

# SyncScan: Practical Fast Handoff for 802.11 Infrastructure Networks

Ishwar Ramani

Department of Computer Science & Engineering  
University of California, San Diego  
E-mail:ishwar@cs.ucsd.edu

Stefan Savage

Department of Computer Science & Engineering  
University of California, San Diego  
E-mail:savage@cs.ucsd.edu

**Abstract**—Wireless access networks scale by replicating base stations geographically and then allowing mobile clients to seamlessly “hand off” from one station to the next as they traverse the network. However, providing the illusion of continuous connectivity requires selecting the right moment to handoff and the right base station to transfer to. Unfortunately, 802.11-based networks *only attempt* a handoff when a client’s service degrades to a point where connectivity is threatened. Worse, the overhead of scanning for nearby base stations is routinely over 250ms – during which incoming packets are dropped – far longer than what can be tolerated by highly interactive applications such as voice telephony. In this paper we describe SyncScan, a low-cost technique for *continuously* tracking nearby base stations by synchronizing short listening periods at the client with periodic transmissions from each base station. We have implemented this SyncScan algorithm using commodity 802.11 hardware and we demonstrate that it allows better handoff decisions and over an order of magnitude improvement in handoff delay. Finally, our approach only requires trivial implementation changes, is incrementally deployable and is completely backward compatible with existing 802.11 standards.

## I. INTRODUCTION

Wireless broadcast networks, by their very nature, provide the opportunity for user mobility. Within the range of a given wireless base station, a client may roam freely and with complete transparency to the network medium. It is exactly this capability that has driven the success of the \$1.5 billion 802.11-based wireless network market [1]. Inexpensive 802.11-based *access points* (APs) provide transparent connectivity to the wired Internet at low cost and with minimal configuration overhead. However, each individual 802.11 access point (AP) has a limited range - frequently under 100 meters indoors - and therefore large-scale deployments of access points are required to provide comprehensive coverage of a building or campus (e.g. UC San Diego’s campus 802.11 network, seen in Figure 1, includes over 250 managed public access points).

Preserving the same network transparency across collections of access points requires a far greater degree of coordination and management. As a client moves outside the range of one access point, it must “hand off” to another to preserve the illusion of seamless connectivity. In cellular data and voice systems this handoff decision is typically coordinated by the network itself, which is able to leverage considerable information about the network topology and client proximity. However, in contrast, the 802.11 standards dictate that handoff



Fig. 1. Geographic map of 802.11b wireless coverage at UC San Diego (courtesy <http://activecampus.ucsd.edu>). The entire coverage area is serviced by over 250 “official” access points and again as many “private” access points.

should be managed autonomously and independently by each client with no *a priori* knowledge of the local network topology [2]. Moreover, while most cellular systems provide a means for continuously monitoring signal quality between each client and *all* of its neighboring base stations – handing off *whenever* an alternative base station is found to provide better service – 802.11-based systems only monitor the signal to their *current* access point and handoff only after service degrades below an acceptable threshold.

Consequently, as a mobile 802.11 client reaches the limits of its current coverage region, it must temporarily abandon its current access point, actively probe the network to discover alternatives, and only then reconnect to the current best AP. This approach minimizes management overhead, but is slow to handoff to superior access points and worse, can produce “gaps” in connectivity of up to a second in duration [3]–[5]. While such disruptions may be acceptable for nomadic applications with limited mobility and flexible response time requirements, emerging applications like wireless voice-over-IP are far more demanding [6], [7]. These applications require highly interactive response *during* mobility and are extremely sensitive to network outages and delays. Moreover, the limited range of 802.11 radios makes handoff actions highly probable for continuously mobile clients (such as a user walking with an 802.11-based phone).

In this paper, we address all of these issues by focusing

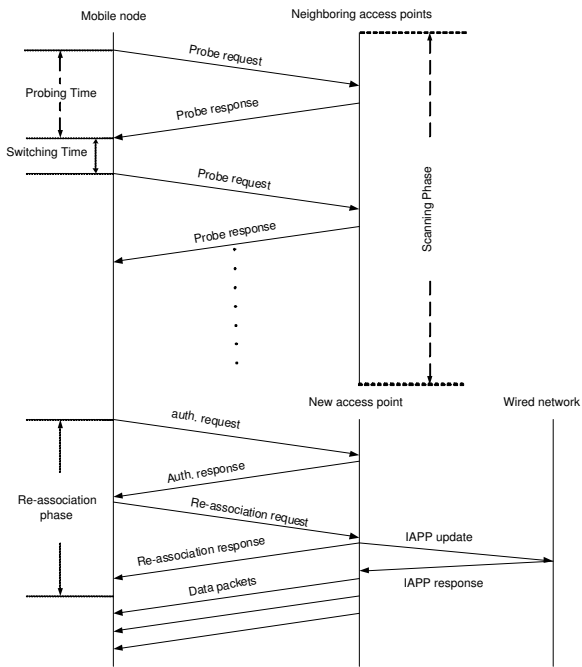


Fig. 2. A timing chart of the 802.11 handoff procedure. Three phases – scanning, authentication and association – are illustrated.

on a single simple problem: how to *continuously* monitor the proximity of nearby 802.11 access points? Our solution, called SyncScan, replaces the large transient overhead of active access point discovery with a continuous process that passively monitors other channels for the presence of nearby access points. The potential disruption of channel switching is minimized by synchronizing short listening periods at the client with regular periodic transmissions from each access point. We have implemented the SyncScan algorithm using commodity 802.11 network interfaces and show that we can improve the timing of handoff decisions (thus improving the client’s signal quality) and reduce handoff delay by over an order of magnitude compared to the existing approach. We demonstrate these benefits in the context of mobile voice applications and yet show that our changes have negligible impact on normal bulk data transfers. Finally, our approach does not require any modification to the 802.11 protocol itself, is incrementally deployable and requires only minor modifications to existing implementations.

The rest of the paper is organized as follows. In Section II we review how existing 802.11 handoff mechanisms function and previous work in improving their overheads. Section III describes the SyncScan algorithm and its optimizations in detail, as well as the potential benefits arising from this capability. An experimental analysis of our prototype implementation is presented in Section III-A. Finally, we summarize our results and open questions in Section V.

## II. BACKGROUND AND RELATED WORK

The handoff process in 802.11 networks has several phases – each with its own costs. Figure 2 depicts this time-line

graphically.

First, a client must *determine* that it is nearing the periphery of its coverage and thus must find an alternative access point to continue. Minimally, this involves detecting that packets are no longer being successfully received. However, typical commercial implementations also monitor the current signal-to-noise ratio (SNR) and will also initiate the scanning phase when this value passes a pre-defined minimum threshold (that said, Velayos and Karlsson provide evidence that a simple trigger of three dropped packets is actually quicker and more accurate than this approach [4]). Setting this threshold is something of a black art: if the client waits too long to look for new access points then it may incur additional disruption, yet if the client is too eager then it may ping-pong between access points needlessly.

Once a client has decided to attempt a handoff it must next identify the set of proximate candidate access points. Since 802.11 does not provide a shared control channel or other means for distributing this information, the client must explicitly “scan” each channel (11 in 802.11b and 802.11g, and 8 for 802.11a indoors) for potential access points. In its simplest form, this scan can be completely passive – the client switches to a candidate channel and listens for periodic *beacon* packets generated by access points to announce their presence (typically every 100ms). However, the latency incurred by this approach can be quite long since the phase of beacon intervals is independent and a client must therefore wait the full interval on each channel. Thus:

$$ScanDelay = NumChannels \cdot MaxBeaconInterval$$

For example, in 802.11b, with 10 channels to be scanned (not counting the current channel) and a beacon interval of 102ms, the scanning latency is well over a second.

To reduce this delay, most 802.11 implementations actively broadcast a probe packet on each channel to force an access point to respond immediately. Consequently, the delay incurred in completing an active scan is regulated by five parameters:

- *ProbeDelay*. Upon settling on a new channel, the NIC passively monitors the channel for activity before sending a probe packet. However, to ensure that a lightly loaded channel does not block the scanning process, the *ProbeDelay* timer indicates a time after which a probe is sent even without monitored activity. Default values for this parameter are negligible ( $< 1ms$ ).
- *Media contention time*. The 802.11 MAC protocol provides a distributed contention mechanism that can cause a probe packet to be delayed, while seeking access to the medium. On lightly loaded networks, this value is also negligibly small.
- *MinChannelTime*. This represents the amount of time to wait for the first response before declaring the channel empty (i.e., no access point in range). This value is an adjustable parameter and is set differently by different systems. Different researchers have suggested ideal

settings of `MinChannelTime` ranging from 1ms [4] to 7ms [8].

- *MaxChannelTime*. This represents the amount of time to wait to collect potential *additional* probe responses from other access points. This value is meant to be configured based on an estimate of the number of overlapping access points and the load on the channel. Previous empirical studies suggest values of roughly 11ms for this parameter [4], [8].

Ignoring the negligible components, the idealized latency of this active scanning approach is:

$$\sum_{c=1}^{c=NumChannels} (1 - P(c)) \cdot Min + P(c) \cdot Max$$

where  $P(c)$  is the probability of one or more access points operating in channel  $c$ . Thus, for aggressive values of 1ms and 11ms for `MinChannelTime` and `MaxChannelTime`, the ideal latency should range from 11ms to 110ms for 802.11b. However, this analysis ignores a practical implementation parameter: the channel switching delay. This overhead is a characteristic of the network interface design and reflects the time to switch to a new frequency, resynchronize and start demodulating packets. Channel switching delay varies considerably across implementations and we have measured times ranging from a maximum of 19ms (12ms to switch and 7ms to resynchronize) for Intersil Prism2-based NICs to just over 5ms for Atheros 5212-based NICs. Since this cost is *per channel* it adds considerable delay to the overall scanning process.

In total, the exact overhead of scanning can vary significantly based on the environment, choice of parameters and hardware capabilities. That said, in our measurements we have found popular Intersil Prism2-based 802.11b NICs observe scanning delays of between 350 and 400ms (based on a firmware initiated scan) and NICs based on Atheros 5212-based designs are roughly 500ms (via a Windows XP driver-controlled scan). This is comparable to the results reported by Mishra et al. [8]<sup>1</sup>

Based on the measured SNR for the probe response packets received from each access point, the client will then select the best access point for *authentication* and *association*. A new access point is only favored if its measured SNR is at least  $\Delta$  above the associated access point as shown in Figure 3. This hysteresis is used to avoid unnecessary handoff operations that might produce a “ping-pong” effect when clients are equally well served by different access points.

Authentication is required to validate the client’s right to use a particular access point and minimally requires a two way handshake (under 1ms in our measurements). Following authentication, a client must then *reassociate* with the new access point to finalize its decision to transfer. In public access systems, reassociation delay is also well under 1ms. However,

<sup>1</sup>To further minimize scanning time, some vendors prioritize the *order* in which channels are scanned to favor “popular” channels (such as the non-overlapping 802.11b channel set of 1, 6 and 11) and will cut short scanning if an AP with the same SSID and an acceptable SNR is found.

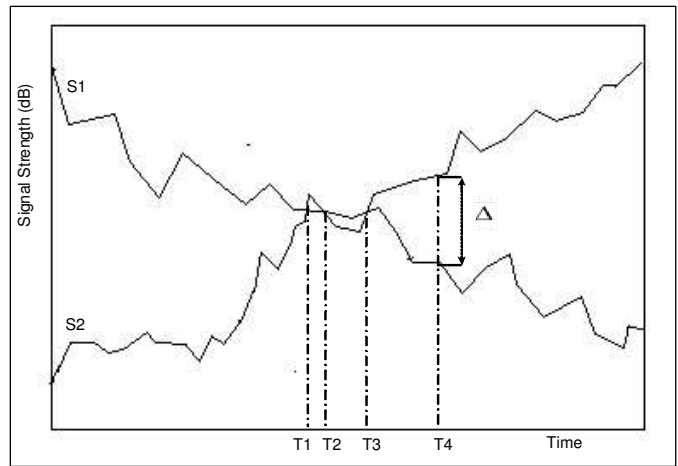


Fig. 3. How received signal strength changes over time from two different access points ( $S_1$  and  $S_2$ ). If the client is associated with  $S_1$  it will consider a handoff when its signal strength drops below the threshold at  $T_1$  (in fact, the graph for  $S_2$  to the left of  $T_1$  is not measured in practice since scanning only commences when  $S_1$ ’s signal has degraded below the threshold). Handoff is only performed when the signal from the new access point,  $S_2$ , is  $\Delta$  better than the signal from  $S_1$ .

PHASE	TIME
Scanning	350-500ms
Authentication	<10ms
Association	<10ms
Wired update	<20ms

TABLE I

APPROXIMATE OVERHEADS FOR CURRENT 802.11 HARDWARE ENGAGED IN EACH STAGE OF THE HANDOFF PROCESS.

in environments implementing the Inter-Access Point Protocol (IAPP) to transfer security state between the old AP and the new AP [9], there may be additional costs to reassociation. In this paper we focus on public access networks, but we refer to reader to recent work by Mishra et al. which demonstrates a proactive caching approach for reducing this cost to under 20ms on average [10].

Finally, the wired network must be informed of the handoff and directed to forward packets destined for the client via the newly associated access point (in most enterprise 802.11 deployments, in which APs share a single LAN or VLAN, this function is implemented by the 802.1d spanning tree protocol).

Table 1 summarizes these costs and while these values can vary (we refer the reader to Mishra et al. for a more in-depth measurement analysis of these overheads [8]), the overall conclusion does not. The scanning phase dominates the cost of handoff – usually contributing more than 90 percent of the overhead.

There have been several previous attempts to minimize 802.11 handoff latency. Velayos et al. and Pack et al. both propose a scanning mechanism using topographical knowledge of access point placement [4], [11]. If each client is made aware of its neighboring access points and their channels, the client can then probe a reduced set of channels during the

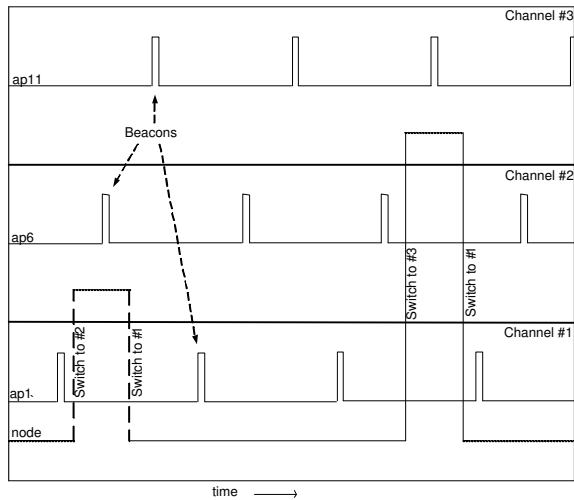


Fig. 4. The state of access points and client in terms of the channel occupied and signal generated. The client associated with AP1 uses SyncScan to receive beacons from AP6 on channel 2 and AP11 on channel 3.

scanning phase and spend less time on each channel. While this approach is attractive, the maintenance and dissemination of this knowledge potentially represents a large management burden for those deploying large-scale 802.11 networks. Recently, Shin et al., have proposed a similar scheme, based on *neighbor graphs* and provide an algorithm for updating these graphs in a distributed on-line fashion [12].

However, all of these approaches require changes to the 802.11 protocol itself as well as significant implementation changes to clients and access points. Consequently, there is a significant deployment hurdle before these approaches will be available for the 10 million+ 802.11 wireless NICs in use today.

### III. SYNCSCAN

802.11-based access points periodically broadcast special beacon packets to identify themselves to potential clients and to synchronize state information with currently associated clients. While access points are commonly configured to send beacons every 100ms, within this period the 802.11 standard does not restrict the *absolute* time at which these packets are generated. We exploit this degree of freedom to synchronize clients with the timing of beacon broadcasts on each channel. In essence, we arrange so that the clients can passively scan by switching channels *exactly* when a beacon is about to arrive.

#### A. Basic Algorithm

At the heart of our approach is the creation of a staggered periodic schedule of beacon periods spread across channels. For example, all access points operating on channel 1 will broadcast beacons at time  $t$  (or close to it), while access points on channel 2 will do the same at time  $t+d$  ms, channel 3 access points will send at time  $t+2d$  ms, and so on. Thus, a mobile client associated with an access point operating on channel  $c$  can detect APs operating on channel  $c+1$  by switching to that channel  $d$  ms after receiving a beacon from its own

access point. We call this approach *synchronized scanning*, or SyncScan.

A mobile client can then utilize this property to efficiently locate all the access points in its neighborhood. By regularly switching to each channel (as shown in Figure III), a complete picture of all nearby access points can be observed and yet the client minimizes the time it is out of contact with its own access point. Consequently, when a handoff must be attempted, the cost is reduced to that of authentication and reassociation. In effect, SyncScan reduces the costs of continuous scanning activity to the point where it is easily amortized against the delay incurred by a full scan during a handoff. Moreover, SyncScan provides the opportunity to make better handoff decisions by continuously monitoring the signal quality of multiple access points (rather than just the associated AP).

The basic procedure as outlined above is backward compatible with existing 802.11 devices since the modification in the *phase* of beacon announcements has no impact on existing scanning approaches. It is also incrementally deployable, since a client employing SyncScan can always fall back to the traditional means of handoff if it cannot synchronize with the infrastructure. Moreover, since it only changes the phase of beacon generations, SyncScan does not require any change to the 802.11 standard itself – it is only an implementation modification.

That said, SyncScan does add several complexities. First, the accuracy of clocks in access points is critical to the global synchronization of beacon timings. Over short time-scales our measurements indicate that drift is negligible for existing commercial access points. However, over longer time-scales access points require a means of time synchronization. By far the simplest approach is to leverage the wide availability of Network Time Protocol (NTP) service over the Internet [13], since it provides a standard out-of-band means for synchronizing APs to absolute time reference (in fact, we find that most enterprise access point products already support NTP).

However, synchronizing brings its own potential risks. Multiple access points operating on the same channel will attempt to generate beacons at the same time – potentially interfering with one another. When the APs are within signal range of each other, the standard 802.11 clear channel assessment (CCA) algorithm used for media access will order conflicting beacons, but if they are further separated the beacons may yet interfere with each other at an intermediate client. To address this issue, the beacon generation time may be randomly varied over a small window (e.g. 3ms). A client lingering on a channel for the entire window should expect to receive most of the beacons on that channel.

Finally, the SyncScan procedure has a hidden cost. While it removes the transient overhead of the scanning phase, it replaces it with a regular overhead. While a client is listening to other channels, it cannot be sending or listening to its own access point. Moreover, the client may miss packets sent to it while it is exploring other channels. How significant these factors are depends, in part, on the overhead of implementing the SyncScan procedure. For each SyncScan session, the client

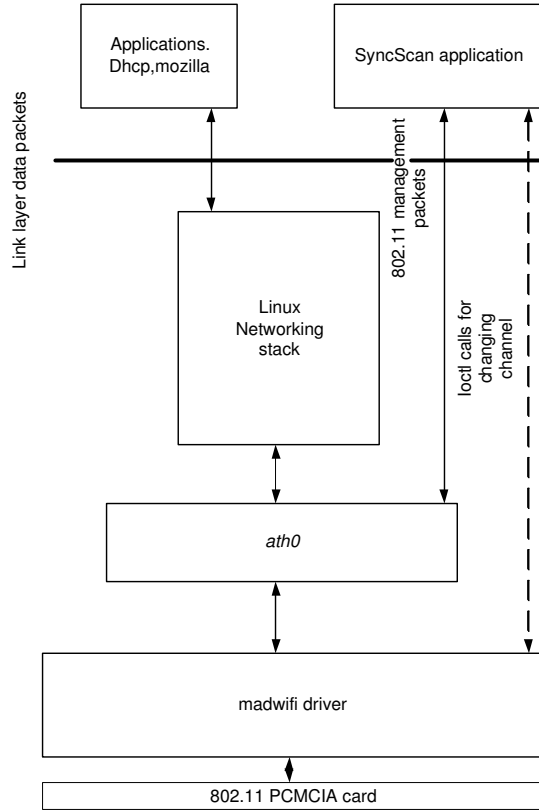


Fig. 5. Architecture of a prototype SyncScan implementation for Atheros chipsets under Linux.

must switch channels, wait for any beacon and then switch back. Thus,

$$SyncScanDelay = 2 \cdot SwitchTime + WaitTime$$

where *SwitchTime* is the time to switch channels and *WaitTime* is how long a client lingers waiting for beacons. How often this overhead is incurred, is a function of how many channels are scanned and how regularly a scan is initiated. In Section III-A, we evaluate these factors in the context of our prototype implementation and additional techniques to eliminate packet loss.

### B. Benefits

Since scanning overhead dominates the 802.11 handoff procedure, the most obvious benefit derived from SyncScan is the reduction in handoff latency. Using SyncScan an actual handoff can occur in a few milliseconds rather than the 400ms latency incurred using standard 802.11 implementations. This reduction is sufficient to preserve the illusion of continuity needed by interactive voice applications.

However, in addition to this obvious benefit, continuous scanning also presents other opportunities for improvement. In particular, while traditional 802.11 implementations initiate scanning only when the signal has degraded below a preset

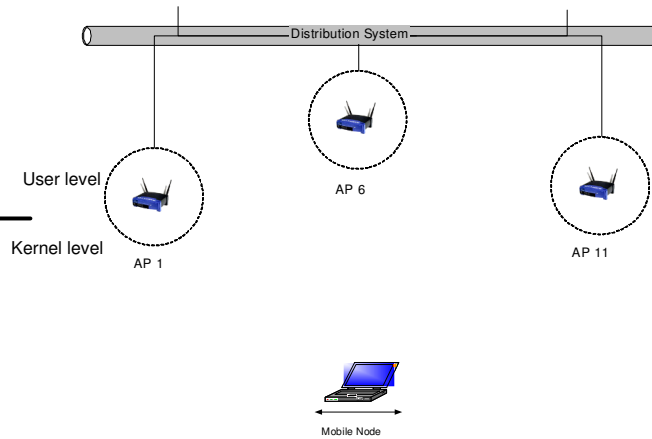


Fig. 6. Basic experimental setting, consisting of three access points in the corridor of our department.

threshold, continuous scanning provides the means to make far more intelligent choices about when to handoff. A continuous scanning implementation can discover the presence of access points with stronger SNRs even before the associated access point's signal has degraded below its threshold. Moreover, since a SyncScan client samples access point SNR measurements over time, it can use the *change* in signal quality as an input to its decision rather than relying on a single sample as in the traditional approach (thus shrinking the amount of hysteresis needed to accommodate variability). Taken together, these capabilities allow handoffs to be made earlier and with more confidence – thereby improving the quality of a client's connectivity and reducing the potential of unnecessary interference. We demonstrate the benefits of this approach in Section III-A.

Finally, SyncScan also provides an opportunity for continuous location tracking. Systems such as RADAR orient a mobile host via trilateration among the received signal strength from APs in known locations [14]. In typical use, this requires a full scan and consequently disrupts any ongoing communication. By contrast, SyncScan provides a means for continuously tracking signal strength from nearby APs while still carrying on communication with a chosen AP. However, the effectiveness of such systems is tied to the currency of its measurements. If the time period of SyncScan operation 500ms, then readings for APs on two distinct 802.11b channels could be separated by as much as 5 seconds in the worst case – potentially reflecting very different client locations. This problem can be addressed by increasing the frequency of SyncScan operations at the cost of reduced bandwidth available to the client. Alternatively, the client can prioritize its SyncScan choices to favor channels on which APs have been heard recently – thereby minimizing the delay between their measurements. We have not yet built a location tracking system based on SyncScan, but we believe that moderate levels of mobility (i.e. walking) should be well-addressed by SyncScan intervals between 200-500ms.

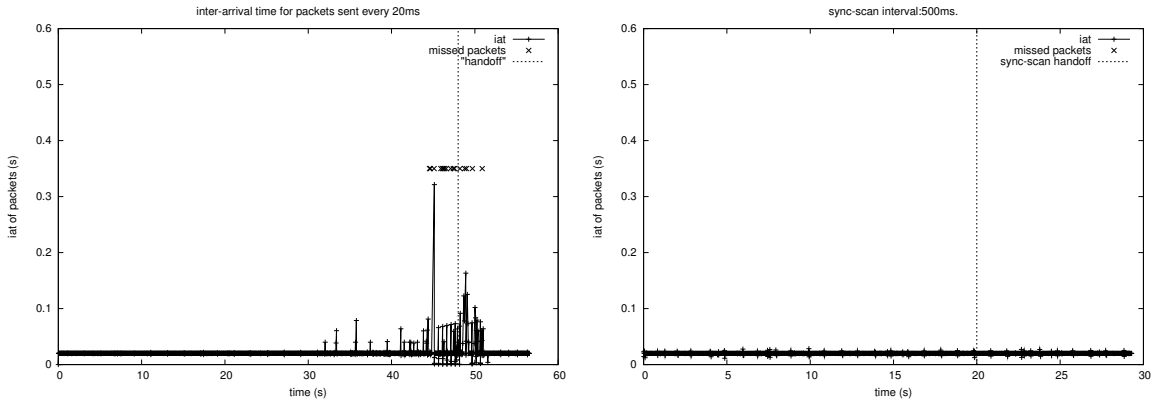


Fig. 7. Received packet inter-arrival times (in seconds) when the mobile node is operating with the standard Windows XP driver (left) and the SyncScan driver (right). Packets are generated at 20ms intervals.

#### IV. IMPLEMENTATION AND EVALUATION

We have implemented a prototype version of the SyncScan client on Linux for two popular 802.11 chipsets: Intersil Prism-based wireless NICs (via the popular HostAP driver [15]) and Atheros-based NICs (via Sam Leffler’s *madwifi* driver). Both environments allow the host to explicitly implement 802.11 management functions, such as scanning, beacon monitoring and association. The prototype’s basic architecture is illustrated in Figure 5 (which depicts the Atheros implementation) and consists of a modified driver which interacts directly with the NIC and with a user-level SyncScan daemon. This daemon, which implements the SyncScan algorithm itself, informs the driver when to switch channels, collects beacon frames and their SNR values, and intercepts management frames necessary for completing a handoff (e.g. association and authentication frames). We have implemented a SyncScan access point by making slight modifications to the *madwifi* driver under Linux. However, we did not use this access point in our experiments (for reasons described below).

During a SyncScan operation, inbound packets may be dropped since the client is unable to acknowledge them. As discussed in Section II the SyncScan delay consists of the time to switch to a new channel, the time to wait for a beacon and the time to switch back to the original channel. For Atheros-based NICs, the channel switching time is roughly 5ms, thus the total delay is 10ms plus any waiting time. For Intersil-based NICs this value is closer to 40ms. While these are relatively short intervals, especially if SyncScan operations are attempted twice a second, they are all greater than the maximum retransmission time for 802.11 frames (4ms) and consequently some packets will be lost.

To overcome this drawback, we modified the client to announce that it was entering Power Saving Mode (PSM) just before switching channels [2]. This causes the access point to buffer packets destined for the client until the client returns to the channel and resets the power saving mode. While these buffers are not large, they are quickly emptied when the client returns. While the client is scanning another channel, it buffers any outbound packets to ensure that they

are not lost either. This technique is similar to the one Bahl et al. used to create virtual wireless NICs [16]. Unfortunately, the *madwifi* software underlying our SyncScan access point does not yet support buffering client packets in PSM mode, and consequently we could not use it to demonstrate this feature. Instead, our experiments use unmodified commercial access points and we emulate access point synchronization at the client as follows: when the SyncScan client first starts, it explicitly synchronizes with all available access points by waiting on each channel for two beacons and recording the times of their arrival and beacon frequency. From these initial times, it calculates a schedule forward – as though their beacon timings were externally synchronized.

All of our experiments were performed in a corridor in our department. The corridor was long enough to warrant three access points which operated at channel 1, 6 and 11 as shown in Figure 6. The mobile client was a Dell Inspiron 6000 running Linux, typically using a Netgear WAG511 PCMCIA card for 802.11 connectivity (unless stated otherwise).

In the remainder of this section we analyze the performance and overhead of SyncScan operations and their impact on existing background traffic. First, we synthetically simulate a real-time application by streaming packets to the mobile client at regular intervals while varying the number of SyncScan operations attempted each second. We examine the packet loss and jitter that occurs while the client is idling and handing-off using the conventional handoff approach, and SyncScan with PSM support. Next, we qualitatively explore SyncScan’s benefits while using a commercial Voice Over IP (VOIP) client. We then explore the benefits of SyncScan for handoff timing – leveraging continuous scanning to make better handoff decisions. Finally, we look at the impact of SyncScan on traditional bulk data applications. We transfer a file using FTP and compare the bandwidth delivered to a normal client and one using SyncScan. This is a worst-case scenario for SyncScan – packets are being transmitted during the scan – and we investigate it to understand if SyncScan has a significant detrimental effect.

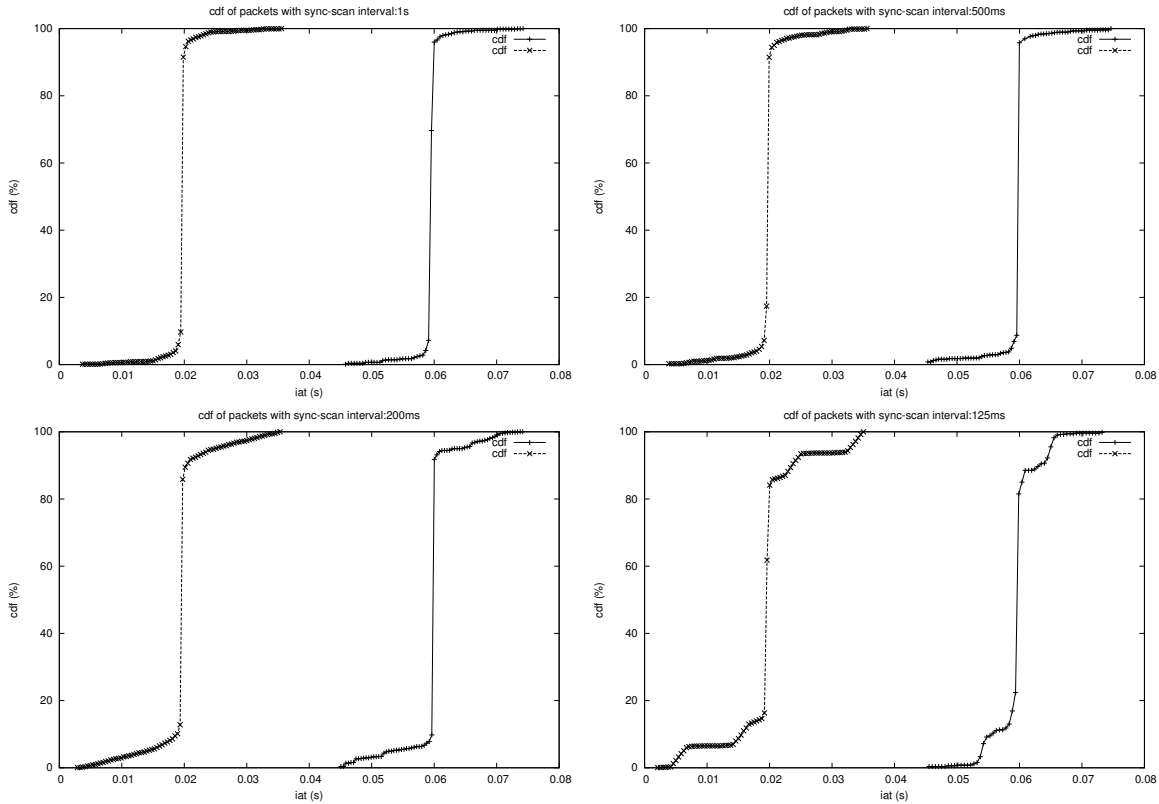


Fig. 8. Cumulative distribution function of packet inter-arrival time (in seconds) using SyncScan for 20ms and 60ms packet streams with varying SyncScan intervals.

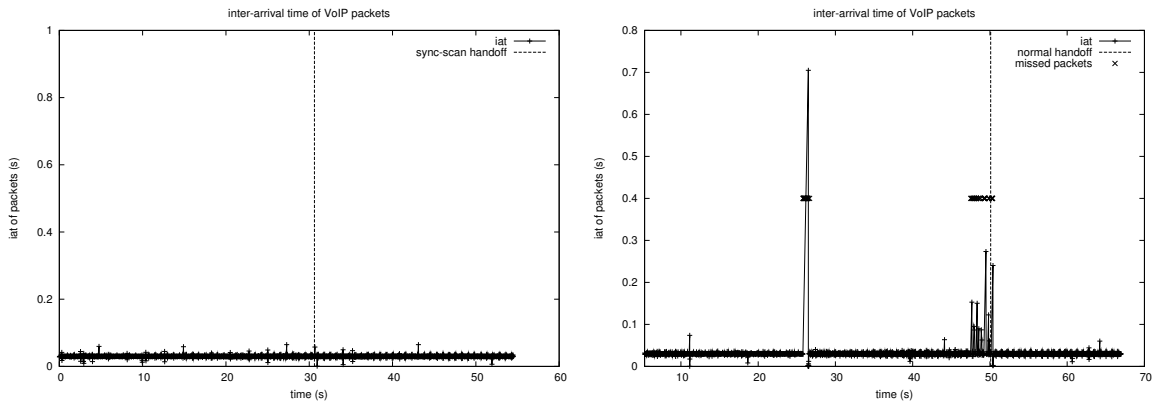


Fig. 9. Inter-arrival time (in seconds) of UDP packets when the mobile node is running a commercial VoIP client.

### A. Real-Time Traffic

To analyze the performance of SyncScan with real-time interactive traffic, we generated ICMP Echo Requests at periodic intervals. The packets were generated at 20ms intervals – the most aggressive rate used by interactive voice applications – and the size was set to resemble a typical RTP packet carrying 20ms audio samples. We also examined 60ms intervals to demonstrate audio applications that aggregate encoded frames within a single packet. The packet generator fed these packets into AP1 and traces were gathered at the mobile client. The experiment was conducted by moving the mobile client from the neighborhood of AP1 to AP11. The mobile client switched

access points at most twice: from AP1 to AP6 and sometimes from AP6 to AP11 (the handoff from AP6 to AP11 depended on the trajectory of movement). The parameters of SyncScan were set as follows: The *WaitTime* for the beacon capture was set conservatively to 5ms and SyncScan operations were initiated at performed at different rates – 1000ms, 500ms, 200ms and 125ms – to explore the tradeoff between temporal fidelity and application impact.

As a basis for comparing the performance of SyncScan with the existing scanning procedures, we first measured packet inter-arrival times when the mobile client was executing the default driver under Windows XP (this was a decision meant

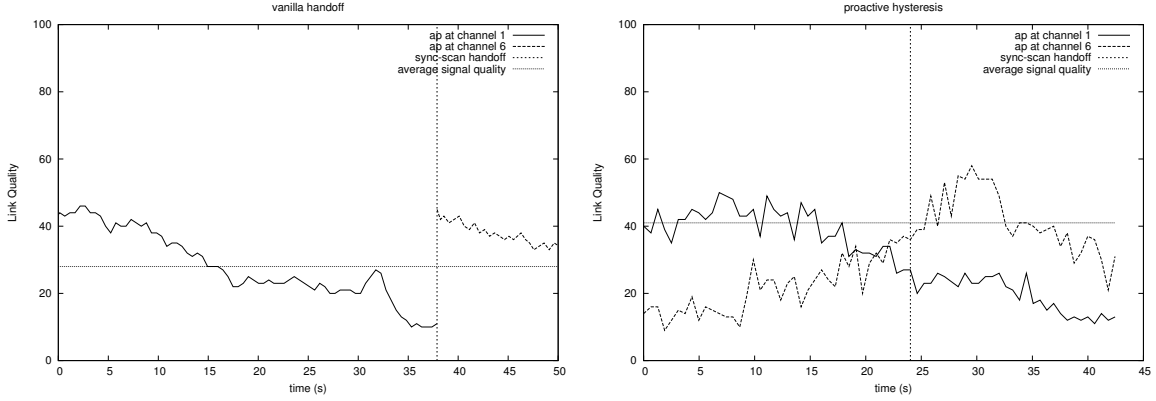


Fig. 10. NIC reported signal quality and the time of handoff using the traditional reactive handoff decision (left) and using a proactive handoff decision (right).

to put the default behavior in the best light, since the Linux scanning latency was even worse). Figure 7 presents the inter-arrival times for packet streams generated at 20ms using this driver (on the left). In general, they are quite close to 20ms until the client reaches the edge of its current APs coverage and starts scanning (roughly at time 45). At this point packet inter-arrivals increase and over 20 packets are lost. Finally, at time 48, after multiple scans have taken place (taking an average of 40ms each), the signal from a new AP is strong enough that a handoff takes place (indicated by the dotted line). The client continues to scan however and the inter-arrival time does not resume to 20ms until time 50.

The right side of Figure 7 shows the same experiment using SyncScan. Note that the y-axis is appropriately scaled to show the range of inter-arrivals – an order of magnitude less than using the standard driver. Moreover, there is no significant inter-arrival variation around the handoff time (20s) and no packets are lost. However, these packet losses are being masked by the PSM buffering and being “transformed” into additional jitter. The probability of packet being buffered is the probability that an arriving packet corresponds to a SyncScan operation:

$$P(\text{BufferedPacket}) = \begin{cases} \frac{SST}{IAT} & \text{if } SST \leq IAT \\ 1 & \text{otherwise} \end{cases}$$

where  $SST$  is the total time for a SyncScan operation (16ms in this experiment) and  $IAT$  is the packet inter-arrival time. While the PSM approach could also eliminate packet loss for conventional scanning, the large 300ms inter-arrival times would still remain.

Figure 8 also shows the cumulative distribution of inter-arrival times of 20ms and 60ms as SyncScan operations become more frequent. As the SyncScan frequency increases, more packets are buffered and hence arrive later than the expected time. At 1 second intervals over 95 percent of packets are within 2ms of the 20ms period, this decreases to a bit under 90 percent for the 200ms interval, reaching only 75 percent at the 125ms interval. For the same SyncScan time, more packets arrive closer to expected time for 60ms compared to 20ms.

For example, at 200ms interval 96% of the packets arrive at  $60 \pm 2ms$  compare to 90% for  $20 \pm 5ms$ . This is expected since, the chances of SyncScans occurring between two packet arrivals increases with an increase in the inter-arrival times. Hence the jitter is inversely proportional to the inter arrival time of packets. Note that in all cases the worst case jitter is 15ms (which is the SyncScan delay) for 20ms and 60ms sending interval. Consequently, there is a tradeoff between how frequently one scans other channels, the application’s inter-arrival rate and jitter. For highly-interactive applications, we believe that 500ms SyncScan intervals represent a reasonable tradeoff.

To analyze the performance of SyncScan, we tested it over a regular voice over IP session. We used a popular VoIP client called Skype which uses UDP packets exchanged between two clients for voice communication. The observed mean inter-arrival time over several experiments was 27ms with 98% of all samples being  $27 \pm 5$  ms. Figure 9 shows the performance of a Skype [17] session while walking between AP1 and AP6. The graph on the right is the inter-arrival time of UDP VoIP packets arriving at the client while using the standard driver. The card performs a scan at 28s and handoff at 50s. In both these operations, packet loss is severe. For the entire run 44 packets were lost. The graph on the left is the same application, but using SyncScan (with a scan interval of 500ms). We observed zero packet losses and no variations in inter-arrival times.

### B. Proactive Handoff

While SyncScan reduces the overhead of actually executing a handoff operation, it doesn’t impact the decision of *when* to handoff. Figure 10 shows how per-API signal quality (as reported by the NIC) changes over time and when a handoff takes place. The graph on the left shows measurements of the conventional reactive handoff approach – handing off at 38s when the old AP hits a signal strength of 10 (an average signal quality of 26 during the experiment). However, using the continual measurements taken during SyncScan operations it is possible to make proactive decisions about when to handoff

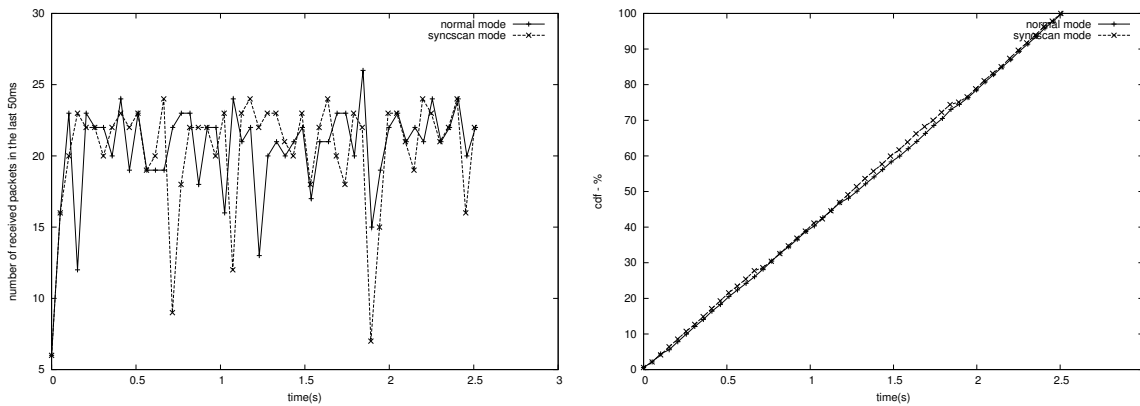


Fig. 11. Instantaneous bandwidth binned at 50 intervals and a CDF of packet vs time during an FTP transfer. Both graphs overlay both *normal* and SyncScan modes of operation and normal mode.

as well. The graph on the right depicts a *proactive* handoff driven by the relative signal quality received from distinct APs. Here the client performs handoff long before its AP reaches the minimum signal quality threshold and can quickly adopt AP 6 as the superior access point (an average signal quality of 42). This improvement increases the probability that client packets are received without retransmission and reduces the probability of packets being lost as the client reaches the edge of the APs range.

subsectionBulk data traffic We also analyzed the performance of TCP/IP over SyncScan using FTP. We compared the performance of an FTP transfer of the same file over SyncScan (using an interval of 500ms) and the normal 802.11 infrastructure mode. In both cases we used the popular Ethernet application to capture the traffic at the receiver. The file size was 1.5MB and was fetched from a well-provisioned remote server. The results are plotted in Figure 11. The first graph shows instantaneous bandwidth in 50ms buckets. During the SyncScan operations the bandwidth drops to a smaller value at 0.7 seconds and 1.9 seconds. But due to PSM buffering the overall throughput is not significantly impacted. To demonstrate this, the second graph depicts the cumulative distribution of packets received as a function of time. For transfers in normal mode and using SyncScan, both the CDFs run parallel to each other indicating the negligible difference in throughput.

## V. CONCLUSION

Mobile voice applications are the next challenge for 802.11-based wireless networks. One of the major impediments is the high cost of handoff as clients roam between access points in an infrastructure network.

In this paper we have proposed using implicit time synchronization to reduce the key cost of discovering new wireless access points. By synchronizing the announcement of beacon packets, a client can arrange to listen to other channels with very low overhead. As a result, handoff using this SyncScan approach is an order of magnitude quicker than using the conventional approach.

We have designed and implemented the SyncScan handoff procedure using existing commodity 802.11 hardware. In spite of the limitations of existing hardware, we have been able to demonstrate the dramatic benefits of SyncScan for interactive real-time traffic and that it has minimal impact on existing bulk-data applications.

## ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their constructive feedback, John Bellardo for his help in setting up our wireless infrastructure, Yu-chung Cheng for his input regarding PSM and letting us use machines provided by Google for testing. Marvin McNett for maintaining our systems. This work was funded in part by DARPA FTN Contract N66001-01-1-8933 and a grant from the UCSD Center for Networked Systems.

## REFERENCES

- [1] Dell'Oro Group, "Dell'Oro Group's Wireless LAN Five Year Forecast Report," jul 2003.
- [2] IEEE Computer Society LAN MAN standards Committee, "IEEE Standard for Information Technology: Part 11: Wireless LAN Medium access Control (MAC) and Physical Layer (PHY) Specifications," 1999.
- [3] A. Mishra, M. Shin, and W. Arbaugh, "An Empirical Analysis of the IEEE 802.11 MAC layer Handoff Process," University of Maryland, College Park, Tech. Rep. UMIACS-TR-2002-75, Nov 2002.
- [4] H. Velayos and G. Karlsson, "Techniques to Reduce IEEE 802.11b MAC Layer Handover Time," KunglTekniska Hogskolen, Stockholm, Sweden, Tech. Rep. TRITA-IMIT-LCN R 03:02, ISSN 1651-7717, ISRN KTH/IMIT/LCN/R-03/02-SE, April 2003.
- [5] F. K. Al-Bin-Ali, P. Boddupalli, and N. Davies, "An Inter-Access Point Handoff mechanism for Wireless Network Management: The Sabino System," in *Proceedings of the International Conference on Wireless Networks*, Las Vegas, NV, June 2003.
- [6] T. Henriksson, "Hardware architecture for 802.11b based h.323 voice and image IP telephony terminal," in *Swedish system-on-chip conference2001, Proceedings of the SSoCC*, Arild, Sweden, March 2001.

- [7] Cisco, "CISCO 802.11b based IP phone," <http://www.cisco.com/warp/public/cc/pd/tlhw/>.
- [8] A. Mishra, M. Shin, and W. Arbaugh, "An Empirical Analysis of the IEEE 802.11 MAC layer Handoff Process," *ACM Computer Communications Review*, vol. 33, no. 2, Apr. 2003.
- [9] IEEE Computer Society LAN MAN standards Committee, "IEEE. Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation," Jan 2002, IEEE Draft 802.11/D3.
- [10] A. Mishra, M. Shin, and W. Arbaugh, "Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network," in *Proceedings of the IEEE INFOCOM Conference*, Hong Kong, China, March 2004.
- [11] S. Pack and Y. Choi, "Fast Inter-AP Handoff Using Predictive Authentication Scheme in a Public Wireless LAN," in *Proceedings of IEEE Networks Conference*, Atlanta, GA, Aug. 2002.
- [12] M. Shin, A. Mishra, and W. Arbaugh, "Improving the Latency of 802.11 hand-offs using Neighbor Graphs," in *Proceedings of the ACM MobiSys Conference*, Boston, MA, June 2004.
- [13] D. Mills, "Network Time Protocol (Version 3) specification, implementation and analysis," IETF RFC-1305, March 1992.
- [14] P. Bahl and V. N. Padmanabhan, "RADAR: An in-building RF-based user location and tracking system," in *Proceedings of the IEEE INFOCOM Conference*, Mar 2000, pp. 775–784. [Online]. Available: [citeseer.ist.psu.edu/bahl00radar.html](http://citeseer.ist.psu.edu/bahl00radar.html)
- [15] J. Malinen, "Host AP driver for Intersil Prism2/2.5/3," <http://hostap.epitest.fi/>.
- [16] P. Bahl, P. Bahl, and R. Chandra, "Enabling Simultaneous Connections to Multiple Wireless Networks using a Single Radio," Microsoft Research, Tech. Rep. MSR-TR-2003-46, June 2003.
- [17] "Skype," <http://www.skype.com/>.