Make-Before-Break MAC Layer Handoff in 802.11 Wireless Networks

Kishore Ramachandran
WINLAB
Rutgers University, New Brunswick, NJ.
Email: kishore@winlab.rutgers.edu

Sampath Rangarajan John C. Lin
Center for Networking Research
Bell Laboratories, Holmdel, NJ.
Email: {sampath, lin}@research.bell-labs.com

***Abstract*— To support real-time applications such as Voice-over-IP within a 802.11 Wireless LAN, efficient handoff mechanisms are required when a mobile client moves from one Access Point to another. In this paper, we present the design, implementation and performance results of MAC layer (layer-2) handoff algorithms which implement *make-before-break* mechanisms at the MAC layer unlike current algorithms that are based on *break-before-make*. As a baseline, we first present an algorithm that uses a single radio card on the client (as is done traditionally), but optimizes MAC layer handoff by periodically probing in the background for Access Points on other channels even when it is already associated with an Access Point and actively sending and receiving data. We then present two novel algorithms that use two radio cards on the client so that when one card is involved in communicating data, the other card can probe for neighboring Access Points. Of these multiple radio algorithms, the first one uses dedicated data and control cards at the client thereby implementing a *soft* version of make-before-break. The second algorithm uses two cards which can both perform control and data forwarding functions thereby implementing a *strict* version of make-before-break. Experimental results from a prototype implementation show that the make-before-break algorithms lead to much decreased MAC layer handoff overhead; the algorithm for the strict version of the make-before-break mechanism leads to sub 10 millisecond handoff latency.**

I. Introduction

In a 802.11 Wireless LAN (WLAN) [1], mobile devices use Access Points (AP) to connect to the wired network and communicate with other hosts. Given that the coverage area of an AP indoors is limited to around 200 to 300 feet, multiple APs are needed to cover a large area. When a mobile client moves from the coverage area of one AP to another, it has to end its association with the *old* AP and start communicating via the *newer* one. This *handoff* could primarily occur at two layers of the protocol stack. If the APs function as layer-2 bridges then handoff is restricted to layer-2 as the old and new APs belong to the same IP subnet. If the APs belong to different IP subnets then layer-3 handoff would have to be performed as well.

A main design issue in the deployment of multiple 802.11 APs to build a WLAN that covers a large area is that of seamless roaming support for mobiles within the network. Mobility has to be supported without any service disruptions; this means, transport and application level sessions should not be disrupted during handoffs between APs. Moreover, real-time applications such as VoIP have strict response time constraints and require that layer-2, and (if necessary) layer-3 handoffs be performed very quickly. For example, disruptions in a VoIP call would be noticed if the "jitter" is above 50 msec.

Currently, layer-2 handoffs in 802.11 WLAN networks that use off-the-shelf components use a *break-before-make* approach. In this scheme, the radio card on the mobile client first breaks its connection with the current AP, if the SNR on this link goes below a certain threshold. It then probes for other available neighboring APs to connect to. Once a "suitable" AP is discovered, the mobile client authenticates with the AP and then performs a layer-2 association. This whole process could take upwards of hundreds of milliseconds depending on the card that is being used, as shown in [2]. In addition, layer-3 handoffs (if required) would add additional latencies on the order of hundreds of milliseconds if standard schemes such as mobile-IP or its variants were used [3]. The above discussion illustrates the point that both layer-2 and layer-3 handoff latencies need to be reduced substantially to support emerging real-time applications on 802.11 WLANs.

In this paper, we focus on the reduction of layer-2 handoff latency in 802.11 WLANs. We introduce the novel concept of applying *make-before-break* mechanisms to 802.11 MAC layer handoff algorithms. As a baseline, we first present an algorithm that uses a single radio card on the client, but optimizes layer-2 handoff by periodically probing in the background to gather neighborhood information even when it is already connected to an AP. We then present two algorithms that use extra hardware to enable the independent execution of these operations. In the first of these algorithms, a second radio card (referred to as the "control" card) is exclusively used to probe for neighboring APs. Information obtained from this control card is then used by the other card (referred to as the "data" card) during handoff. Since the data card has to break its connection with the old AP before performing an *authentication* and *association* with the new AP, this approach implements a *soft* version of the make-before-break mechanism. In the second algorithm, control and data communication functions alternate between the two cards and a single IP address is shared between them. In this algorithm, when one card is actively sending and receiving data (data card), the other card (control card) probes for APs in the neighborhood. If quality of the channel to the current AP deteriorates, the control card authenticates and associates with another AP that it may have identified during the probing phase. The data card then disconnects from the old AP and its IP address is assigned to the control card (which becomes the new data card). The old data card starts probing for neighborhood

APs and takes over the functionality of the control card.

The rest of the paper is organized as follows. The next section describes the motivation for our work as well as related work in this area. Section III describes our layer-2 handoff algorithms and their tradeoffs. Section IV describes a prototype implementation of the different handoff algorithms. Section V provides experimental results based on measurements conducted on the prototype implementation. Section VI concludes the paper.

II. Motivation and Related Work

As mentioned in the previous section, low-latency handoff schemes are important both from the perspective of layer-2 (MAC) and layer-3 (IP). At layer-3, a standard mechanism for supporting mobility is mobile-IP [4]. But handoff latency with Mobile-IP is quite high. Variants of Mobile-IP and other micro-mobility protocols have been proposed to decrease layer-3 handoff but we will not discuss them further in this paper, as the focus here is on layer-2 handoff.

Initial works that identified the problem of layer-2 latency overhead in 802.11 WLANs were [2] [5]. It was shown that existing layer-2 schemes lead to handoff latencies on the order of hundreds of milliseconds. Further, it was shown that commercially implemented mechanisms for layer-2 handoff are based on a break-before-make paradigm where the client breaks its existing connection to an Access Point once the signal strength on that connection goes below a certain threshold before making a connection to a new AP. The process of connecting to a new AP was shown to consist of four steps namely *probing* (also referred to as *scanning*), *channel switching*, *authentication* and *association*. During the probing phase, the client *probes* all the channels sequentially to find another AP with a better signal strength to connect to. Once an AP is found, the client *switches* its channel to that of the AP. It then tries to *authenticate* with it. If the authentication is successful, the client *associates* with the AP. The association step completes the layer-2 handoff process. The probing phase was shown to dictate the overall layer-2 handoff delay. If probing is *passive* where the client switches to a channel and waits for a *beacon* to be received from an AP, in the worst case, the wait on each channel could be on the order of 100 milliseconds (which is the typical beacon interval). With a large number of channels to be scanned (11 channels in 802,.11b), the probing overhead could be on the order of a second. An alternative to reduce this delay is the *active* probing approach in which, clients broadcast "probe request" frames to force APs to respond immediately. Even using such an *active* probing scheme, it was shown [2] that the probing delay could be in the range of 350 to 500 milliseconds.

Normally, the channel switching delays are dictated by the hardware and the authentication and association delays are less than 10 milliseconds. Thus previous work on layer-2 handoff has focussed on decreasing the probing delay. In a proposed solution based on neighbor graphs [6], each client is made aware of the neighboring Access Points and their channels so that the client can probe a reduced set of channels thereby reducing the probing delay. Similar schemes that require the client to acquire wireless network topology information with respect to AP placement have been proposed in [7] and [8]. These approaches require **a)** changes to the 802.11 protocol itself[1], **b)** significant changes to Access Point implementations and **c)** the active management of information which could be a problem in large-scale deployments of these networks. Other related work includes the use of context caching [9] to reduce authentication delay, where security state is cached and moved between APs. Recently, a scheme referred to as SyncScan [10] has been proposed that aims to eliminate handoff delay by performing passive probing in the background. In this scheme, it is assumed that the clocks on all the APs can be synchronized and that each AP broadcasts its beacon based on a well-defined time schedule. By switching to channels on a scheduled basis and looking for beacons, the client can learn about the neighboring access points. However, this scheme suffers due to several drawbacks such as, the requirement for clock synchronization on all the APs, the possible loss of frames during channel switching and the performance degradation in congested networks – either the client will miss the beacon or wait longer on the channel. The implementation attempts to address some of these concerns but the requirement for the calculation of an *a priori* channel switching schedule based on an initial, one-time "hearing" phase assumes a static environment with a relatively stable set of APs. This will be infeasible in a dynamic environment with an *a priori* unknown or frequently changing set of APs. This is also the drawback of the scheme proposed in [11], where a reduced set of channels are probed, based on a one-time "hearing" phase and adjacent AP information from each probe is cached. Finally, all these schemes are still break-before-make and address only the probing delay.

In our approach, by adopting make-before-break semantics, we aim to eliminate *all* sources of layer-2 handoff delay. Further, we aim to present robust techniques with applicability to static as well as dynamic environments. Given that 802.11 radio cards are a commodity and their costs are decreasing, we believe it feasible to implement our schemes by incorporating two radio cards on the client. An alternative to this dual-card approach would be a single-card, dual-radio approach. However, even though such cards [12] exist in today's market, none of them offer independent control of the radios. Use of multiple radios to enhance mobility management has been mentioned as a possible application in [13] but no implementation is presented to validate this concept. Our contribution in this paper is the detailed design, implementation and experimentation of layer-2 make-before-break algorithms using two radio cards on mobile clients.

III. Layer-2 Handoff Algorithms

As a baseline to compare our algorithms based on two radio cards, we first present a handoff algorithm that uses a single radio card but optimizes on the channel probing phase by performing this task periodically in the background. We refer to this as the *one-card periodic probe* approach. We then present

[1]It should be noted that extensions to the 802.11 protocol such as 802.11k to perform radio resource management and dissemination of the list of neighborhood APs to the clients, and 802.11r to perform fast basic service set transition (by authenticating a priori at a new AP) are under consideration

two dual-radio algorithms which implement the make-before-break mechanism in varying degrees. We refer to these two algorithms as the *two-card static* approach and the *two-card dynamic* approach, respectively.

A. *One-card periodic probe algorithm*

In this algorithm, we use a background probing approach similar to the one proposed in [10] except that we do not require the APs to send beacons at fixed time periods. When the card is actively associated with an AP, we periodically let it switch and perform *active* probing on all other channels (then switch back to the original channel to send and receive data).

This ensures pro-active maintenance of up-to-date information about the neighboring APs but it also suffers from packet loss and increased jitter when probing is performed (the SyncScan solution [10] has the same problem but to a lesser extent as **a)** the client knows exactly when to expect a beacon and **b)** it scans one channel at a time as opposed to our solution that probes all channels in one *probing* phase.) In addition, the probing delay component of the layer-2 handoff latency is eliminated only if the handoff event does not coincide with a periodic probing event. This approach serves as a baseline in our experiments.

B. *Two-card static algorithm*

In this algorithm, each mobile client uses a second radio card exclusively to gather information about nearby APs. In other words, one card is dedicated for *control* and the other for *data*. The data card is associated with the current AP and is responsible for typical data communication functions. The control card actively probes for APs in the vicinity. The exact algorithm is as follows:

```
On control card  {
  every x intervals of time {
    probe all 802.11 channels;
    /* (channel, SNR, ESSID) */
    store info. on neighborhood APs;
  }
}
On data card {
  every y intervals of time {
    /* To current AP */
    monitor channel quality;
    if (channel quality degrades) {
      /* Based on stored info */
      choose ‘‘best’’ AP;
      disassociate with current AP;
      change channels;
      authenticate with new AP;
      associate with new AP;
    }
  }
} /* x -> large enough to reduce
overhead & small enough to maintain
up-to-date info. y << x for fast
response to channel degradation. */
```

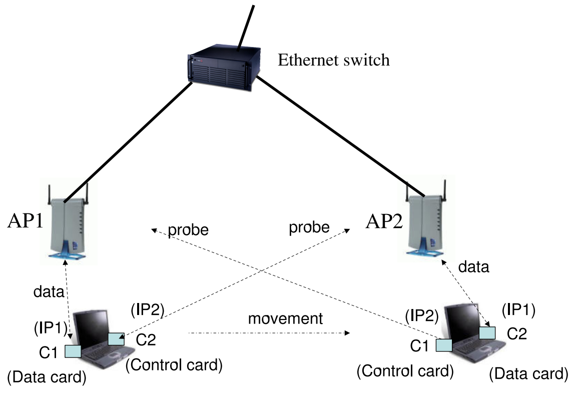

Fig. 1. Two-card dynamic algorithm

In the probing phase, information about discovered APs such as its extended service set (ESSID), channel number and SNR are locally stored and constantly updated. When the channel quality of the link to the current AP starts degrading, the stored information is consulted to determine which is the current "best" AP. Based on this information, the data card *switches channels*, *authenticates* and *associates* with that AP. Thus, using the two-card static algorithm, the probing delay can be completely eliminated, but the channel switching, authentication and association delays still remain.

C. *Two-card dynamic algorithm*

In this algorithm, the two cards on the mobile client alternate between the functionalities of the data card and control card, as shown in figure 1. Initially, the mobile client is using card C1 as the data card to associate with AP1; the IP address of the mobile client (IP1), is associated with card C1. At this time, card C2 serves as the control card and actively probes for other APs in the vicinity. Assume that the SNR on the channel through which C1 is connected to AP1 goes below a threshold. At this time, C2 identifies AP2 as the "best" AP and initiates layer-2 authentication and association. Note that during this time, C1 is still sending and receiving data through AP1. Once the layer-2 association completes (which means layer-2 handoff has been completed), IP1 is now associated with C2. This is when C1 ceases to be the data card and C2 becomes the new data card. Once card C2 is assigned IP1, the routing table on the client is correspondingly changed to reflect the new mapping between IP1 and interface C2. A gratuitous ARP is also sent to the router behind AP2 to update its ARP cache and preserve the reverse path to the client. Data will now be sent and received by C2 through AP2. From this point onwards, C1 will take over as the control card. The exact algorithm is as follows:

```
/* Initialization */
curr_data = card1; curr_control = card2;
On curr_control  {
  every x intervals of time {
    probe all 802.11 channels;
    /* (channel, SNR, ESSID) */
    store info. on neighborhood APs;
  }
}
```

```
On curr_data {
  every y intervals of time {
    /* To current AP */
    monitor channel quality;
    if (channel quality degrades) {
      /* Based on stored info */
      choose ‘‘best’’ AP;
      On curr_control {
        change channels;
        authenticate with new AP;
        associate with new AP;
        Assign current IP;
        Change routing table;
      }
      disassociate with current AP;
      swap(curr_control,curr_data);
    }
  }
} // x and y constraints remain same.
```

Note that in this algorithm, the probing, channel switching, authentication and association delays have all been eliminated. The only delay components that are incurred are **a)** the delay to associate the IP address with the new data card, **b)** the delay to change the routing table at the client and **c)** delay to send a gratuitous ARP; these delays add up to only a few milliseconds as shown in Section V.

IV. Implementation

We have implemented prototype versions of all three handoff algorithms on Linux kernel v2.4.26 using wireless 802.11b NICs based on Intersil's Prism chipsets (using the HostAP driver v0.3.7 [14]). We chose this driver as it provides a feature (`iwpriv wlan0 host_roaming 2`) to disable firmware-based as well as driver-based roaming and enable user-space implementation of these functions.

We were able to fully implement all three algorithms in user-space. This was largely due to the extensive set of lower-level details exposed by the HostAP driver. Our main motivation for choosing user-space over kernel-space was ease of implementation. We did, however, need to make a minor wireless device driver modification – specifically to send out association requests and authentication frames on the "current" channel (set using `iwconfig` or the equivalent ioctl) rather than channel 1[2]. Note that it is possible to use these user-space implementations with other wireless device drivers as well provided they support the ability to: **1)** disable in-built "scan-and-associate" algorithms, **2)** export control and status monitoring of these functions to user-space (via ioctls or other interfaces), and **3)** support two cards on the same host.

All algorithms were implemented using an application, written in C that consists of two threads – one to perform the control card functionality (referred to as the control thread) and the other to perform the passive monitoring of the link to the current AP as well as implementing functions to complete layer-2 handoff (referred to as the data thread). The functionality is split in this manner to enable a common implementation framework for both the one-card and the two-card algorithms. In the presence of a second wireless card, each thread can utilize a seperate card. Channel conditions are measured using the signal strength reported by the card in dBm. Also, to avoid a ping pong effect, the client performs handoff to a new AP only if its signal strength is greater than the current one by a certain margin (empirically determined to be 5dBm in our setup). Signal strength for new APs is available from the *probe response* frames received. Periodic measurement of signal strength, periodic active probing and forcing associations to specific APs as well as terminating them are achieved using *ioctl* functions provided by hostap or the Linux wireless tools package [15]. We now proceed to describe how the implementation differs for each of these algorithms.

A. One-card periodic probe algorithm

In this algorithm, *a single interface (wlan0)* is shared between all applications running on the client and hence, between the two threads of this application. Given the multiplexing of control thread with all other applications on the common card and the high cost of probing [2], two factors that could adversely impact the performance of all applications on the client are: **1)** the amount of time spent in one probe cycle, and **2)** how often this application initiates these probe cycles. The amount of time spent in one probe cycle is determined by a number of factors including the hardware and the number of channels probed. For an active probe of all 802.11b channels, this number has been empirically determined to be on the order of hundreds of milliseconds [2]. The interval between two consecutive probe cycles can be set and depends on the client environment. More specifically, it is a function of how fast the "active set" of APs near the client changes. An ideal interval value is one that keeps up-to-date information without adversely impacting active applications. In all our experiments, we set this value to 1 second in order to measure the performance under highly dynamic environments.

B. Two-card static algorithm

In this algorithm, one wireless interface (wlan1) is dedicated for the control thread and the other (wlan0) is multiplexed between other applications on the client and the data thread. If there is a process utilizing the data card while the channel switching and association procedure is taking place, it could potentially result in packet loss, increased delay or reduced throughput. That being said, we still expect this algorithm to outperform the one-card periodic probe algorithm.

C. Two-card dynamic algorithm

For this algorithm, we maintain one DHCP assigned IP address per client and a dummy IP address. The dummy IP address is associated with the control card (say wlan1) and the DHCP assigned IP is associated with the data card (say wlan0).

[2]The driver, in its unmodified form, sends out association requests and authentication frames on channel 1 unless it knows about the AP *a priori* using driver-level data structures. Given our user-space implementation, these data structures were never updated. Since all we required was the ability to associate with APs on different channels, we modified the driver such that if the AP is unrecognized, frames are sent out on the channel that is explicitly set rather than on channel 1.

During handoff, the assignment of these IP addresses to the respective interfaces is swapped and the kernel routing table is modified appropriately. Both the swapping of IP addresses and the modification of the routing table are achieved using *ioctl* function calls.

V. Results

In this section, we analyze the performance of the three handoff algorithms by simulating a real-time streaming application using ICMP ping requests. Ping packets are generated periodically at 10ms intervals – a rate far more aggressive than that used in actual interactive applications. We installed a modified version of the `ping` program on the mobile to generate periodic probe packets to a local destination. Because each ping (ICMP echo) request carries a sequence number, a lost `ping` packet can be easily checked. For each received `ping` reply, the program records the arrival time, the sequence number, the round-trip time, the signal quality, and the AP used, all in RAM to minimize the overhead caused by the added data collecting code. When the `ping` program terminates, the data is dumped from RAM to a file on disk for off-line analysis.

Each experimental run consists of a mobile client communicating with a fixed wired node while handing off between two Access Points (AP). All runs were carried out in a corridor of our lab at walking speeds and the two APs were operating at 802.11b channels 1 and 6. The mobile client was an IBM Thinkpad T20 running Linux kernel v2.4.26, using Microsoft 802.11b PCMCIA cards, based on Intersil's Prism chipset. The client was either carried by a person or placed on a cart along with a "sniffer" laptop (an IBM Thinkpad T40p running tcpdump). Both APs were Dell PCs running Linux 2.4.26 kernel. The APs are configured to belong to the same IP subnet and thus act as bridges. This implies that when the mobile moves from one AP to another it moves within the same IP subnet and thus there is only layer-2 handoffs are performed. Thus, we eliminate the effect of layer-3 handoffs on the measured parameters and focus only on the layer-2 mechanisms described in the paper. Twenty experimental runs were performed for each of the handoff algorithms. The metrics we use to evaluate the three schemes are *inter-arrival time (IAT)*, *packet loss* and *total handoff latency (THL)*.

A. Inter-Arrival Time at the receiver

To measure the IAT at the receiver (fixed wired node), we used the Click Modular Router software package (v1.4.3) [16] to print out the timestamp and sequence number of each incoming ICMP request. Figure 2 shows the mean and standard deviation curves for the IAT for the three schemes in every run. From this figure, all three average curves appear to be quite close to the "ideal" 10ms IAT line, with the two-card algorithms expectedly outperforming the one-card approach. However, we can also see that the one-card periodic scan algorithm has a very high variance In order to investigate this high variance, we analyzed the IAT curves from a single run for all three algorithms (figure omitted due to space considerations). In these curves, we observed periodic bursts with very large IAT values (around 700ms) for the single card approach. Given their periodic nature and magnitude, we believe that the periodic active scanning of all channels and the subsequent buffering of packets to be the primary cause for these bursts as well as the subsequent high IAT variance.

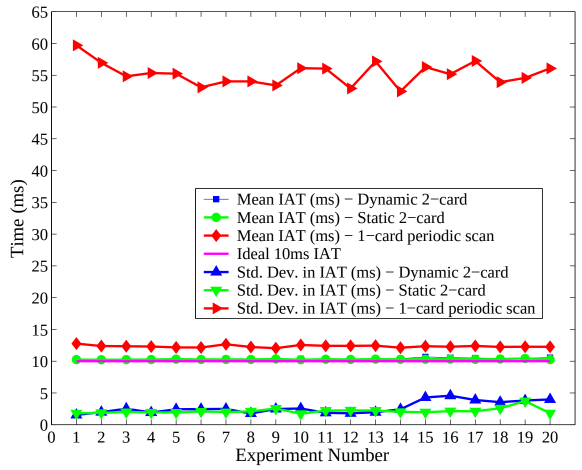

Fig. 2. Average and std. dev. of inter-arrival times (IAT), measured at the receiver, for the three schemes. The ideal 10ms IAT line is also plotted.

B. Packet loss measured at the sender

Packet losses are measured at the sender as the number of ping replies that are missed by the mobile during handoff using sequence number analysis.

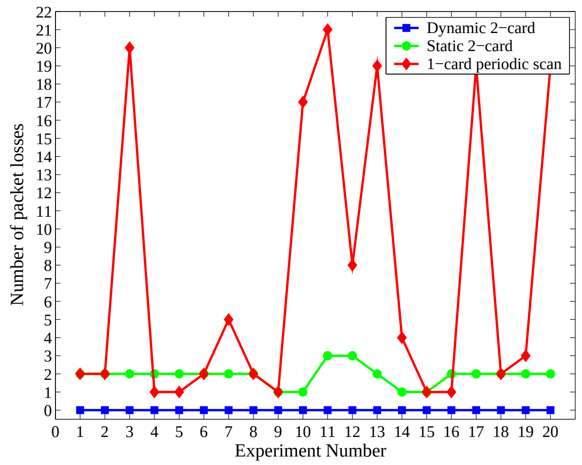

Fig. 3. Packet loss plotted for all three schemes.

Figure 3 represents the packet loss measured for each scheme at the sender. Note that this includes losses in the forward and reverse paths and provides us with an approximation of "worst-case" performance. The two-card dynamic algorithm has zero packet losses in all experimental runs. The two-card static algorithm surprisingly shows a few packet losses in spite of its maintaining an active set of APs and the usage of extra hardware. We hypothesize that the primary reason for this packet loss is the channel switching delay of around 20 milliseconds associated with these cards [10]. Furthermore, we observed this delay being present even when the two APs were on the same channel leading to our hypothesis that both the firmware and driver lack sanity checks to obviate the delay in this special case. The packet loss curve for the one-card approach shows high variance and we hypothesize that the reason behind it is the periodic nature of scanning in our implementation and the random nature of when handoff actually takes place. If the card starts scanning just prior to handoff and the buffer overflows either at the client or the AP, it results in packet loss. However, if the scanning does not take place around the time when the client initiates handoff, client and AP buffers mask any packet losses that would occur. These same buffers also result in the high variance in the IAT curves seen earlier.

C. *Total Handoff Latency (THL)*

The total handoff latency is defined as the time it takes for the client to transition from one AP to the next. Given the differences in the design and implementation, the total handoff latency for the two-card dynamic algorithm was measured differently from the other two algorithms. For the one-card periodic scan algorithm and the two-card static algorithm, THL was measured from the tcpdump trace as the time difference between the de-authentication and association request frames, sent out by the client. We observed that whenever the client switches from one AP to the other, it first sends out a de-authentication frame to the older AP before sending out the association and authentication frames to the newer AP. For the two-card dynamic algorithm, THL was measured as the time it takes to switch IP addresses on the interfaces, to setup the IP routing table (using *gettimeofday* system call) and to send out a gratuitous ARP.

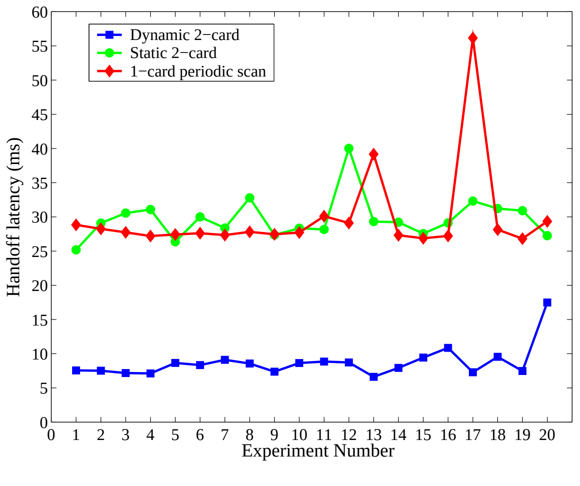

Fig. 4. Handoff latency in ms, plotted for all three schemes.

Figure 4 shows the THL for all three algorithms in all runs. Each point on each curve represents the time it took for the solitary handoff to complete in that experimental run. As expected, the dynamic two-card algorithm outperforms the other schemes by about 200% and provides sub 10 millisecond layer-2 handoff performance. However, the graphs show that both the two-card static algorithm and the one-card periodic scanning algorithm have similar THL numbers. One reason for this is that the one-card periodic scanning algorithm scans for other channels periodically and this may not coincide with an handoff event in which case the handoff latency is not influenced by the active scanning latency (probing delay on the order of hundreds of milliseconds). Also, the use of the appearance of a de-authentication frame as an indication of handoff ignores any active scanning latency (if present) prior to this event. Thus, our technique underestimates THL for the one-card algorithm and should be improved to detect and account for the active scanning latency as well.

VI. Conclusions

We presented the design, implementation and experimental measurements for soft-handoff algorithms in a 802.11 WLAN. As a baseline case, we first presented a periodic, background probing approach using a single radio card on the client. We then presented two soft-handoff algorithms that implement make-before-break semantics; of these, the two-card static algorithm uses a dedicated control card to probe for APs in the vicinity while the data card is sending and receiving data. The two-card dynamic algorithm switches the two cards between the control and data modes of operation. The first algorithm eliminates the probing delay but the authentication, association and channel switching delays still remain. The second algorithm eliminates all the components that make up the layer-2 handoff latency in a traditional implementation. Experimental results show that sub 10 millisecond handoff latencies can be achieved with the two-card dynamic algorithm.

References

[1] IEEE. ANSI/IEEE Std 802.11, 1999 Edition. 1999.
[2] A. Mishra, M. Shin, and W. Arbaugh. An empirical analysis of the IEEE 802.11 MAC layer handoff process. *ACM Computer Communication Review*, 33(2):93–102, April 2003.
[3] S. Sharma, N. Zhu, and T-C. Chiueh. Low-Latency Mobile IP Handoff for Infrastructure-Mode Wireless LANs. *IEEE JSAC*, 22(4):643–652, May 2004.
[4] C. Perkins. IP Mobility Support for IPv4. *RFC-3344, IETF*, Aug 2002.
[5] F.K. Al-Bin-Ali, P. Boddupalli, and N. Davies. An Inter-Access Point Handoff Mechanism for Wireless Network Management: The Sabino System. *Proceedings of ICWN*, 2003.
[6] M. Shin, A. Mishra, and W.A. Arbaugh. Improving the Latency of 802.11 Hand-offs using Neighbor Graphs. *Proceedings of ACM MobiSys*, June 2004.
[7] H. Velayos and G. Karlsson. Techniques to Reduce IEEE 802.11b MAC Layer Handover Time, Kung Tekniska Hogskolen, Stockholm, Sweden, Technical Report TRITA IMIT LCN R 03:02. April 2003.
[8] S. Pack and Y. Choi. Fast Inter-AP Handoff Using Predictive Authentication Scheme in a Public Wireless LAN. *IEEE Networks*, August 2002.
[9] M. Shin, A. Mishra, and W.A. Arbaugh. Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network. *Proceedings of IEEE INFOCOM*, March 2004.
[10] I. Ramani and S. Savage. SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks. *Proceedings of IEEE INFOCOM*, March 2005.
[11] S. Shin, A. G. Forte, A. S. Rawat, and H. Schulzrinne. Reducing MAC layer handoff latency in IEEE 802.11 wireless LANs. pages 19–26, NY, USA, 2004.
[12] Cisco aironet 802.11 a/b/g cardbus wireless lan client adapter. http://www.cisco.com/en/US/products/hw/wireless/ps4555/products_data_sheet09186a00801ebc29.html.
[13] P. Bahl, A. Adya, J. Padhye, and A. Wolman. Reconsidering Wireless Systems with Multiple Radios. *CCR*, 34(5), October 2004.
[14] J. Malinen. Host ap driver for intersil prism2/2.5/3 chipset-based cards. http://hostap.epitest.fi.
[15] Wireless tools for linux. http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Tools.html.
[16] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.