Beacon-Stuffing: Wi-Fi Without Associations

Ranveer Chandra, Jitendra Padhye, Lenin Ravindranath, Alec Wolman
Microsoft Research

1 Introduction

IEEE 802.11 [4] attempts to model wireless networks as a replacement for wired networks. For example, a wireless client needs to "associate" with an Access Point (AP) before it can communicate, which is similar to the act of connecting a wired client to an Ethernet bridge or a switch. Once a wireless client is associated to an AP, it can no longer communicate with other APs around it without using sophisticated software [3]. This wired model needlessly limits the capabilities of wireless networks. For example, when a wireless client can hear APs other than the one with which it is associated, this restricted communication model prevents them from exchanging useful information. One example of useful information that APs could communicate to non-associated clients would be the AP load, which clients can use to improve their AP selection strategy.

In this paper, we present *beacon-stuffing*, a low bandwidth communication protocol for IEEE 802.11 networks that enables APs to communicate with clients without association. This enables clients to receive information from nearby APs even when they are disconnected, or when connected to another AP. Our scheme is complimentary to 802.11 association and works by overloading 802.11 management frames while not breaking the standard. The beacon-stuffing protocol is based on two key observations. First, clients receive beacons from APs even when they are not associated to them. Second, it is possible to overload fields in the beacon and other management frames to embed data. APs embed content in Beacon and Probe Response frames, while clients overload Probe Requests to send data. We have implemented beacon-stuffing on various wireless cards on Windows XP and Windows Vista. These techniques require minimal driver changes at the clients and APs.

Beacon-stuffing enables a number of new applications, and in this paper we explore three of them. First, APs can embed network selection content into beacons with the beacon-stuffing protocol, for example to broadcast performance or pricing information about their wireless network. The clients can use this information to select either the "best" AP or the best wireless network to connect to. For this application, both clients and APs benefit from being able to exchange information without assocation. Another application of beacon-stuffing is for APs to send location-specific advertisements to nearby clients that are not associated to it. This can be used to advertise network services (e.g. network printing), or real-world goods and services (e.g. a coffee shop or restaurant). Finally, as an extension of location-specific ads, APs may want to provide coupons to nearby clients without requiring association. Beacon-stuffing also provides an alternative mechanism to implement several other location-based applications that have been explored in prior work [5, 6].

In the rest of this paper, we first describe the beacon-stuffing protocol in Section 2. We describe the three applications in more detail in Section 3, our system implementation in Section 4, and finally conclude in Section 5.

2 Mechanisms for Broadcasting Information

Our approach is based on the "push model" of information delivery. The key idea is to overload IEEE 802.11 beacons to carry additional information. Beacon frames are used to announce the presence of a Wi-Fi network. As a result, an 802.11 client receives the beacons sent from all nearby APs, even when it is not connected to any network. In fact, even when a client is connected to a specific AP, it periodically scans all the channels to receive beacons from other nearby APs to keep track of networks in its vicinity. The client does not have to transmit anything to receive the beacons; it merely has to listen. This push model is in contrast to the model currently being used where a client establishes an Internet connection, transmits information about its location (obtained in a variety of ways), and "pulls" information relevant to that location.

Under common environmental conditions, the beacons frames have a range of 100-200 meters. Thus, information carried in these beacons is implicitly "localized". For example, if the AP is located in a restaurant, only clients who are physically in the vicinity of the restaurant will receive the messages (presumably, an advertisement for the restaurant) transmitted by the restaurant AP. Thus, we eliminate the need to explicitly locate the client. By varying transmit power, and encoding scheme, we can further control the range of the beacons. Although beacons are typically sent as the lowest data rate, beacon-stuffing APs may choose to transmit beacons at higher data rates to reduce the airtime utilization or to control the range.

We treat the information to be broadcast as a string of bytes. In most cases, we expect the information to be a short text message. However, our techniques could also be used to deliver non-text information such as short audio jingles. The AP splits the message into smaller fragments, and transmits each fragment in a separate beacon. The size of the fragment depends on the mechanism being used. The fragment sent in each beacon has the following format:

$$UniqueID : SequenceNumber : MoreFlag : InfoChunk$$

$UniqueID$ identifies the message being broadcast, and $SequenceNumber$ is the fragment number, and $MoreFlag$ informs the client if it should expect more fragments: i.e., the last fragment has a value of 0, and all others have a value of 1. Finally, the $InfoChunk$ has the contents of the message. Clients reassemble the message after receiving all fragments.

We now discuss details of three specific techniques to carry the messages in a beacon. The main difference between the techniques is the field in the beacon packet that is used to carry the messages. The format of the 802.11 beacon packet is shown in Figure 1. Commercial APs only allow us to modify the SSID in the beacon packet, and this is one of the techniques that we use. However, if we have access to the source code running on the AP, other fields can be easily modified. Specifically, we can either modify the BSSID, or add an extra Information Element to the beacon. None of the three techniques require any modifications to the hardware or firmware of the client device to receive the messages. For the SSID and BSSID based techniques, a simple user-level application is sufficient to reassemble the fragmented messages. The third technique, which uses Information Element, requires changes to the Wi-Fi driver on the client devices.

(1) **SSID Concatenation:** The SSID field in the Beacon carries the name of the wireless network. The maximum length is 32 bytes. Assuming the $UniqueID$ is 1 byte and $SequenceNumber$ and $MoreFlag$ can fit in 1 byte, we are left with 29 bytes for the $InfoChunk$. Fragments are transmitted in successive beacons. The maximum length of each unique message is 3712 bytes.

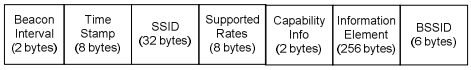

Figure 1: *Some fields in the IEEE 802.11 beacon packet.*

This approach is easy to implement. Most commercial APs provide a user interface to set the SSID. Windows and Linux clients can query the beacon's SSID from user-level, so there is no need for kernel modification on client devices. A simple user-level program is sufficient reassemble the fragments and display the reassembled message.

A primitive version of this approach is already used by a number of retail stores. For example, a hotspot in Starbucks has an SSID of Starbucks. Our approach enables retailers to send longer messages at a reasonable bandwidth. For example, if we set the AP Beacon Interval to 10ms, we can send messages at the rate of 23Kbps.

However, this approach has a significant limitation. Most client devices such as handheld or laptop computers include an application that displays the SSIDs of networks within the range of the device. Unless this application is modified, the user interface of this application will get swamped with the large number of "bogus" SSIDs, which might obscure legitimate SSIDs.

(2) **BSSID Concatenation:** BSSIDs are 6 byte unique identifiers of an AP. Generally, they are set to be equal to the MAC address of the wireless NIC in the AP. However, these can generally be set to any value. Assuming once again that the $UniqueID$ is 1 byte and $SequenceNumber$ and $MoreFlag$ can fit in 1 byte, we can transmit 4 bytes of the message in a beacon. This gives us a maximum length of 512 bytes for each unique message.

All beacons have their SSID set to a fixed, well-known SSID (e.g. $Reserved$). When a client receives a beacon with this SSID, it queries the BSSID field from the user-level. It assembles the message after receiving all fragments.

This approach overcomes the primary limitation of the SSID concatenation. Most network selection UIs do not display multiple entries for the same SSID. Instead, the user is only presented with a list of unique SSIDs, and when the user selects a particular SSID, the driver determines which BSSID to connect to. This scheme does not require any kernel modifications. Windows and Linux both provide APIs that can query the BSSID of a beacon form user level. The only limitation of this approach is the bandwidth. Assuming that beacons are sent every 10ms, the transmission rate is 3.2Kbps.

(3) **Beacon Information Element:** The IEEE 802.11 standard allows AP vendors to add 253 bytes of vendor-specific Beacon Information Elements (BIEs) in its beacon. We use this feature to define a special BIE for broadcasting information. Each message is fragmented into 251 byte chunks and sent in successive beacons. All beacons have their SSID set to a fixed, well-known SSID such as $WiFiAds$. The Wi-Fi driver at client devices are modified to recognize this BIE and pass it to the user level. This approach provides a bandwidth of approximately 200Kbps, which is higher than the other two approaches. It also does not spam the wireless UI of the client device. The main drawback of this approach is that it requires driver modification for client devices, making it relatively difficult to deploy.

For each of the above encoding schemes, messages that span multiple beacons run the risk that if any fragment is lost then the entire message is lost. One can combat this problem by using forward error correction when encoding the message, although we have not yet implemented

this approach. The SSID and BSSID concatenation techniques can also be used by clients to send information back to the AP, embedding this information in Probe Request frames rather than Beacons. We omit the details due to space considerations.

3 Applications

3.1 Network Selection

Currently, end-users make decisions about which wireless network to connect to based on very limited information. For example, the Windows wireless zero-config service simply presents a list of the available network names along with an approximation of the signal strength of an AP from that network. Using the beacon-stuffing mechanisms described above, the beacons can be overloaded to provide a variety of different information that can be useful to end-users in making their network connection decisions. The following kinds of information can be embedded into beacons to help with this process: pricing information, the number of users connected through this AP, other performance-related network utilization information, when the network is open or closed for public use, and other AP selection criteria [7, 8].

3.2 Wi-Fi Advertisements

One compelling application for beacon-stuffing is to use it to deliver location-sensitive advertisements. The delivery of advertisements over the Internet has become a huge market, and one key reason for this success is that the ads are *targeted* – the ads delivered to the end-user are relevant to the search queries typed by that user. We believe that location-sensitive advertisements – ads that are targeted to a user based in part on the physical location of that user will be a very important market in the near future.

Existing approaches to delivering location-sensitive advertisements require two capabilities: the user must have a reasonable quality connection to the Internet, and there must be an automatic method for indicating the user's location to the ad delivery service. Most of the existing systems rely on GPS for location (which doesn't work indoors), and Internet connectivity on-the-go can be both costly and difficult to maintain a reasonable quality-of-service.

Our approach is based on the "push model" of ad delivery. By overloading 802.11 beacons to carry ad messages, any 802.11 client can receive advertisements sent from any nearby AP, regardless of that client's connection status. Even when a client is connected to a specific AP, it periodically scans all the channels to receive beacons from other nearby APs to keep track of other networks in its vicinity. The client does not have to transmit anything to receive the beacons; it merely has to listen.

One benefit of the beacon-stuffing approach is that it is implicitly location-aware, because the ads are sent from Wi-Fi access points whose locations are known and whose "reach" is limited by the propagation of 802.11 signals (typically 100 meters when the access points are placed at ground-level). Our approach works regardless of whether the client Wi-Fi devices are already connected to an existing Wi-Fi network, or the client is completely disconnected from all Wi-Fi networks. It also works in indoor environments such as malls and subway stations where GPS is unlikely to work, and in highly-mobile environments where Internet access will be intermittent at best (e.g. a commuter bus traveling on a highway).

An additional benefit of this approach is that we eliminate the need to explicitly locate the client, which as a side-effect improves the privacy model. Previous approaches to location sensitive advertising require revealing the location of the client to some centralized Internet infrastructure. Our approach to location sensitive ad delivery works in such a way that we do *not* require clients to send messages that explicitly reveal their location.

3.3 Distributing Coupons over Wi-Fi

In addition to broadcasting advertisements, we can also use the beacon-stuffing techniques to broadcast coupons from participating business. A typical usage scenario is: a user with a Wi-Fi device walking near a coffee shop sees an ad for the coffee shop on his Wi-Fi device, and also captures a coupon from the coffee shop. The user then enters the shop and redeems his coupon by displaying it on the screen, at which point the vendor uses a bar code reader to validate the coupon.

Coupons differ from simple advertisements in that the business sending the coupons wants to track certain information. For example, a retailer that sends ads with coupons may want to track where the customers who use the coupons received them, to determine which locations are most effective. A business may also be interested in some form of protection for the coupons it broadcasts. It may want to ensure that the coupons can not be forged by competitors, or it may want to limit the total number of coupons distributed.

We use well known cryptographic techniques to support coupon distribution. Our approach relies on an "Ad service provider"(ASP), who is responsible for distributing the public and private key-pairs to all the entities involved (i.e., the Wi-Fi clients, the APs, and the stores) and who is responisble for online validation during coupon distribution and redemption. Note that the client keys may be encrypted and stored on the user's device when the user installs the client software.

When a user receives a coupon they are interested in, they click a button on the UI to capture it. We use this to trigger the client to send a directed Probe Request to the AP that sent the coupon. The SSID field of the Probe Request frame contains a time-stamp, the user ID, and the client's MAC address, all three of which are encrypted with the ASP's public key. The AP signs this packet and forwards it to the ASP server. The ASP server validates the client's request by matching the user ID with the client MAC address and forms a packet that contains the coupon, the time stamp, and the MAC addresses of the client and AP, and encrypts it with the store's public key. This encrypted packet is sent back by the AP in its Probe Response. When the user tries to redeem the coupon at the store, the store decrypts the coupon packet, and validates the client's MAC address and the coupon.

The above scheme is resilient to various attacks. First, it prevents an attacker from replaying a request to get the coupon, since the attacker not only has to spoof the MAC ID, it also has to learn about the user's ID to cash-in the coupon. The user ID is always encrypted before sending on the air. Second, in a scenario where a store wants to distribute a limited number of coupons, this scheme limits an attacker from exhausting all the coupons. This is achieved by having the user provide a MAC address and user ID information before the coupon is issued. Various other attacks, such as forging coupons, forging user identity, and AP spoofing are also thwarted.

Our approach supports various models for coupon distribution: we enable stores to track the AP that distributed a coupon by looking the AP's MAC address in the encrypted coupon packet; we also allow stores to limit the number of coupons per user device. Finally, it is simple to extend our approach to scenarios where the AP is not connected to the Internet, as long as the store server

where the coupon is redeemed is connected to the Internet.

4 System Implementation

We have implemented all the variations of the beacon-stuffing protocol described in Section 2 on Windows XP and Windows Vista. We implemented the beacon-stuffing AP functionality on Realtek 8185 [2] and Atheros 5523 [1] chipsets. We have also tested the client-side code on cards from 5 different wireless card vendors.

We have two implementations of the beacon-stuffing protocol on the AP side. Our first implementation uses a software AP implementation that works with Windows XP and Vista, but requires Realtek 8185 cards. We modified the software AP driver to include the Information Element in its beacons. Our second implementation uses the Atheros 5523 USB chipset. We modified the this driver to send any user constructed frame. For both implmentations, we built a user-level service that takes as input a byte string to be sent to the clients. Our service fragments the string on the fly, and calls the appropriate API to send the packet (e.g., add the appropriate information element to beacons sent by the Realtek AP, or construct an entire beacon frame with the appropriate information element and send it using the Atheros driver).

Our client-side implementation runs as a user-level service. It periodically queries the driver to check for special beacons sent by the beacon-stuffing protocol. It assembles the fragmented byte string from multiple beacons and displays this information to the user, either in the form of an advertisement, coupon or for AP selection.

5 Conclusion

In this paper, we have described a novel approach for embedding arbitrary content in 802.11 management frames for the purpose of enabling low bandwidth communication without the heavyweight process of association. We have described a few applications of this technique, including delivery of location-sensitive advertisements and coupons over Wi-Fi networks.

References

[1] Atheros Communications. http://www.atheros.com.

[2] Realtek. http://www.realtek.com.tw.

[3] R. Chandra, V. Bahl, and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of the IEEE Conference on Computer Communications (Infocom)*, 2004.

[4] IEEE802.11b/D3.0. Wireless LAN Medium Access Control(MAC) and Physical (PHY) Layer Specification: High Speed Physical Layer Extensions in the 2.4 GHz Band.

[5] J. H. Kang and G. Borriello. Ubiquitous computing using wireless broadcast. In *WMCSA*, December 2004.

[6] J. H. Kang and G. Borriello. Harvesting of location-specific information through wifi networks. In *LoCA*, pages 86–102, 2006.

[7] A. Nicholson, Y. Chawathe, M. Chen, B. Noble, and D. Wetherall. Improved Access Point Selection. In *ACM/USENIX MobiSys*, 2006.

[8] K. Sundaresan and K. Papagiannaki. The need for cross-layer information in access point selection algorithms. In *imc*, pages 257–262, 2006.