A DISTRIBUTED SYSTEM FOR EASY CLIENT ASSOCIATION IN WIRELESS NETWORKS

Mihai BUCICOIU[1], Victor ŞERBU[2], Nicolae ŢĂPUŞ[3]

As wireless networks evolve more and more driven by the low-priced wireless devices, the load on those networks raises. A challenge for these networks is enabling a way for fast scaling and easy client control. Motivated by this problem, we present a self-aware network, capable of deciding how to accept a new user. Our approach uses self-discovery for the access points. Furthermore, it enforces distributed decision making in order to associate a new client. Our implementation requires less than 53 lines of application-specific code changes in the current wireless implementation and also, a "thin" service, running as a daemon that allows access point communication. We demonstrate the effectiveness of our approach using ns-3, a discrete-event network simulator for Internet systems, targeted primarily for research and educational use.

Keywords: wireless, client association, access point, distributed systems

1. Introduction

The rapid growth of the mobile devices made today's wireless access a fight. Yesterday's wireless technology could offer access for 54 clients, we assume 1Mbps per client and 54Mbps access points even though this is highly unlikely, while today's mobile technology brings us to more than 300 clients. Increasing the speed of access points from 54Mbps, 802.11g, up to 300Mbps, 802.11n, enabled a 6x increase in the number of users, and with a more efficient usage of the wireless spectrum we can add 3x speed-ups by using three access points instead of one for the same location. This sums up to an ideal speed-up of 18x, but the problem of client association still remains.

Consider a simple example that can explain why client distribution is so important. We have an open space with 200 possible clients and three access points, each running on a different wireless channel and offering the same bandwidth. The decision on the association is at the client side and everyone can decide to go with the same access point, living the other two unused. One can

[1] PhD Assist., Dept.of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: mihai.bucicoiu@cs.pub.ro
[2] MSc., Dept.of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: victor.serbu@cti.pub.ro
[3] Prof., Dept.of Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: nicolae.tapus@cti.pub.ro

assume that we have a "smart" client that can listen to the channel before trying to associate, but in wireless the hidden node problem would make the implementation of such a smart client nearly impossible without communication with the access point. In this paper we present a "smart" algorithm implemented on the access point side that allows easy association and fast client migration.

In this paper we introduce a new way of managing clients in a wireless network, allowing the access points to dynamically decide where a client should go and, when necessary, migration can also be achieved.

The rest of the paper is structured as follows. In Section II we introduce related work. Section III covers the design of our solution, including the algorithms used for balancing and for computing the metrics. Section IV describes how we've implemented the proposed solution both in a simulated and real environment. In section V we present the results of our implementation, while section VI concludes the paper.

2. Related work

In order to improve the performance of wireless LAN deployment, many approaches have been proposed to allow a better client association. Different algorithms try to allocate channel frequency and power level to the access points in an intelligent manner so as to allow a better coverage. Also, these algorithms control the association of clients to the access points in order to avoid major differences between loads on each access point. Broustis quantifies the effect of individual optimization and combinations of these optimizations [1].

The intelligent association of clients with access points can be obtained by adding the following fields to beacon frames: current number of clients associated, the mean RSSI of signal received by access point from its clients and the RSSI of Probe Request frame received from a client scanning the medium [2]. The main issue with this approach is that the clients must be modified in order to take in consideration these additional fields. We consider this approach difficult to implement on production devices as they need to modify the structure of the beacon and change the implementation of both clients and access points.

Another solution that does not require modifying neither the standardized protocols nor the clients is presented by Rohan [3]. Each access point communicates with a Central Controller when a new client appears in the network, including in messages the RSSI of received Probe Request frame. The access points must also send periodic messages that contain traffic pattern of associated clients. All decisions are taken by Central Controller which further instructs the access point to permit/deny association of a client or to move an associated client.

Our key contribution is developing a distributed solution that does not require a centralized controller. Also, we propose a different methodology to select the access point for initial client association and a methodology to select clients that need to be moved from an overloaded one to a less utilized one. Our application can be extended so that access points can act together upon some network attacks such as distributed denial of service [4].

3. System design

Naming convention. For convenience we will use the following notations throughout this paper.

- $AP = \{a_1, a_2 \dots a_n\}$, the list of access points (APs) in the network;
- $Ca_k = \{c_1, c_2 \dots c_n\}$, all the clients visible to access point a_k;
- $RL(c_i) = \{Ra_1(c_i), Ra_2(c_i) \dots Ra_n(c_i)\}$, the Received Signal Strength Indication (RSSI) for all the possible access points that c_i can connect to, where $Ra_j(c_i)$ is the RSSI calculated by the access point a_j for client c_i;
- $MKa_j(c_i) = \{Ma_1(c_i), Ma_2(c_i) \dots Ma_n(c_i)\}$, the list of metrics, for all the access points in the network, as known by access point a_j for client c_i; we will explain how this metric is computed in Section III.

We note LOAD(aj) the load for access point aj, and NO_ACTIVE_CLIENTS the number of ACTIVE CLIENTS as calculated in Section III.B.

In order to make our system work we assume that all the access points are in the same network, with multicast messages not being filtered by intermediate devices. This assumption is made only for an easier explanation and implementation of the process. We also believe that this is the case in most of the situation where the wireless infrastructure is under the same management, creating a single Virtual Local Area Network for wireless would be the simplest choice when deploying wireless. If this is not the case, multicast routing protocols, such as PIM [5], could be used to deliver multicast.

The connection to the network is made through the LAN interface for every access point. Because we do not use layer 3 roaming for our scenario, we could also make the connection through the WAN port and allow the access point to play the role of a router, as it happens in many situations.

Neighbour discovery. The first step in creating a distributed system is discovering all its members. Our application accomplishes this by sending hello messages to a predefined multicast address, 224.0.1.50, similar to what is used in different routing protocols [6]. This approach is convenient as multicast traffic will not take up bandwidth to deliver the same message to more than one client.

We also define a hold-down timer in case an access point losses access to the infrastructure.

The hello messages are sent every two seconds, containing the following fields: ID, LOAD, NO_ACTIVE_CLIENTS. The ID is the IP address of the access point. Using this mechanism, the AP list is built. Although there might be a slight difference in the list contained by each access point, e.g. when a new access point is added, we advocate that the network will converge after few seconds. In the initial discovery process the access point will not allow any client association for the first six seconds.

Metric computation. A linear dependency exists between the value of RSSI and the bandwidth [7]. From experimental results, the following formula can be used to deduct the bandwidth available for a client:

$$RSSI(c_i) = 1.0 * B_i - 86 \quad (1)$$

In order to test our algorithm on a large scale wireless infrastructure, we used a network simulator, ns-3. We have verified if the preceding formula also applies in ns-3. For this, we created a topology of 30 clients and one access point located on a grid, 31 lines and 1 column, with a distance of 2. In this topology, RSSI for frames received by the access point is situated between -36 and -82, being constant for each of the clients. In the simulation we've used 30 non-concurrent TCP flows between access point and each client.

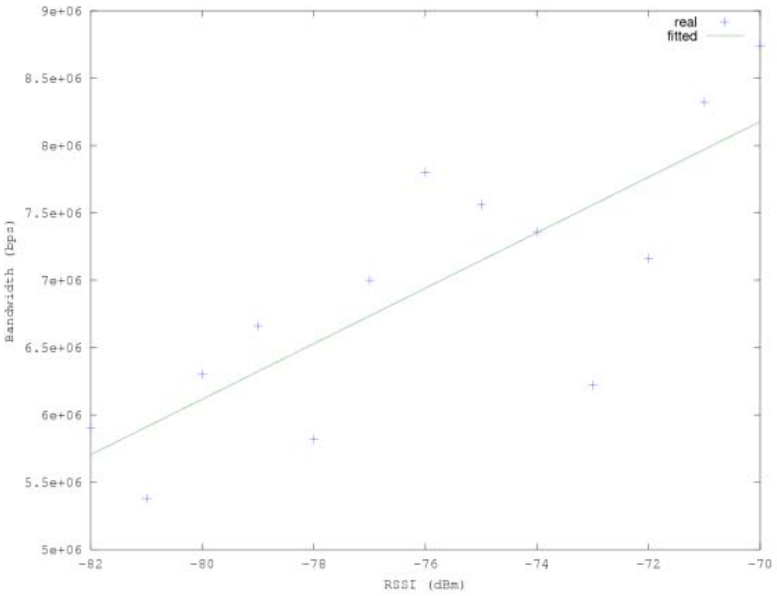

Fig. 1. Determine an equation between RSSI and Bandwidth

From our tests, Fig. 1, we've observed that for $RSSI(c_i) > -70$,the bandwidth remained constant at approximately 8.48 Mbps. For $RSSI(c_i) \leq -70$ we've determined the following polynomial equation of degree one using Matlab *polyfit* function:

$$B_i = 0.19 * \text{RSSI}(c_i) + 21.55 \quad (2)$$

We propose the following equation to determine the load generated by each client associated.

$$LOAD_{CLIENT}(c_i) = 100 * \frac{D_i}{B_i} * \frac{1}{NR\_SEC} \quad (3)$$

D_i represents the amount of data transferred to and from client c_i in NR_SEC seconds and B_i is the estimated bandwidth available to c_i. Intuitively, the value of LOAD shows the percentage of *"air time"* used by client c_i. For each access point we compute the load as a sum of all clients load.

In the preceding simulation and observed that if the RSSI of a client is greater the -70, the load it generates is constant at 104, while for clients with RSSI between -70 and -82 the load varies between 63 and 104, with an average of 80.22. We've also taken into consideration how an access point balances the channel between different clients. In order to test this we've created a small topology with three clients having the RSSI of -73, -75, and, respectively, -77 and joining the access point after 0, 6 and, respectively, 13 seconds. We've observed an almost perfect distribution of channel, Fig. 2, each client receiving equal number of time slots.

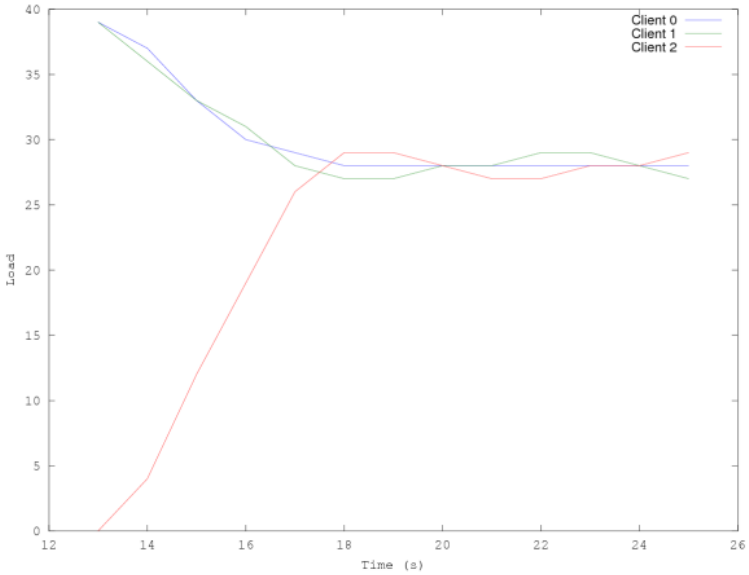

Fig. 2. Evolution of clients load

We note as *ACTIVE CLIENTS* the clients associated with the access point that use the channel. Let $MAX_{LOAD}(a_j)$ be the maximum load generated by one of the associated clients. We considered a client c_i as *ACTIVE CLIENTS* if:

$$LOAD_{CLIENT}(c_i) \geq 60 * \frac{MAX_{LOAD}(a_j)}{100} \quad (4)$$

Let $NO\_ACTIVE\_CLIENTS(a_j)$ be the number of *ACTIVE CLIENTS* associated with access point a_j.

In order to determine the access point to which a client should be connected to, a metric is required. This metric must consider both the signal strength and also the number of clients competing for medium. Our metric estimates the maximum amount of data that a new client will transfer in a second. We suggest the following formula:

$$Ma_j(c_i) = B_i(a_j) * \frac{0.6}{NO\_ACTIVE\_CLIENTS(a_j) + 1} \quad (5)$$

A new client will become an active client and active clients will occupy the medium for at least 60% of total time. Estimated bandwidth $B_i(a_j)$ is computed based on the average RSSI value of the *Probe Requests* frames sent before association by the client, as received by the access point.

If a forced computation is required, see 0, we have set a minimum number of Probe Request frames that an access point must have received in order to calculate the metric. Let this constant be called MIN_PR. We advocate that if an access point receives enough frames and the second one receives less than this minimum, the second access point should not be used for association as the speed will be very low and the metric is equal with zero. All the access points will respond with a metric of 0 for the clients that are not in their range.

Initial client association. The most important decision is where to associate a new client. For this, the access points must be able to "talk" between them and decide who should accept the client. Listing 1 exposes the algorithm used to achieve this for access point a_k and client c_i.

```
Ca_k = Ca_k + {c_i}
compute Ma_k(c_i)
LA = {a_k} #list of the access point interrogated
RL(c_i) = {Ra_k(c_i)} #RSSI for all APs of this client
MKa_k(c_i) = {Ma_k(c_i)} #list of metrics discovered
do for random (a_j)in AP − LA:
 send Ma_k(c_i)
 get Ma_j(c_i)
 MKa_k(c_i) = MKa_k(c_i) + {Ma_j(c_i)}
 LA = LA + {a_j}
 if (Ma_j(c_i) > 0):
  Send MKa_k(c_i)
  Receive MKa_j(c_i)
  MKa_k(c_i) = MKa_k(c_i) + MKa_j(c_i)
  foreach a_x in MKa_j(c_i)
   LA = LA + {a_x}
   if Ma_x(c_i) ! = INF
    RL(c_i) = RL(c_i) + Ra_x(c_i)
while (#LA ! = n)#number of elements in LA
if Ma_k(c_i) = MAX(MKa_k(c_i)) then
 associate client c_i
```

Listing 1. Client association decision process

The first step taken when a new client arrives in a network is to compute a metric as discussed in Section III by all the access points where the client can associate. The goal is to compute $MKa_k(c_i)$ and based on this to determine which access point is preferred.

For each access point in the network a metric for client c_i needs to be computed. Those access points that don't "see" the client will compute the 0 metric. This process is initiated by an access point that received a number of *Probe Request* frame bigger than a constant named MAX_PR. After these access points have calculated the metric, they will start to announce their metric and request the neighbors to respond with their metric.

If an access point receives a request for a client that it cannot "see" (received a number of *Probe Request* less than MIN_PR or does not have access to the client), he will respond with a metric of 0. If he has already computed the metric for this client, an exchange of the MK lists will takes place. When the access point hasn't calculated the metric yet, a forced computation will take place. After exchanging the MK list with the initiator of exchange, this access point will also begin to announce it's metric.

For better performance we interrogate a random neighbor and receive his MK list. Using this approach we ensure a fast process to compute MK by using the results from the rest of the prospective access points.

Client relocation. Load balancing is a must in wireless networks. Previous work use a central controller that tells the access points how to associate

or migrate its clients [8]. In order to allow our network to adjust itself in case of a network reconfiguration or overload, we have developed a mechanism that allows client migration. Each access point monitors its local LOAD and if it is greater than THRESHOLD_LOAD, a reconfiguration algorithm will be triggered.

Provided the threshold is reached, the access point will try to relocate one of its *active clients*. We impose the limit that a maximum of one client will be moved in a round of this load-balancing process. Load balancing process will try to move it to another access point in order to obtain the best bandwidth after association.

The migration process is a two-step process. Firstly, the access point must determine if the relocation is possible based on RL and the expected bandwidth the client will reach after association. Secondly, it will send a message to the selected access point to inform him to accept this client and then disassociate the client. The clients are identified based on their layer 2 addressing, the MAC address.Let a_k be the current access point, a_j the access point that can "take" c_i, the client to be moved. A client will be moved only if:

$$Ra_j(c_i) < INF \tag{6}$$

$$Ma_k\ (c_i) < Ma_j(c_i) \tag{7}$$

$$CURRENT_{TIME} - TIME_{last\_client\_moved} > NR\_SEC \tag{8}$$

The first rule is required in order to consider only access points that are in the range of that client. The second rule tries to find one where the client can obtain a higher throughput than the current throughput by comparing metrics. The third rule states that an access point cannot move two clients to a new one in an interval within less than NR_SEC. This rule takes in account that the load of new access point might be inaccurate because the load generated of new clients is determined for a period of NR_SEC seconds.

4. Design implementation

802.11 devices today tend to be SoftMAC wireless devices [9]. These devices allow for a finer control of the hardware, allowing for 802.11 frame management to be done in software for them, for both parsing and generation of 802.11 wireless frames. The drivers for these devices use mac80211 framework [10]. In mac80211 Media Access Control (MAC) Sublayer Management Entity (MLME) events (e.g. Authentication, Association) are handled in user space for wireless network devices that are in access point mode.

In order to function in access point mode, a daemon that implements IEEE 802.11 access point management must run on the host. One of these daemons is *hostapd*, [11], which supports Linux drivers, such as *Host Access Point, MadWifi*,

Prism54 and some of the drivers which use the kernel's *mac80211* subsystem. This daemon uses *netlink* (the nl80211 driver) to create a master mode interface for traffic and a monitor mode interface for receiving and transmitting management frames.

Our target is to provide a helper for *hostapd* in order to improve the association process to avoid the situation when one access point is overloaded and others are free. For this, we created a new daemon that does all the calculation concerning load balancing and initial association. This daemon writes in a file the MAC addresses of clients that *hostapd* should permit association. When a modification occurs on that file, our daemon will send SIGUSR2 signal to *hostapd* that will re-read those MAC addresses in memory. The only modification in *hostapd* daemon was rewriting a function that handles the SIGUSR2 signal and a minor modification of function *handle_probe_req*: before responding to a PROBE REQUEST, *hostapd* must verify if the sender's MAC is on the permitted list of MAC addresses.

In order to decide which access point will accept a new client, all access points must obtain the average RSSI from *Probe Request* frames sent by client. This information can be obtained by sniffing on an interface working in monitor mode. After configuring monitor interface, all the management frames can be captured. Each frame will contain an additional header called *radiotap*. This header is a mechanism to supply information about frames from the driver to user-space applications such as *libpcap* and contains, among other information, the value of RSSI and the channel frequency.

By default, the monitoring interface created by *hostapd* is called *mon.*wlan0. We discovered that this interface does not add *radiotap* header to captured frames. In order to solved this problem we changed the flags of the monitoring interface using following commands: "*iw mon.wlan0 set monitor none*".

Our daemon uses *libpcap* to read frames from the monitoring interface. The structure of this daemon is: (1) Open the capture interface; (2) Set the following capture filter in order to receive only *Probe Request* frames: "*type mgt subtype probe-req*"; (3) Register a callback function that is called every time a frame is captured.

To calculate the LOAD on an access point we must determine the amount of data transferred by each client associated with it. We used the *iw station dump* command and then we parsed the output in order to obtain RX/TX bytes transferred by each client. To disassociate a client use the *hostapd_cli* command: "*hostapd_cli –p hostapd disassociate $client_mac*".

In order to determine how many packets are lost when a client is migrated from one access point to another, we started an *icmp* flood using command ping and then force client to migrate. We observed that on average 35 packets are

dropped and this corresponds to approximately 0.01314 seconds of failure (the time difference between two *icmp echo* request packets is approximately of 0.000365 seconds).

5. Simulation

For testing our algorithm we use two different scenarios. One that involves three real users, which proofs that our solution can be implemented only with little modification on the access point, and the other using the ns-3 simulator in order to have a bigger number of clients.

Real scenario. We've implemented the access points, noted here as AP1 and AP0, using two Cisco wireless PCI cards with AR5212/AR5213 Atheros chipset. One instance of hostapd is configured to use channel 1 and the other one to use channel 6. Also, we started two instances of our helper and each instance communicates with its corresponding hostapd instance.

Our test bed included 3 wireless clients: two notebooks (noted here as C1 and C2) and one smartphone (C3). The RSSI for the same client observed by both access points was approximately equal, with C1 having a RSSI of -59, C2 of -55 and C3 of -62. After association, each client started to transfer a 1GB file from an http server (running in the local network).

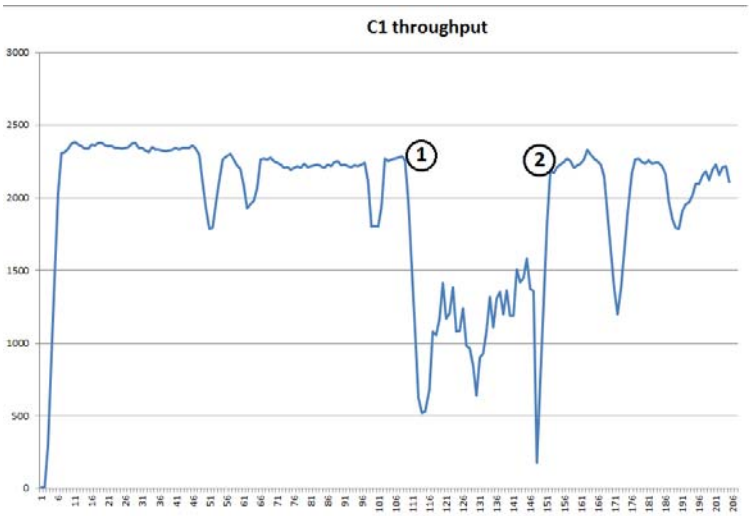

Fig. 3. C1 throughput

First, we started the client C1 and associate it with AP1. After 30 seconds, C2 was started and associated with AP0. After another 60 seconds we started the client C3 which associated with AP1 (marked as 1 in Fig. 3). At this point the

average throughput of C1 dropped from 2261 KB/s to 1140 KB/s and the average throughput of client C3 was 1139KB/s. Last, we've stopped the transfer on C2 and after two seconds client C1 was migrated from AP1 to AP0. At this point the average throughput of C1 increased to 2237 KB/s (marked as 2 in 3). The throughput of C1 can be seen in Fig. 3.

Simulation scenario. The topology of first simulation is composed of five clients and two access points, Fig. 4. The corresponding values for RSSI of frames received by AP0, respectively by AP1, can be observed in *Table **1***; these values were read from *pcap* trace generated by ns3 for each access point.

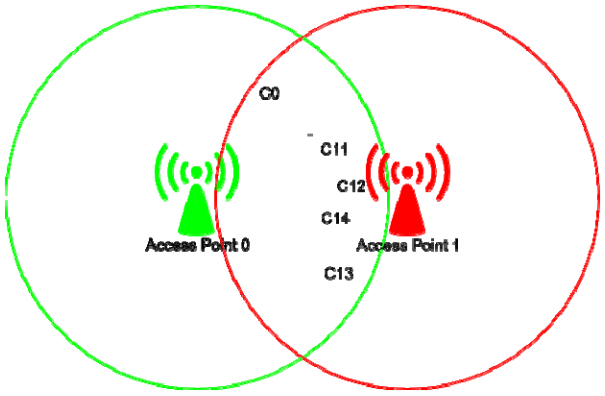

Fig. 4. Simulation topology 1

Table 1.

RSSI values for simulation 1

Access Point	CLIENT				
	C0	**C11**	**C12**	**C13**	**C14**
AP0	-74	-74	-73	-74	-79
AP1	-85	-68	-64	-68	-64

The simulation consists of client association; download some data and disassociation. The time difference between associations of two consecutive clients is seven seconds and the clients "appear" in the following order: C14, C0, C11, C12, and C13. After one second from the moment of associating with an access point, clients start downloading the amount of data depicted in *Table **2***. After two seconds from the moment when each client have finished downloading, that client will disassociate.

Table 2.

Amount of data to downloaded in simulation 1

	C0	C11	C12	C13	C14
Amount(MB)	38.14	228.88	38.14	76.29	38.14

This simulation was repeated three times in order to show how time necessary to finish downloading the same amount of data differs in the following situations: (A) Normal client association (based on RSSI only); (B) Proposed association algorithm; (C) Proposed association algorithm + load balancing. The results can be observed in *Table 3.*

Table 3.

Donwload time simulation 1

Client		C0	C11	C12	C13	C14
Download start time		14	21	28	35	7
Download finish time	Case A	53	352	130	236	80
	Case B	83	317	101	186	74
	Case C	83	281	101	145	74
Download time	Case A	39	331	102	201	73
	Case B	69	296	73	151	67
	Case C	69	260	73	110	67

The sum of client's duration time necessary to download data was in case A, 746 seconds, in case B, 656 seconds and in case C, 579 seconds. It can be observed that our proposed solution has the effect of lowering the total download time with 22% in comparison with the scenario where association decision is took based only on RSSI.

For the first case, four clients were associated with AP1, leaving client C0 alone with AP0. With our association algorithm this was significantly improved, clients C11, C13 and C13 were associated with AP1, while C0 and C12 with AP0.

When including also the load balancing mechanism, case C, after 102 seconds, C2 associated finished downloading, client 11 was moved from AP1 to AP0 and his throughput increased from 567KB/s to 998 KB/s. After another 44 seconds, C13 finished downloading, C11 was moved again from AP0 to AP1 and the speed increased now to 1126KB/s.

Table 4.

Simulation topology 2

C33	C34	AP4			
C27	C28	C29	C30	C31	C32
C21	C22	AP3	C24	C25	C26
C16	C17	C18	C19	AP2	C20
AP1	C11	C12	C13	C14	C15
C5	C6	C7	C8	C9	C10
C0	C1	AP0	C2	C3	C4

Table 5.

Download time in simulation 2

Case	Total download Time
A	1721
B	1443
C	1342

For the second simulation we built a topology composed of 36 clients and 5 access points placed on a grid with 6 columns and 7 rows. We have chosen this topology to observe how our algorithm performs in a situation where the clients

are distributed uniformly and are not concentrated nearby one of the access points. The topology can be observed in Table 4. The clients associated in random order at interval of two seconds. The amount of data downloaded by each client was chosen randomly from five values, see Table 5.

Table 6.

Amount of data to download simulation 2

Amount of Data(MB)	Clients
2	6, 9, 16, 19, 22, 24, 26, 35
4	7, 23, 29, 34
8	0, 1, 3, 12, 13, 15, 25, 30, 31
26	4, 8, 10, 11, 17, 18, 20, 27, 28, 32
32	2, 5, 14, 21, 33

We observed that in case B 17 clients (*7, 8, 9, 12, 13, 14, 17, 18, 20, 23, 24, 25, 26, 29, 30, 33, 34*) have chosen a different access point comparing with case A where the association takes in consideration only the value of the RSSI. The sum of clients duration time necessary to download data in the three cases analysed is shown in Table 5. Analysing case C, we observed that in our simulation appeared 45 events where an access point moved a client to another one, a total of 20 clients were moved at least once and the maximum number of times a client have been migrated was equal with six.

6. Conclusion

This paper presents a new approach for client association in large wireless networks by creating a self-aware distributed system. We demonstrated that involving each access point into the decision process of client association could be beneficial for the entire network in terms of resource utilization. Our key contributions are the algorithm used for load balancing the metric computation for each client and the self-aware system. Our application can be easily integrated with today's access point and needs no modifications on the client side.

We plan to extend the developed prototype and include some security features like client filtering, malicious client migration and secure channel between access points. Moreover, a real-world simulation using our prototype will give us more insights on how to improve the proposed algorithms.

REFERENCES

[1] *Ioannis Broustis, Ioannis Broustis, Srikanth V., Michalis Faloutsos, and Vivek P. Mhatre,* "MDG: Measurement-Driven Guidelines for 802.11WLAN Design," IEEE/ACM Transactions on Networking, vol. 18, no. 3, pp. 722-735, June 2010.

[2] *Paramvir (Victor) Bahl et al.*, "A study on dynamic load balance for IEEE 802.11b wireless LAN," IEEE Transactions on Mobile Computing, vol. 6, no. 2, pp. 164-178 , 2007.

[3] *Rohan Murty, Jitendra Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill*, "Designing High Performance Enterprise Wi-FI Networks," in 5th USENIX Symposium on Networked Systems Design and Implementation, 2008, pp. 73-88.
[4] *Laura Gheorge, Razvan Rughinis*, "Storm Control Mechanism in Wireless Sensor Networks," in Roedunet International Conference (RoEduNet), Sibiu, 2010.
[5] RFC4601
[6] RFC2328
[7] *Yuto NAKATSU, Tomotsugu HASEGAWA, and Manabu OMIYA, Irokazu TAKENO*, "Throughput Measurements and Numerical Prediction Methods for IEEE802.11n Wireless LAN Installations at 2.4 GHz in a Residential Two-Story House," in International Symposium on Antennas and Propagation (ISAP 2011), Jeju, Korea, 2011, pp. 1-4.
[8] Aggressive Load Balancing on WLAN, http://www.cisco.com/image/gif/paws/107457/load_balancing_wlc.pdf (February 2014)
[9] Linux Wireless, http://wireless.kernel.org/en/developers/Documentation/Glossary#SoftMAC (October 2013)
[10] MAC 802.11, http://wireless.kernel.org/en/developers/Documentation/mac80211 (January 2014)
[11] hostapd IEEE 802.11, http://hostap.epitest.fi/hostapd/ (October 2013)