AuthScan: Enabling Fast Handoff across Already Deployed IEEE 802.11 Wireless Networks

Jaeouk Ok, Pedro Morales and Hiroyuki Morikawa
Graduate School of Frontier Sciences, The University of Tokyo, Tokyo 153-8904, Japan
Email: {okjaeouk, pedro, mori}@mlab.t.u-tokyo.ac.jp

Abstract—Handoff procedure in IEEE 802.11 wireless networks must be accomplished with as little interruption as possible to maintain the required quality of service (QoS). We have developed a fast handoff scheme, called AuthScan, to provide a novel usage of the open system authentication phase to reduce channel scanning latency. AuthScan comprises two steps: firstly, a client caches its handoff history with beacon information. Secondly, when in need of handoff, a client transmits Authentication Request frames to the selected Access Points (APs) from the cache instead of broadcasting Probe Request frames as in active scan to discover the available APs. The next AP is selected by comparing quickly acquired handoff metrics during the handoff procedure. This fast handoff is possible by upgrading only software on the client side. This paper presents the theoretical handoff latency of AuthScan in comparison with other approaches, and shows the effectiveness of our system through implementation and experiments.

I. INTRODUCTION

Fast handoff in IEEE 802.11 wireless networks [1] can provide not only higher throughput to best-effort applications, but also uninterrupted service to delay-sensitive applications on the move. However, the current standard does not provide clients moving across Basic Service Sets (BSSs) with fast handoff functionality, because an AP in infrastructure mode was originally designed to provide wireless connectivity for fixed, portable, and moving clients mainly within a BSS. Although there are various works in progress for standardizing fast handoff in IEEE 802.11 Working Groups, the sheer number of already deployed devices makes it necessary to have a practical solution to work with the current standard.

Link-layer handoff is composed of four sequential phases: detection, channel scanning, link-layer authentication and reassociation. Previous works reported that the standard handoff incurs a latency in the magnitude of hundreds of milliseconds to several seconds, and the channel scanning latency accounts for more than 90% of that [2]. This is unacceptable for delay-sensitive applications such as Voice over IP (VoIP). Thus, our work focuses on minimizing the channel scanning latency.

Our goal is to achieve fast handoff by reducing the time-consuming channel scanning latency in IEEE 802.11 wireless networks without modifying the current standard. We further require that 1) the handoff metrics (e.g. signal strength, etc.) should be promptly acquired during the handoff procedure and 2) the fast handoff scheme should work with only client software upgrade. The former increases the likeliness of selecting the next AP providing the highest QoS on the spot and at the moment of handoff. Handoff metrics collected over a long time period and before the handoff triggers are liable to become outdated at the moment of handoff trigger, therefore their validity comes into question. As for the latter, upgrading hardware to the client, or modifying already deployed APs hinders the adoption of this new technology, and limits its impact.

We propose a fast handoff scheme, called AuthScan, to provide a novel usage of the open system authentication phase for reducing channel scanning latency. Channel scanning performs two different tasks in a single phase: 1) finding information about “which APs have an overlapping coverage area with that of the currently associated AP”, and 2) acquiring and comparing their handoff metrics to find “which AP provides the best QoS to the client at the moment of handoff.” With nearby APs’ identity acquisition (the first task) performed before the handoff trigger (like other approaches that use the static characteristics of APs’ location over time [3] [4] [5]), AuthScan tackles quick handoff metrics acquisition and their comparison (the second task). The basic idea behind AuthScan is to exploit multiple Authentication Request/Response frames and decide the next AP to handoff based on the metrics promptly measured during multiple open system authentication phases. Instead of requiring extra support from the already deployed APs [6], AuthScan utilizes a client’s handoff history for addressing the target APs to scan in a unicast fashion. In this paper, we present the theoretical handoff latency of AuthScan with comparison to other approaches, and show the effectiveness of our system through implementation and experiments.

The rest of this paper is organized as follows. In Section II, we explain link-layer handoff procedure in IEEE 802.11 wireless networks, related work and the channel scanning latency analysis. Section III describes our proposed method to improve the efficiency of channel scanning while satisfying the additional two requirements. Section IV presents the details of our implementation on a Linux notebook and experimental results. Section V concludes the paper, and shows future work.

II. BACKGROUND

A. Link-Layer Handoff

Link-layer handoff procedure consists of four sequential phases: detection, channel scanning, link-layer authentication, and reassociation.
Detection is the phase where the need for handoff is discovered. Detection latency is not considered in measuring handoff latency, because it can be completely eliminated by using handoff triggers from the physical layer.

Channel scanning is the phase to acquire the information about nearby APs operating on different channels. The standard defines two methods: passive and active scan. In passive scan, a client listens to each channel for beacon frames. In active scan, a client broadcasts Probe Request frames and waits for Probe Response frames on each channel.

Link-layer authentication is the phase to verify the identity of each other between an AP and a client. The standard defines two algorithms: open system and shared key authentication. The latency of each phase sums up to one and two round trip times (RTTs), respectively.

Reassociation is the phase for a client to acquire permission to access the wired network over an AP, taking one RTT.

Because Wired Equivalent Privacy (WEP) was deprecated by the amended standard due to its security vulnerabilities in 2004, Wi-Fi Protected Access (WPA) based on IEEE 802.11i [7] has gained popularity. WPA implements a four-way handshake after reassociation. It adds 2 RTTs to the conventional link-layer authentication latency.

B. Related Work

The necessity of channel scanning phase comes from a client’s inability to acquire information about APs on different channels while communicating with the currently associated AP. Its large latency lies on the inefficiency that the standard mandates a client not only to scan through all the channels regardless of AP’s existence on it, but also to wait for a fixed amount of time at each channel just in case more responses might arrive. Many fast handoff schemes have been proposed to address the channel scanning issues, being classified into two groups as below.

The first group tries to eliminate the necessity of channel scanning phase. Some approaches decouple the time-consuming channel scanning phase from the actual handoff procedure and do scan earlier for maintaining a list of candidate APs with their handoff metrics before the connection to current AP is terminated [8] [9]. Pre-scan approaches, however, have a risk that the acquired information may become outdated by the time that the handoff decision has to be made. The other problem of pre-scan approach would be the performance degradation of its on-going session. In [8], an extra method is also required to synchronize all the APs. Other approaches involve extra support to enable scanning while communicating with the currently associated AP. A shared beacon channel is introduced in [10], and multiple network interface cards (NICs) are utilized in [11]. These hardware augmentation approaches have a deployment difficulty due to its high overhead, and power consumption concerns.

The second group tries to improve the efficiency of channel scanning phase. The first way to do this is to reduce the number of channels that are effectively going to be scanned by the client. This can be achieved through various methods such as a cache [3], Neighbor Graph (NG) [4], sensor overlay network [5], etc. These selective scan approaches, however, would end up with poor handoff performance in scenarios such as densely deployed APs over various channels. Another way to improve the efficiency is by reducing the time waiting at each channel. In order to do this, a client is provided with an AP list, and only scans target APs in a unicast fashion to reduce the time to wait at each channel by creating new unicast scanning messages [6] [12] [13]. The performance of this approach depends on the prior knowledge on nearby APs.

C. Channel Scanning Latency

The standard defines two parameters involved in active scan at each channel: MinChannelTime, and MaxChannelTime. MinChannelTime is the time in Time Units (TU) long enough to guarantee the reception of a Probe Response frame. If a client waits during MinChannelTime without receiving any Probe Response after broadcasting Probe Request, it assumes that there should be no AP available in this channel. On the other hand, MaxChannelTime is the time long enough to guarantee the reception of the Probe Response frames from multiple APs available in the same channel. If a client receives a Probe Response during MinChannelTime after broadcasting Probe Request, it must extend its waiting time to MaxChannelTime in case more Probe Responses might arrive in the same channel. Typically used values are found below [12].

- \( \text{MinChannelTime} \geq \text{DIFS} + \text{CW} \ast \text{aSlotTime} = 50 \mu\text{sec} + 31 \ast 20 \mu\text{sec} = 670 \mu\text{sec} \) (Since \( \text{MinChannelTime} \) is defined in TU, \( \text{MinChannelTime} \) will be 1 TU (1024\( \mu\text{sec} \))
- \( \text{MaxChannelTime} = 15 \text{ msec} \)

If there is at least one AP in \( N \) channels in Japan (i.e. out of 18 channels in total: 1 - 13ch for 11 b/g, 14ch for 11h, and 34, 38, 42 and 46ch for 11a), the active scan latency would exceed \( (\text{MaxChannelTime} \ast N + \text{MinChannelTime} \ast (18 - N)) \). For example, if there are 2 channels having at least an AP on each of them, the active scan latency would exceed \( (15 \ast 2 + (1.024 \ast 16)) = 46.384 \text{ msec} \).

Besides the aforementioned parameters, there are hardware induced delays such as interface setup time. These delays are not considered in the previous analysis, because they vary from maker to maker and, therefore, are unsettled. In fact, the real latency observed in experiments is even larger than the values obtained in the analysis.

III. AuthScan

AuthScan comprises two steps: first, a client caches its handoff history with beacon information. Second, when in need of handoff, a client performs unicast scan by transmitting Authentication Request frames to the selected APs from the cache to discover the next AP to handoff.

A. Target AP Cache

The unicast scanning approach requires some method to acquire knowledge about nearby APs prior to performing channel scanning. This knowledge can be obtained by various
ways: cache [3], NG [4], pre-scan [8], multiple interfaces [11], off-line manual collection by service providers, etc. AuthScan builds a target AP list by caching previous handoff information simply because of its low overhead.

A cache does not require any data communication performance degradation, supports from the infrastructure network, modifications to the protocols nor hardware upgrade to any devices. All these benefits come at the cost of unimproved support for a client’s unusual behavior like visiting a place for the first time. The performance improvement will be higher for people with regular behavior, who we believe are the majority. Even though we chose a cache, note that AuthScan’s approach can be implemented with any of the aforementioned methods.

Each time a client performs handoff, it updates the cache information. Since an Authentication Response frame does not contain some information found in Probe Response or beacon frames, the target AP cache needs to save and update them for successful handoff completion. The information includes the BSSID of AP before handoff, the BSSID of AP after handoff, channel number, PHY type, capability information, SSID, supported rates, PHY parameter sets, WPA parameters, etc. The handoff count during a fixed time period is used to filter stale information. An example of a possible cache structure is shown in Table I. In order to add newly built APs into the target AP cache, a client needs to perform active scan periodically or on demand.

| AP<sub>prior</sub> (00:05:4e:XX:XX:00) | count | SSID | channel | PHY type | capability info | supported rates | ...
|----------------------|-------|------|---------|----------|----------------|----------------|------
| 00:05:4e:XX:XX:01    | 12    | AP1  | 42      | 11a      | 0x0511         | 6, 9, 12, 18, 24, 36, 48, and 54 | ...
| 00:05:4e:XX:XX:02    | 4     | AP2  | 14      | 11b      | 0x0531         | 1, 2, 5.5, and 11 | ...
| 00:05:4e:XX:XX:03    | 8     | AP3  | 6       | 11g      | 0x0531         | 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48, and 54 | ...
| 00:05:4e:XX:XX:04    | 22    | AP4  | 1       | 11g      | 0x0531         | 1, 2, 5.5, 11, 6, 9, 12, 18, 24, 36, 48, and 54 | ...
| 00:05:4e:XX:XX:05    | 3     | AP5  | 34      | 11a      | 0x0511         | 6, 9, 12, 18, 24, 36, 48, and 54 | ...

B. Handoff Procedure

In our system, handoff procedure is performed in the following steps. When detecting the need for handoff based on its policy (e.g. signal strength, transmission rate, missed beacon number, retransmission number, etc.), a client looks up its target AP cache and selects one as the next target AP using the handoff count number for a fixed time period. Then it sets up its interface to the desired channel and PHY type, and transmits an Authentication Request to the selected AP.

We define two algorithms: comparative mode, and fast mode. In comparative mode, a client checks all the target APs in the cache, before selecting the AP to handoff. This is done by sending Authentication Request frames to all the target APs, and comparing the Received Signal Strength Indications (RSSIs) measured when receiving Authentication Response frames. This is in conformation with the current standard which allows authentication with multiple APs. Since the number of entries in the cache affects the total delay of the procedure, we will discuss what would make a reasonable number of APs to scan based on time constraints from experiments later in Section IV.

In the case of fast mode, shown in Figure 1, if the RSSI of the received Authentication Response frame ($RSSI_{authres}$) is higher than a certain threshold ($RSSI_{threshold}$), the client immediately moves to the reassociation. If it receives an Authentication Response frame whose RSSI is lower than the threshold or there is no response during $MinChannelTime$, it repeats this operation with the next target AP. In case no AP in the cache can satisfy the client’s handoff policy, the client moves to active scan and saves the newly found AP in the cache.

In terms of AuthScan algorithmic delay, and assuming at least one of the cached APs is available and can fulfill the handoff policy, comparative mode takes $M * RTT + (N - M) * MinChannelTime$, where $N$ is the number of target APs in the cache, and $M$ is the number of Authentication Response frames received. In the case of fast mode, under the best case scenario the first AP from the cache provides signal strength higher than the threshold, so the delay is $1 * RTT$. The worst case happens when the last AP from the cache provides signal strength higher than the threshold, so the delay is the same as in the case of comparative mode.

For example, assume that there are five target APs under open system authentication on different channels to one another in the cache as shown in Table I, and four of them return Authentication Response frames. The highest total handoff la-
tency will be composed of four RTTs for the APs that reply, the one waiting period of MinChannelTime due to the AP that does not reply and one RTT for reassociation phase. Therefore the total latency will be $4 \times \text{RTT} + 1 \times \text{MinChannelTime} + 1 \times \text{RTT}$. The total handoff latencies using various schemes under the channel assignment being used in Japan are compared in Table II, where RTT is 0.6 msec, beacon interval is 100 msec, MaxChannelTime is 15 msec, and MinChannelTime is 1024 μsec. From the table, we can observe that AuthScan achieves the lowest handoff latency, with one RTT less than selective unicast scan.

### IV. Implementation and Experiments

#### A. Implementation Details

We have opted for dividing the system into two main parts: changes to the kernel driver, and a userland tool to control the handoff procedure. The driver changes are only the minimum required to support the new functionality, while all the actual work is done in the application. This provides us with greater flexibility and simplicity in this prototype implementation.

The driver changes are implemented in the madwifi driver [14], and consist of changing the protocol state machine, adding system calls to manage the driver from the application (ioctl) and generation of informational events from the driver to the application.

Even though the IEEE 802.11 standard allows to authenticate with multiple APs, the current protocol state machine does not support this behavior. The main problem that arises is that as soon as an Authentication Response frame is received by a client, the driver begins the reassociation phase by sending a Reassociation Request frame. This results in a driver interface that effectively fuses the authentication and reassociation phases in one that starts with the command to send the Authentication Request, and ends with the emission of an Associated Event. Failure is not clearly indicated, and therefore applications must set a timeout for the whole process.

In our implementation we enable the application to fully control the handoff procedure by making each stage explicit. To achieve this we first remove the automatic transitions in the protocol state machine. Then, upon reception of a Authentication Response frame we create and emit a new informational event to signal the application the end of the new stage. Finally, we add new system calls to accomplish these transitions from userland.

The controlling application is implemented by modifying wpa_supplicant [15], a client software which uses the interfaces provided by drivers to control the handoff process and supports WPA and WPA2. In order to do all of these, this application follows the protocol state transitions in its own state machine. This state machine has less states than the one in the driver because of the fused states we mentioned earlier, so the stage before being Associated is known as Associating, with no reference to the authentication phase.

The changes to the application involve the creation of a new state for the authentication phase, and the addition of the cache logic. The new state and transitions are obtained by making use of the newly created system calls and the handling of the new informational event. For the cache implementation we add a persistent storage of information, and change the application’s scanning procedure. In the case of a comparative mode, when the need of handoff is detected, we loop over the cache and try to authenticate with each AP in turn. As we explained before, failure is not clearly signaled by the driver, so we set up a timeout for this process. If the Authentication Response arrives before the timeout, we cancel the timeout and update the cache information for this AP. If the timeout triggers, the AP is flagged as unreachable.

#### B. Experimental Results

We have implemented a prototype of AuthScan client on an IBM Thinkpad X31 (CPU Pentium M 1.7GHz, 1GB RAM) with an Atheros AR5212-based wireless interface. It runs Debian Linux 4.0 Etch with a 2.6.18-5 kernel, modified madwifi as the wireless interface driver, and modified wpa_supplicant as the application software.

We evaluate the performance of our prototype by measuring handoff delay across BSSs of different PHY types and channels, where six APs are available: AP0(11b, ch. 11), AP1(11a, ch. 42), AP2(11b, ch.14), AP3(11g, ch. 6), AP4(11g, ch. 1), and AP5(11a, ch. 34), described in Figure 2. Assume that a

### TABLE II
Handoff Latency Comparison

<table>
<thead>
<tr>
<th>Handoff scheme</th>
<th>Channel Scanning</th>
<th>Authentication</th>
<th>Reassociation</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive scan</td>
<td>$18 \times \text{BeaconInterval}$</td>
<td>$1 \times \text{RTT}$</td>
<td>$1 \times \text{RTT}$</td>
<td>1.8012 msec</td>
</tr>
<tr>
<td>Active scan</td>
<td>$4 \times \text{MaxChannelTime} + 14 \times \text{MinChannelTime}$</td>
<td>$1 \times \text{RTT}$</td>
<td>$1 \times \text{RTT}$</td>
<td>75.536 msec</td>
</tr>
<tr>
<td>Selective active scan [4]</td>
<td>$4 \times \text{MaxChannelTime} + 1 \times \text{MinChannelTime}$</td>
<td>$1 \times \text{RTT}$</td>
<td>$1 \times \text{RTT}$</td>
<td>62.224 msec</td>
</tr>
<tr>
<td>Selective unicast scan [6]</td>
<td>$4 \times \text{RTT} + 1 \times \text{MinChannelTime}$</td>
<td>$1 \times \text{RTT}$</td>
<td>$1 \times \text{RTT}$</td>
<td>4.624 msec</td>
</tr>
<tr>
<td>AuthScan</td>
<td>0</td>
<td>$4 \times \text{RTT} + 1 \times \text{MinChannelTime}$</td>
<td>$1 \times \text{RTT}$</td>
<td>4.024 msec</td>
</tr>
</tbody>
</table>

![Fig. 2. Experimental setup](image-url)
client with an IEEE 802.11 a/b/g NIC is in the middle of a cross section while associated to 11b AP0. As the client moves towards 11a AP5, it transmits ICMP Echo Request frames to the target host in the same subnet. We set ICMP frame size as 480 bytes, and interval as 10 msec. We trigger the client’s handoff by reducing the transmission power of AP0. At the moment of handoff it scans the other APs (AP1-AP5) before reassociating with AP5.

Figure 3 shows one example of the way RTT changes during handoff from AP0 to AP5. The x-axis shows the ICMP sequence number and the y-axis shows the RTT in milliseconds. Due to the different data rates, we find that the average RTT while associated to 11b AP0 (1.709 msec) is more than twice as large as the one obtained while associated to 11a AP5 (0.754 msec). We can see that handoff from AP0 to AP5 occurs between 28th and 31st frame (i.e., 4 frames dropped, 40 msec disrupted).

Figure 4 shows the average handoff delay from ten runs of the experiment since the sending of the Authentication Request frame to the first AP scanned (1AQ in the graph). The x-axis shows the steps in the authentication scanning process. They correspond to the sending of the Authentication Request frame (AQ), reception of the Authentication Response frame (AS), sending of the Reassociation Request frame (RQ) and reception of the Reassociation Response frame (RS). The number before each of them is a sequential number related to authentication scanning steps, and does not refer to the actual AP name being checked. The y-axis is the time delay in milliseconds measured in the application in order to get the handoff delay from the user’s perspective. We added a checkpoint right before calling the driver ioctl, in the case of the AQ and RQ, and right after receiving the driver’s informational event for the AS and RS.

The total handoff delay obtained when checking five APs with open system authentication is 48.97 msec in average. It takes 8.266 msec in average to check each AP. This includes hardware induced delay such as interface setup time, delay introduced by system calls and events that flow between userland and kernel space, and 1 RTT for Authentication Request and Authentication Response. Note that this procedure is run through the modified wpa_supplicant in the application level. Though the current prototype is able to check five APs within 50 msec, the number of APs to scan under the same time constraint would increase by moving the authentication scanning logic from the application to the driver.

V. CONCLUSION AND FUTURE WORK

We have developed a fast handoff scheme, called AuthScan, to provide a novel usage of the open system authentication phase for reducing channel scanning latency. AuthScan maintains a target AP list cached from a client’s previous handoffs, and performs unicast scanning by transmitting not Probe Request frames, but Authentication Request frames only to the selected APs. In this paper we have shown the theoretical handoff latency and the effectiveness of our system through implementation and experiments. Our future efforts will be oriented to the accurate cache management and the method to reduce the number of APs to check when using fast mode.

REFERENCES