DLB-SDPOP: A Multiagent Pseudo-tree Repair Algorithm for Load Balancing in WLANs

Shanjun Cheng, Anita Raja
Department of Software and Information Systems
The University of North Carolina at Charlotte
Charlotte, NC 28223
{scheng6, anraja}@uncc.edu

Jiang(Linda) Xie, Ivan Howitt
Department of Electrical and Computer Engineering
The University of North Carolina at Charlotte
Charlotte, NC 28223
{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 temporary failures of access points. This paper addresses the questions of how to dynamically balance the load and how to quickly respond to instability in WLANs[1]. We present a new decentralized multi-agent load balancing framework for WLANs that uses DLB-SDPOP, a distributed optimization algorithm, to dynamically load balance the WLAN. The algorithm implements a less expensive pseudo-tree repair mechanism instead of complete pseudo-tree reconstruction when instability problems occur. It also leverages an efficient communication mechanism to control communication overhead in heavily loaded situations. We empirically show that DLB-SDPOP improves WLAN load balancing performance significantly.

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]. Also, in a dynamic operational environment like a WLAN, interference may significantly impact the signal quality, and hence, impact the decision-making related to network management. The ability of the system to handle such dynamic changes and quickly move from a previous stable state to a new optimal stable state is a critical issue.

We show that the WLAN load balancing issue can be handled using a multiagent system (MAS) [2]. An agent is located inside each AP within the WLAN and interacts with agents within its neighborhood. The *neighborhood* of a certain AP is the set of those APs with whom it has frequent interactions. These interactions include sharing of data and negotiating about resource assignments. Individual agents act as coordinators and cooperate with agents in their neighborhood to take care of resource management across the WLAN. We assume that all the agents in this domain are cooperative. The dynamics in WLAN includes agent (AP) failure [3] and the movement of mobile stations (MSs).

1. This work is supported in part by the US National Science Foundation (NSF) under Grant No.CNS-0855200, CNS-0915599 and CNS-0953644.

In this paper, we map the WLAN load balancing issue to a distributed constraint optimization problem (DCOP) [4] and define a decentralized framework that leverages DLB-SDPOP, a distributed optimization algorithm with a self-stabilizing mechanism, to solve the problem.

We discuss this framework by first describing DLB-DPOP, a Dynamic Load Balancing-Distributed Pseudo-tree Optimization Procedure. We designed DLB-DPOP to be an improvement of the DPOP algorithm [5] in that it reduces the total message size by adding a communication filtering mechanism in its utility propagation phase. We then extend DLB-DPOP to Dynamic Load Balancing-Self-stabilizing Distributed Pseudo-tree Optimization Procedure (DLB-SDPOP) by augmenting it with a self stabilization mechanism. The main data structure in the DPOP family of algorithms is the pseudo-tree. 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 [5]. The key feature of DLB-SDPOP is that once it finds initial assignments to reach steady state, it uses *self-stabilizing* [6] pseudo-tree localized repair mechanisms instead of complete pseudotree reconstruction in order to load balance the WLAN dynamically. Self stabilization in distributed systems [7] is the ability of a system responding to transient failures, eventually reaching a legal state, and maintaining it afterwards. Self-stabilizing systems are particularly fault tolerant and able to cope with dynamic environments [8]. DLB-SDPOP pseudo-tree localized repair mechanism helps to maintain the current tree structure and avoid the retransmission of redundant messages. In this paper, we empirically show that DLB-SDPOP is a scalable approach and significantly improves WLAN load balancing performance in dynamic environments when compared to other state-of-the-art algorithms.

The following is an example scenario to motivate our approach. 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.

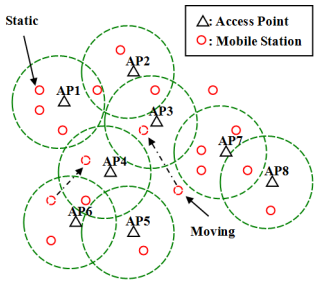


Fig. 1. 8-APs WLAN scenario.

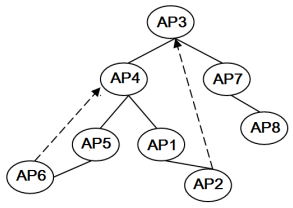


Fig. 2. Pseudo-tree for the WLAN scenario in Fig 1.

Fig 2 is the whole pseudo-tree representing the 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.

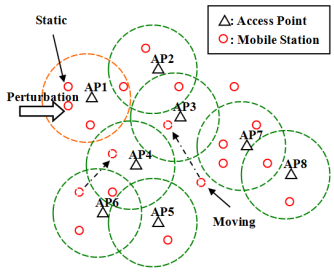


Fig. 3. Perturbation at AP_1.

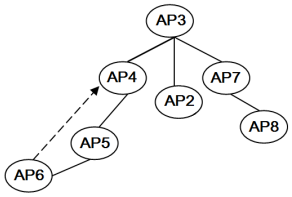


Fig. 4. Repaired pseudo-tree where $AP_1 - AP_4$ edge is deleted.

Suppose at time t_k, a new source of perturbation causes AP_1 to cease working temporarily (Fig 3). We apply our pseudo-tree localized repair algorithm as shown in Fig 4 where AP_2 and AP_1 have switched parent/child roles. This allows for AP_1 to be removed from the tree without detrimentally affecting the other nodes.

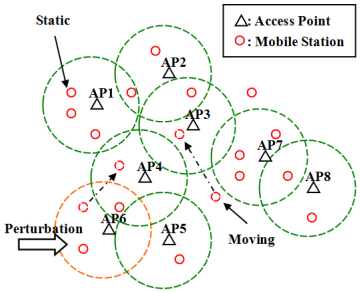


Fig. 5. Perturbation moves into AP_6.

Now suppose at time t_{k+1}, the previous perturbation moves out of AP_1 and interferes with the functionality of AP_6 (Fig 5). AP_1 is active again and AP_6 ceases working. AP_6 would be deleted resulting in Fig 6. AP_1 is then added back

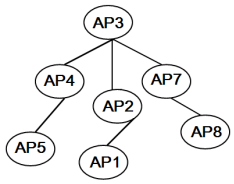


Fig. 6. Repaired Pseudo-tree (AP_6 is deleted).

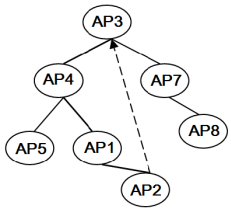


Fig. 7. Repaired Pseudo-tree (Adding back AP_1).

to the tree and the final repaired pseudo-tree is depicted in Fig 7.

The rest of the paper is organized as follows. First, we discuss the mapping of the WLAN load balancing problem as a DCOP. Our DLB-SDPOP algorithm and related work are presented and the performance results of the algorithms are discussed later, followed by the conclusions and future work.

2. WLAN load balancing

The goal of the WLAN load balancing problem is to dynamically assess the associations of 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 when required and complete handoffs by t_{k+1} (the term *time* refers to discrete time in this paper). 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 task 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. 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.

Criterion II and III may not be satisfied simultaneously. In our work, load balancing is more critical to maintain capacity

availability across the network. We give Criterion III higher priority than Criterion II.

In our 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. We map the WLAN load balancing problem to a DCOP [4] 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 a DLB 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 for the variable set X, 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 that provides a measure of the value associated with a given combination of variables. In WLAN, $\mathcal{R}$ represents the objective functions, which are the three criteria defined above for load balancing.

Our goal is to find a complete instantiation X^* for the set X 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.

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 AP's DLB agent A_i recognizes that it 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. Upon receiving a handoff event trigger, A_i initiates agent interactions within its neighborhood by sending request messages to its neighbors. The request messages contain the information about which MSs should be handed off and the deadlines of such handoffs. A_i has the pseudo-tree which is made up of its neighborhood and itself. Similarly, the neighboring agents also have partial views of the global pseudo-tree and send back response messages to announce the possible new assignments based on local evaluations of the three aforementioned criteria. The response messages contain the information about the possible time durations in which handoffs can be initiated. Each DLB agent calculates the utility values of its own assignment choices, and runs the DLB-SDPOP algorithm using the local utility values as initial inputs. After the optimal assignment is found, the handoff decisions are implemented by the appropriate target DLB agents.

3. Solution

In this section, we discuss our approach to handle the dynamics of the load balancing problem by first describing DLB-DPOP, a variation of the DPOP algorithm. It regenerates a new pseudo-tree that excludes the fault nodes every time a perturbation occurs.

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 [9]. 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 values propagated in the $UTIL$ messages of DLB-DPOP are the reward values and (b) a communication filtering mechanism is used to reduce the size of $UTIL$ messages. We define $U_i^j(t_k)$ as the normalized signal strength above the threshold of MS_i from AP_j per unit time:

$$U_i^j(t_k) = \frac{\sum_{t_s}^{t_e} P_{ratio}^{i,j}}{t_e - t_s} \quad (4)$$

where $P_{ratio}^{i,j} = P_{receive}^{i,j} - P_{thres}$, $P_{receive}^{i,j}$ denotes the received power (dBm) of MS_i from AP_j at a certain time unit, P_{thres} denotes the threshold value (If $P_{receive}^{i,j}$ 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.), $P_{ratio}^{i,j}$ measures how much $P_{receive}^{i,j}$ 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 defined as the estimated handoff cost function:

$$C_i^j(t_k) = \begin{cases} \frac{C_h}{t_e - t_s} & \text{if } t_e - t_s > t_{delay} \\ C_{max} & \text{otherwise} \end{cases} \quad (5)$$

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. 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:

$$R_i^j(t_k) = U_i^j(t_k) - C_i^j(t_k) \quad (6)$$

The reward value $R_i^j(t_k)$ is used to reduce the possibility of reaching myopic solutions. The handoff decision is viable only if the value of $R_i^j(t_k)$ is positive.

In DPOP, the child node sends its $UTIL$ message as a hypercube to its parent node. The largest message is exponential in the induced width along the particular pseudo-tree chosen [5]. Sometimes the hypercube transmitted contains a large portion of uninformative reward values (In our problem, the value of $R_i^j(t_k)$ is zero or negative). This always happens when a MS is moving away and continually loses signals from its nearby APs. This MS has few potential APs to be handed off to which results in a large number of uninformative reward values in the hypercube. Instead of sending the whole hypercube, the child agent of DLB-DPOP only sends the set of reward tuples to its parent agent that contain the positive reward values. The filtering process continues bottom-up until the root agent receives its $UTIL$ message. This reduces the total message size substantially. For example, Consider a very dense network with pseudo-tree induced width 8, neighborhood size 9 and each reward value in the hypercube uses 1 byte of storage. So the total size of the hypercube is calculated as: $(1\ byte) * 9^8 \approx 41\ MB$. In a heavily loaded situation, most APs would have reached their maximum capacities and will be incapable of handling any new MSs, meaning their information does not have to be transmitted in the hypercube. Suppose only 2 of the 9 neighbors can take on additional MSs. Each reward tuple contains 8 bytes for information about domain combinations and 1 byte for the reward value. Using the filtering strategy, the total size is calculated as: $(9\ bytes) * 2^8 \approx 2\ KB$. The filtering strategy filters huge amounts of useless information in this scenario.

Phase 3-Optimal VALUE Propagation: The optimal value assignments are then propagated top-down from the root node [5]. 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.

The domain size (dom) and induced width (w) of the pseudo-tree 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. In the worst case (All the APs are in the same neighborhood and no fault nodes exist.), the complexity converges at $\mathcal{O}(|dom^*|^{w^*})$, where dom^* =the set of all APs in the WLAN, w^* =induced width of the pseudo-tree of the whole WLAN.

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. This is sometimes wasteful, since some of the faults have limited, localized effects, that do not need to propagate through the whole problem. It is therefore desirable to maintain as much of the current DFS tree as possible.

3.2. DLB-SDPOP Algorithm

We now describe DLB-SDPOP, an extended version of DLB-DPOP, designed to respond to the dynamic failures of APs in WLANs quickly, repair the original pseudo-tree efficiently, and make the optimal hand-off decisions for load balancing in the whole networks.

Algorithm 1 DLB-SDPOP

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

Phase 1:DFS Traversal
2: $root \leftarrow electedleader$
3: assignNum($root$)
4: assignLow($root$)
5: afterwards, X_i knows $Num(X_i)$ and $Low(X_i)$.
Phase 2:Self-stabilizing Utility Propagation
6: store all new $UTIL$ messages $\left(X_i, UTIL_i^j\right)$
7: **if** any perturbation is detected in WLAN **then**
8: **case 1:** agent X_i stops working
9: **for all** constraints R_i^k of X_i in the pseudo-tree **do**
10: deleteEdge$\left(X_i, R_i^k\right)$
11: delete the single agent X_i from the pseudo-tree
12: **case 2:** agent X_i resumes working
13: connect X_i to existing agent X_j
14: send_Message$\left(X_i, UTIL_i^j\right)$
15: **for all** original constraints R_i^k of X_i **do**
16: addEdge$\left(X_i, R_i^k\right)$
Phase 3:Optimal Value Propagation
17: $X_i \leftarrow v_i^* = choose\_Optimal\ (agent\_view)$
18: send $VALUE_i^l$ to all $X_l \in C(X_i)$
END ALGORITHM

DLB-SDPOP extends DLB-DPOP by incorporating a self-stabilizing mechanism. It 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 *assign-Low*$(vertex)$ 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*). $initiateUTIL(X_j, P(X_j))$ runs the communication filtering mechanism and sends the set of reward tuples from X_j to $P(X_j)$.

In order to prove that DLB-SDPOP 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 pseudo-tree that guarantees completeness as described below. We extend the optimality proof for DPOP [5] to DLB-SDPOP. 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.

Procedure 1 Main Functions in DLB-SDPOP

1: **assignNum**$(vertex)$
2: $vertex.num \leftarrow counter$++
3: **for all** X_i **do**
4: **if** $vertex.relation[i] ==$ **true then**
5: **if**X_i has not been visited **then**
6: $X_i.parent \leftarrow vertex$
7: assignNum(X_i)
8: **return true**

9: **assignLow**$(vertex)$
10: $vertex.low \leftarrow vertex.num$
11: **for all** X_i **do**
12: **if** $vertex.relation[i] ==$ **true then**
13: **if** $X_i.num > vertex.num$ **then**
14: assignLow(X_i)
15: $vertex.low \leftarrow \min(vertex.low,\ X_i.low)$
16: **elseif**$vertex.parent \neq X_i$
17: $vertex.low \leftarrow \min(vertex.low,\ X_i.num)$
18: **return true**

19: **deleteEdge**(X_i, R_i^k)
20: **if** back-edge$(i,k) ==$ **true then**
21: remove R_i^k from the pseudo-tree
22: **for all** lower agents X_j involved in R_i^k **do**
23: initiate$UTIL(X_j, P(X_j))$
24: **if** tree-edge$(i,k) ==$ **true then**
25: **let** $X_k \leftarrow P(X_i)$
26: **if** $(\forall X_l \in C(X_i), Sep(X_l) = \{X_k\})$
27: && $(Sep(X_i) = \{AP_k\})$**then**
28: $X_i \leftarrow root$
29: initiate$VALUE(X_i, subtree(X_i))$
30: initiate$UTIL(X_k, P(X_k))$ $\leftarrow$ previous $UTILs$
31: **else**
32: **let** X_m $\leftarrow$ highest agent in $Sep(X_i)$
33: send$VALUE(X_m, C(X_m) \cup PC(X_m))$
34: **for all** X_s $\in subtree(X_m)$ in right-hand side **do**
35: $traversal \leftarrow UTIL_s^{C(X_s)}$
36: $switchRole \leftarrow$ valueChange$(X_s, C(X_s))$

37: **addEdge**(X_i, R_i^k)
38: **if** X_i and X_k are in an ancestor-descendant relation
39: **then**
40: connect X_i, X_k as a back-edge (Suppose X_i is
41: descendant)
42: initiate$UTIL(X_i, P(X_i) \cup PP(X_i))$
43: **if** X_i and X_k are siblings
44: **let** $X_k \leftarrow P(X_i)$
45: **let** $X_l \leftarrow$ lowest common ancestor of X_i and X_k
46: **for all** agents X_s on the tree-path from X_i to X_l
47: $switchRole \leftarrow$ valueChange$(X_s, C(X_s))$
48: $PC(X_l) \leftarrow C(X_l)$

This also means that each node will receive a message from its last neighbor, thus terminating the algorithm. This proves liveness. Hence, DLB-SDPOP algorithm is complete.

We now describe DLB-SDPOP's self-stabilizing pseudo-tree repair mechanism in the context of the motivating example presented in Section 1.

Given that WLAN scenario in Fig 1, Fig 2 is the whole pseudo-tree generated by Phase 1 ($line\ 2-5$, Algorithm 1) as a global view for AP_3. If we assume that a new source of perturbation causes AP_1 to cease working at time t_k as described in the example, then 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 . The algorithm first considers 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): $AP_3 \rightarrow AP_2 \rightarrow AP_1 \rightarrow AP_4 \rightarrow AP_3$. It should be noted that 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. The algorithm considers 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). The result is depicted in Fig 4.

At time t_{k+1}, the perturbation affects AP_6 and AP_1 resumes its activity (Fig 5). The algorithm deletes AP_6 from the pseudo-trees by doing the following: First, the back-edge $AP_6 - AP_4$ is removed ($line\ 20-21$, Procedure 1). Then tree-edge $AP_6 - AP_5$ is removed just as $AP_1 - AP_2$ was removed in previous scenario. AP_1 is added back to the pseudo-tree implying it is connected as a child of AP_2 ($line\ 13$, Algorithm 1). Fig 6 shows the repair results so far. AP_1 begins propagation by sending AP_2 $UTIL$ messages that considering all the influence of tree-edges and back-edges between them ($line\ 14$, Algorithm 1, in this case, only consider the tree-edge $AP_1 - AP_2$). AP_1 and AP_4 are siblings (they lie in different branches of the pseudo-tree [9]). Adding $AP_1 - AP_4$ violates the required property that agents in different branches of the pseudo-tree should be disconnected [9]. So the pseudo-tree is not valid any more and has to be repaired ($line\ 43-48$, Procedure 1). The final repaired pseudo-tree is depicted in Fig 7. AP_4 becomes AP_1's parent and AP_1 and AP_2 switch their roles.

4. Related Work

Distributed algorithms such as DSA/DBA [10], ADOPT [4] and DPOP [5] have previously been proposed to solve distributed constraint optimization (DCOP) problems. 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 [11]. Atlas [12] presented a complete mapping to DCOP for large-scale team coordination problems that offers fast convergence to high quality solutions. However, their algorithm did not address the issue of handling agent failures. Choxi and Modi [13] proposed an approach to reposition wireless routers to maximize signal strength in the network. In their work, small robots act as wireless routers and can reposition themselves. In our WLAN problem, APs are fixed and can not be repositioned. DSA and DBA do not guarantee completeness. We show in this paper that the DLB-SDPOP algorithm is complete. ADOPT requires polynomial memory, but it may produce a very large number of small messages, resulting in large communication overheads which would occupy the bandwidth used by MSs. 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.

SDPOP [8] is the first self stabilization mechanism for multi-agent combinatorial optimization. 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 [9]. SDPOP has been implemented to solve the meeting scheduling problem with up to 10% of the agents having simultaneous perturbations.

Our algorithm DLB-SDPOP is different from SDPOP in that (a) We introduce a self-stabilizing pseudo-tree repair mechanism to handle the perturbations in WLAN efficiently. (b) It can solve the complicated load balancing situations with up to 50% agents simultaneously failing. (c) It has the complexity of $\mathcal{O}(dom^w)$, where dom bounds the domain size and $w =$ *the maximum induced width* of all the localized pseudo-trees involved in the perturbations in WLAN.

5. Empirical Evaluation

All experiments in this section were performed using the 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. We assume that all the APs are cooperative. We consider the dynamics of both agent (AP) failure and movement of MSs. In the experiments, our baselines are ADOPT, DSA and DPOP, the existing state of the arts. We show the computational advantage of DLB-SDPOP over others with a low overhead.

5.1. Metrics for Evaluation

We evaluate the performance of ADOPT, DSA, DPOP and DLB-SDPOP 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.
- *# of MS/AP* - the average number of MSs associated with each AP in the scenario.

We compare the algorithms by measuring the following metrics when perturbation occurs: *DLB-Value*, *Messages*, *Message size* and *Solving time*.*DLB-Value* measures the **Criterion III** in Section 2. The lower *DLB-Value* is, the better the load balancing performance is. *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. *Message size* is defined as the total amount of information bytes exchanged among DLB agents to solve a problem scenario. *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.

We make the assumptions in our experiments. When perturbation occurs (DLB agents stop working or previous fault DLB agents resume functionalities), the DFS trees in ADOPT, DSA and DPOP which represent the communication topology among DLB agents would be regenerated according to the changes. In DSA, the probability p [10] which controls how frequently neighboring agents changing values is set to 0.6 (If p is too high, the phase transition occurs that would decrease the performance sharply).

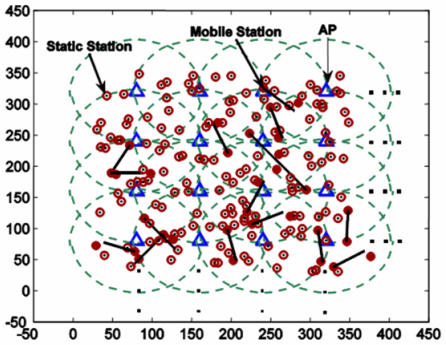


Fig. 8. Simulation Scenario.

# of APs	Algorithm	*Solving time*(sec.)	*Messages*	*Message size*
9	ADOPT	0.480 ± 0.016	306 ± 57	$16.5\ K$
	DSA	0.037 ± 0.023	129 ± 20	$3.3\ K$
	DPOP	0.053 ± 0.008	38 ± 5	$2.4\ K$
	DLB-SDPOP	0.024 ± 0.006	72 ± 9	$1.7\ K$
16	ADOPT	1.341 ± 0.302	2393 ± 430	$1665.9\ K$
	DSA	0.984 ± 0.153	473 ± 87	$123.6\ K$
	DPOP	0.109 ± 0.012	96 ± 27	$366.2\ K$
	DLB-SDPOP	0.028 ± 0.009	147 ± 41	$58.0\ K$
36	ADOPT	14.456 ± 2.482	9531 ± 967	$3.5\ M$
	DSA	2.846 ± 0.634	2420 ± 592	$2.8\ M$
	DPOP	0.458 ± 0.037	162 ± 42	$8.3\ M$
	DLB-SDPOP	0.041 ± 0.013	253 ± 72	$248.9\ K$
81	ADOPT	273.833 ± 58.484	386246 ± 4958	$611.0\ M$
	DSA	19.473 ± 3.298	49857 ± 6447	$93.8\ M$
	DPOP	8.522 ± 0.122	384 ± 83	$294.4\ M$
	DLB-SDPOP	0.849 ± 0.105	540 ± 127	$1.8\ M$

TABLE 1. DLB-SDPOP $vs.$ other DCOP algorithms (*Neighborhood size*$= 5$, *# of MS/AP*$= 5$, *#-changes*$= 1$)

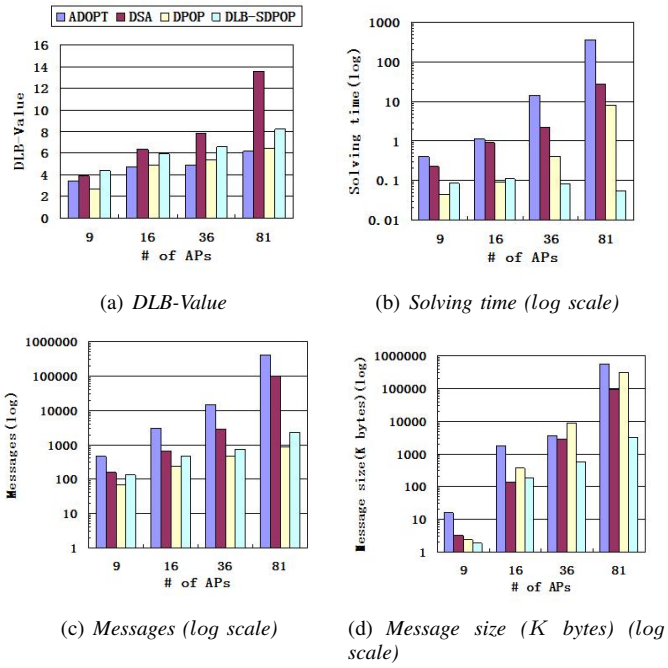
5.2. Scenario Setup

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 $P_R(d_{ij}(t_k)) = P_T - (20\log_{10} f_c + 10\kappa \log_{10}(d_{ij}(t_k)) - 28)(dBm)$ [14]. In our simulation, 10% of MSs belonging to 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 all the algorithms (We set the perturbation areas in the scenario randomly to influence the functionalities of APs). $C_h = 30dBm$ and $C_{max} = 25dBm$. Fig 8 shows the simulation scenario.

5.3. Discussion

Table 1 provides the performance comparison of the algorithms. ADOPT spends substantial time and messages to reach a solution. This is mainly because ADOPT is more susceptible to the variations in the construction of the constraints. DLB-SDPOP performs significantly better than others on *Solving time* and *Message size* (compared with DPOP, the p values from t-tests are 0.00478, 0.00359, 0.0370 and 0.000846 on *Solving time* and 0.0221, 0.00351, 0.00613 and 0.000289 on *Message size*). This shows the effectiveness of pseudo-tree repair and communication filtering in DLB-SDPOP. DLB-SDPOP consumes more *Messages* only 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.

In Fig 9, DLB-SDPOP performs a little worse than ADOPT and DPOP on load balancing, but much better than DSA when *# of APs* increases to 36 and 81. However, DLB-SDPOP significantly outperforms ($p < 0.05$) the others on *Solving time* on most cases (DPOP spends less time than DLB-SDPOP on small-scale scenarios ($\#ofAPs$ = 9 and 16)). *Solving time* of DLB-SDPOP decreases when *# of APs* increases to 36 and 81. This is mainly because the percentage of *#-changes* in the whole scenario becomes much smaller which helps the self-stabilizing algorithm to discover the fault points and recover from the inconsistent state. DLB-SDPOP significantly outperforms ($p < 0.05$) ADOPT and DSA on *Messages*. DPOP uses fewer messages than DLB-SDPOP. The increase in

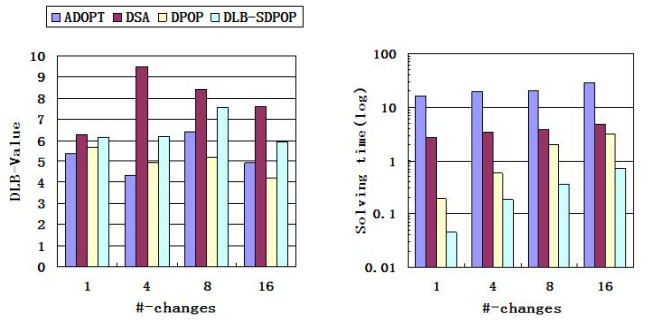


(a) *DLB-Value* (b) *Solving time (log scale)*

(c) *Messages (log scale)* (d) *Message size (K bytes) (log scale)*

Fig. 9. DLB-SDPOP vs. other DCOP algorithms (*Neighborhood size*=5, *#of MS/AP*=5, *#-changes*=4), for *# of APs* to be 9,16, 36 and 81.

messages comes from the pseudo-tree repair process of DLB-SDPOP. DLB-SDPOP consumes substantially ($p < 0.05$) less *Message size* than the others when the scenario scales up.



(a) *DLB-Value* (b) *Solving time (log scale)*

Fig. 10. DLB-SDPOP vs. other DCOP algorithms (*# of APs*=36, *Neighborhood size*=9, *# of MS/AP*=5), for *#-changes* to be 1,4, 8 and 16.

When the *#-changes* parameter is tweaked as in Fig 10, DLB-SDPOP takes extremely short time (compared with other algorithms) in the price of not very large decrease of load balancing. In real-time WLAN environments, recovering from perturbations and making handoff decisions quickly are given higher priority. So DLB-SDPOP outperforms the other algorithms in response to dynamism in WLAN environments.

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. DLB-SDPOP dynamically repairs the affected nodes in the original pseudo-tree retaining the topology and states of unaffected nodes when inconsistency is detected. Empirical evaluation of DLB-SDPOP shows that the self-stabilizing, scalable mechanism leverages pseudo-tree localized repair efficiently and handles up to 50% perturbations in real-time WLAN scenarios. It outperforms ADOPT, DSA and DPOP as the problem scales (up to 81 APs). The experiments also show that the communication filtering mechanism improves communication efficiency especially in heavily loaded scenarios. As future work, we plan to extend our algorithm to handle situations where the interference prediction is probabilistic in keeping with real-world scenarios.

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*, 2002, pp. 195–205.

[2] S. Cheng, A. Raja, L. Xie, and I. Howitt, "A distributed constraint optimization algorithm for dynamic load balancing in WLANs," in *Proceedings of Eleventh International Workshop on Distributed Constraint Reasoning to be held in conjunction with IJCAI 2009*, pp. 31–45.

[3] M. N. Sahoo, P. M. Khilar, and B. Majhi, "A redundant neighborhood approach to tolerate access point failure in IEEE 802.11 WLAN," in *Fourth International Conference on Industrial & Information Systems*, December 2009.

[4] P. J. Modi, W. Shen, M. Tambe, and M. Yokoo, "ADOPT: Asynchronous distributed constraint optimization with quality guarantees," in *AI Journal*, vol. 161, 2005, pp. 149–180.

[5] A. Petcu and B. Faltings, "DPOP: A scalable method for multiagent constraint optimization," in *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 266–271.

[6] S. Dolev, *Self-stabilization*. MIT Press, 2000.

[7] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Commun. ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[8] A. Petcu and B. Faltings, "S-DPOP: Superstabilizing, fault-containing multiagent combinatorial optimization," in *Proceedings of the National Conference on Artificial Intelligence (AAAI-05)*, Pittsburgh, Pennsylvania, July 2005, pp. 449–454.

[9] R. Dechter, "Constraint processing." Morgan Kaufmann, 2003.

[10] W. Zhang and Z. Xing, "Distributed breakout vs. distributed stochastic: A comparative evaluation on scan scheduling," in *Proceedings of AAMAS-02 Workshop on Distributed Constraint Reasoning*, Bologna, Italy, 2002, pp. 192–201.

[11] R. Junges and A. L. C. Bazzan, "Evaluating the performance of DCOP algorithms in a real world, dynamic problem," in *AAMAS (2)*, 2008, pp. 599–606.

[12] J. Atlas and K. Decker, "Coordination of agent schedules using distributed neighbor exchange," in *Proceedings of International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, Toronto, Canada, May 2010.

[13] H. Choxi and P. J. Modi, "A distributed constraint optimization approach to wireless network optimization," in *Proceedings of AAAI07 Workshop on Configuration*, July 2007.

[14] 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.