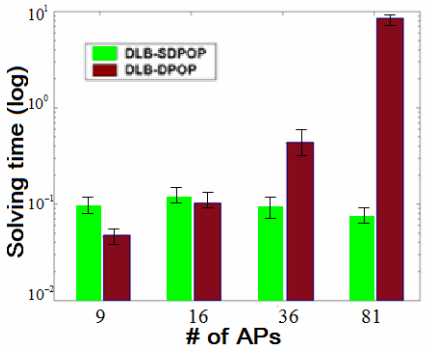


Fig. 7. *Solving time* (*log* scale) with variance for DLB-SDPOP and DLB-DPOP (*Neighborhood size*=5, *#of MS/AP*=5, *#-changes*=4), for *# of APs* to be 9,16,36 and 81.

In Fig. 7, *Solving time* of DLB-SDPOP increases a little when *# of APs* changes from 9 to 16 and decreases slowly when *# of APs* increases to 36 and 81. This is because the total number of MSs is getting larger when *# of APs* changes from 9 to 16. The increase in *Solving time* (*# of APs* from 9 to 16)

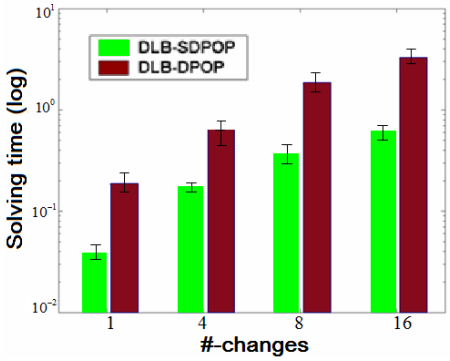


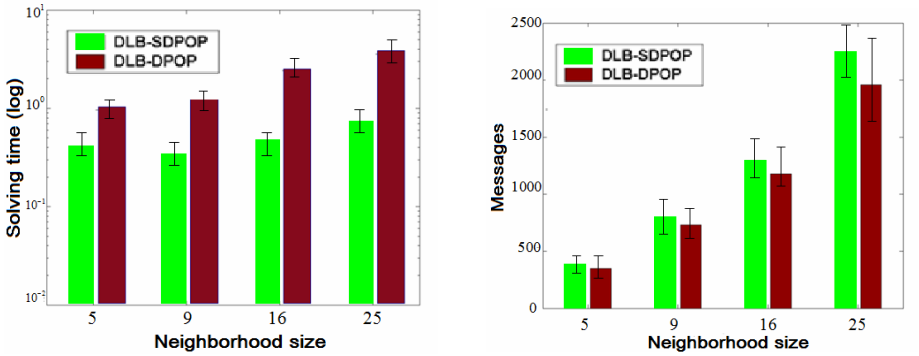
Fig. 8. *Solving time* (*log* scale) with variance for DLB-SDPOP and DLB-DPOP (*# of APs*=36, *Neighborhood size*=9, *# of MS/AP*=5), for *#-changes* to be 1,4,8 and 16.

mainly comes from the utility calculation of each MS which needs to be handed off. However, when *# of APs* increases from 16 to 81, the percentage of *#-changes* in the whole scenario becomes much smaller. The lower the percentage of failing APs in the pseudo-tree, the less time it takes for the self-stabilization algorithm to discover the fault points and recover from the inconsistent state. Meanwhile, the time saved by running self-stabilization algorithm outweighs the time overhead which is caused by the computation of handoffs. Combining these two factors, it results in a slightly decrease. *Solving time* of DLB-DPOP increases exponentially when *# of APs* increases from 9 to 81. Since DLB-DPOP does not use a self-stabilization algorithm to modify the pseudo-tree, each time a perturbation occurs, DLB-DPOP reconstructs the pseudo-tree in its DFS Traversal Phase. DLB-SDPOP significantly outperforms ($p < 0.05$) DLB-DPOP by spending 21% and 0.87% of *Solving time* of DLB-DPOP in the WLAN with 36 and 81 APs respectively.

In Fig. 8, *Solving time* increases almost linearly according to *#-changes* for both DLB-SDPOP and DLB-DPOP. In DLB-SDPOP, the time cost mainly comes from the hand-off decision making process in DLB agents (including sending and receiving communication messages among neighboring APs, utility calculation and assignment computation). In DLB-DPOP, the time cost mainly comes from the hand-off decision making process, the pseudo-tree reconstruction phase and the $UTIL$ messages and $VALUE$ messages retransmitted through the pseudo-tree to reach a new optimal assignment without the failing APs. *Solving time* is longer in DLB-DPOP because of complete pseudo-tree reconstruction and the associated $UTIL$ and $VALUE$ messages. DLB-SDPOP performs significantly ($p < 0.05$) better than DLB-DPOP on *Solving time*.

In Fig. 9(a), *Solving time* of DLB-SDPOP decreases a little when *Neighborhood size* changes from 5 to 9. If *Neighborhood size* is too small, it takes a longer time to convey the failure information to the entire pseudo-tree (each time one AP can only send messages to its neighboring APs.). The self-stabilization al-

gorithm requires that each AP have an overview of the system and coordinate with its neighbors to detect the fault. The faster it takes to convey the failure information, the faster self-stabilization takes effect. If *Neighborhood size* is too large, the self-stabilization algorithm runs fast, but each message sent may have more information, aggregated from the communication between parents and children in the pseudo-tree. When *Neighborhood size* increases to 16 and 25, the recomputed and retransmitted information in the messages results in the increase of *Solving time*. DLB-SDPOP responds to perturbations much more quickly ($p < 0.05$) than DLB-DPOP over all *Neighborhood size* in this scenario.



(a) *Solving time (log scale)* with variance (b) *Messages* with variance

Fig. 9. DLB-SDPOP vs. DLB-DPOP (*# of APs*=36, *#of MS/AP*=5, *#-changes*=8), for *Neighborhood size* to be 5,9,16 and 25.

DLB-DPOP uses fewer *Messages* than DLB-SDPOP at each *Neighborhood size* (Fig. 9(*b*)). The overhead comes from *Repair cost* of DLB-SDPOP. In the self-stabilization algorithm, failing APs send messages to their neighboring APs to notify the perturbations so that the neighboring APs can modify their relation tables and get ready for pseudo-tree repair. In real-time WLAN environments, recovering from perturbations and making hand-off decisions quickly are given higher priority. So DLB-SDPOP performs much better than DLB-DPOP with respect to both *Messages* and *Solving time* in this experiment setup.

6 Conclusion and Future Work

We have developed a multi-agent approach for decentralized load balancing in WLANs. This approach uses DLB-SDPOP, a constraint optimization algorithm to determine the optimal allocation of MSs under each AP. Empirical evaluation of DLB-SDPOP shows that the self-stabilizing mechanism efficiently handles up to 50% perturbations in real-time WLAN scenarios and it outperforms DPOP as the problem scales (up to 81 APs). However, as expected, our approach does

not perform well in tight constrained networks that have dense clustering of heavily loaded APs. The cut-off point of our experiments that DLB-SDPOP works successfully is a scenario where *# of APs*=81, *Neighborhood size*=25, *#-changes*=36, and *# of MS/AP*=30.

In the future work, we plan to make some abstractions in transmitted messages to improve our algorithm. We would send tuples instead of hypercubes in $UTIL$ messages in order to reduce the complexity from $\mathcal{O}(dom^w)$ to polynomial. Also, We plan to extend our algorithm to handle situations making the interference prediction probabilistic in keeping with real-world scenarios.

7 Acknowledgements

The authors would like to acknowledge Adrian Petcu and James Atlas for constructive suggestions.

References

1. A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan. Characterizing user behavior and network performance in a public wireless LAN. In *Proceedings of ACM SIGMETRICS*, pages 195–205, 2002.
2. H. Choxi and P. J. Modi. A distributed constraint optimization approach to wireless network optimization. In *Proceedings of AAAI07 Workshop on Configuration*, July 2007.
3. R. Dechter. *Constraint Processing*. Morgan Kaufmann, 2003.
4. E. W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.
5. S. Dolev. *Self-stabilization*. MIT Press, 2000.
6. R. Junges and A. L. C. Bazzan. Evaluating the performance of DCOP algorithms in a real world, dynamic problem. In *AAMAS (2)*, pages 599–606, 2008.
7. P. J. Modi, W. Shen, M. Tambe, and M. Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. In *AI Journal*, volume 161, pages 149–180, 2005.
8. A. Petcu and B. Faltings. DPOP: A scalable method for multiagent constraint optimization. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, pages 266–271, 2005.
9. A. Petcu and B. Faltings. S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-05)*, pages 449–454, Pittsburgh, Pennsylvania, July 2005.
10. J. Xie, I. Howitt, and A. Raja. Framework for decentralized wireless LAN resource management. In *Emerging Wireless LANs, Wireless PANs, and Wireless MANs*. Wiley, 2008.
11. W. Zhang and L. Wittenburg. Distributed breakout algorithm for distributed constraint optimization problems - DBArelax. In *Proceedings of International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, 2003.

Procedure 1 Main Functions in DLB-SDPOP

assignNum($vertex$)
$vertex.num \leftarrow counter{+}{+}$
for all X_i **do**
  if $vertex.relation[i] ==$ **true then**
    ifX_i has not been visited **then**
      $X_i.parent \leftarrow vertex$
      assignNum(X_i)
return true

assignLow($vertex$)
$vertex.low \leftarrow vertex.num$
for all X_i **do**
  if $vertex.relation[i] ==$ **true then**
    if $X_i.num > vertex.num$ **then**
      assignLow(X_i)
      $vertex.low \leftarrow \min(vertex.low,\ X_i.low)$
    elseif$vertex.parent \neq X_i$
      $vertex.low \leftarrow \min(vertex.low,\ X_i.num)$
return true

deleteEdge$\left(X_i, R_i^k\right)$
if back-edge$(i,k) ==$ **true then**
  remove R_i^k from the pseudo-tree
  for all lower agents X_j involved in R_i^k **do**
    initiate$UTIL\left(X_j, P\left(X_j\right)\right)$
if tree-edge$(i,k) ==$ **true then**
  let $X_k \leftarrow P\left(X_i\right)$
  if $\left(\forall X_l \in C\left(X_i\right), Sep(X_l) = \{X_k\}\right)$
  && $(Sep(X_i) = \{AP_k\})$**then**
    $X_i \leftarrow root$
    initiate$VALUE\left(X_i, subtree\left(X_i\right)\right)$
    initiate$UTIL\left(X_k, P\left(X_k\right)\right)$ $\leftarrow$ previous $UTILs$
  else
    let $X_m \leftarrow$ highest agent in $Sep(X_i)$
    send$VALUE\left(X_m, C\left(X_m\right) \cup PC\left(X_m\right)\right)$
    for all $X_s \in subtree\left(X_m\right)$ in right-hand side **do**
      $traversal \leftarrow UTIL_s^{C(X_s)}$
      $switchRole \leftarrow$ valueChange$(X_s, C\left(X_s\right))$

addEdge$\left(X_i, R_i^k\right)$
if X_i and X_k are in an ancestor-descendant relation
then
  connect X_i, X_k as a back-edge (Suppose X_i is
  descendant)
  initiate$UTIL\left(X_i, P\left(X_i\right) \cup PP\left(X_i\right)\right)$
if X_i and X_k are siblings
  let $X_k \leftarrow P\left(X_i\right)$
  let $X_l \leftarrow$ lowest common ancestor of X_i and X_k
  for all agents X_s on the tree-path from X_i to X_l
    $switchRole \leftarrow$ valueChange$(X_s, C\left(X_s\right))$
  $PC\left(X_l\right) \leftarrow C\left(X_l\right)$