

ARBOR: Hang Together rather than Hang Separately in 802.11 WiFi Networks

Xinyu Xing

Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309-0430
Email: xingx@cs.colorado.edu

Shivakant Mishra

Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado 80309-0430
Email: mishras@cs.colorado.edu

Xue Liu

School of Computer Science
McGill University
Montreal, Quebec H3A 2A7
Email: xueliu@cs.mcgill.ca

Abstract—With 802.11 WiFi networks becoming popular in homes, it is common for an end-user to have access to multiple WiFi access points (APs) from residents next door. In general, wireless networks have much higher bandwidth than residential Internet (DSL or Cable) connections. This provides an incentive for an end-user to simultaneously harness bandwidths from multiple APs. This paper introduces ARBOR, an 802.11 driver that aggregates broadband connections in a neighborhood and maximizes Internet access bandwidth in a secure manner. ARBOR has four important characteristics. First, ARBOR can sustain a much longer switching cycle without losing packets queued at different APs. Second, it can schedule traffic loads and (in)directly aggregate AP backhaul bandwidths. Third, ARBOR designs and implements a light-weight authentication mechanism that provides sufficient amount of security, and at the same time, ensures fast switching time. Finally, ARBOR is transparent to the upper layers of the network stack. A prototype of ARBOR has been implemented and extensively evaluated. Experiment results show that ARBOR provides significantly better throughput gains and lower Internet access delays.

I. INTRODUCTION

Availability of high speed Internet access at home has been increasing over the past few years. A recent survey from Point Topic[1] indicates that the number of subscribers using Digital Subscriber Line (DSL) for broadband access worldwide reached 64.5% as of 2008. Indeed, tariffs and packages on offer now-a-days for broadband worldwide provide a general story of increasing speed and decreasing prices. Furthermore, WiFi networks in combination with fixed-line broadband access, have taken off in many American households. This has allowed people the convenience of high speed Internet access anywhere in their homes. However, there is a significant difference between the bandwidths provided by WiFi networks and the fixed-line broadband access such as DSL or Cable. As a result, although WiFi APs provide high speed wireless connectivity¹, WiFi user's throughput is significantly limited by the relatively low-speed DSL or Cable links. Cost of higher speed DSL is quite high. For example, a recent survey shows that subscribers are paying over \$53.32/month on average for a megabit of bandwidth.

Fortunately, pervasive WiFi networks that possess the capacity of Internet access provide broadband subscribers a

¹802.11b/g/n have a maximum raw data rate of 11Mbit/s, 54Mbit/s and 600Mbit/s respectively.

great opportunity to increase Internet bandwidth access at no additional cost². The key idea is to simultaneously harness bandwidths from multiple APs. To do this, two different connectivity approaches have been proposed: (1) A WiFi user utilizes separate wireless adapters to associate with different APs (See Figure 1 (a)) [2][3][4][5]; (2) A software-based approach enables simultaneous connections to multiple APs by virtualizing a single wireless adapter (See Figure 1 (b)) [6][7].

While the first connectivity approach has the advantage of simplicity, it suffers from several limitations [7]. First, the number of APs that can be accessed simultaneously in this approach is limited by the number of wireless adapters that a computing device has. Typically, the number of wireless adapters a device supports is much less than the number of available APs. Second, most WiFi devices, especially handheld, small devices typically do not support multiple adapters due to limitations in physical dimensions, power, etc. Finally, WiFi cards close to each other may experience radio interference causing increased errors in transmission or reception. This paper presents ARBOR (AggRegate ap Backhaul capacity in a neighborhood), an 802.11 driver that aggregates broadband connections in a neighborhood and maximizes Internet access bandwidth in a secure manner. ARBOR is based on the second connectivity approach.

Although [7] demonstrates several benefits of connection to multiple Internet channels via a single WiFi adapter, there are two primary challenges that we believe have not been adequately addressed. First, security mechanisms defined in 802.11i specification incur tremendous difficulty in designing and implementing this approach. In particular, a wireless adapter has to authenticate itself every time it switches from an associated AP to a not-associated one. FatVAP[7] results in performance gains in an *open* 802.11 WiFi environment. It also discusses the possibility of adding WEP to FatVAP, although we are not aware if this has been done. However, unencrypted or WEP-based WiFi networks can be effortlessly monitored and expose data transmitted over the networks

²According to our field study at Bear Creek Apartment in the University of Colorado and McGill University Macdonald Campus, 7.8 WiFi networks on average were detected and more than 80% of them possess the capacity of Internet access.

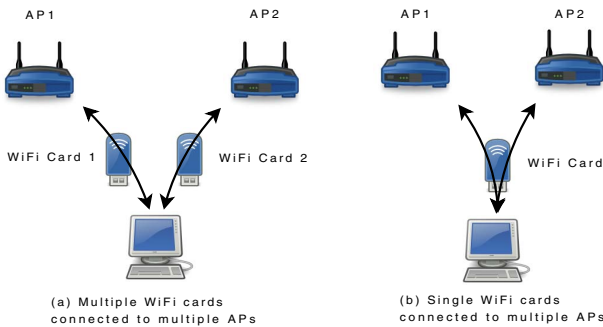


Fig. 1. Primary connectivity approaches: (a) Using multiple wireless adapters to connect to multiple APs (many-to-many); (b) Using a single wireless adapter to switch between APs (single-to-many).

[8]. For stronger authentication and security, IEEE 802.11 RSN uses the Extensible Authentication Protocol (EAP) as well as 802.1X for the authentication phase of establishing a Robust Security Network Association (RSNA). During this authentication procedure, EAP/802.1X involves several cryptographic computations and access control interactions between a supplicant and authentication server [9]. As we show later, this significantly increases the switching overhead and thus limits the advantages of switching between multiple APs. Furthermore, traversing Internet can result in unexpected delays, if the authentication server is far away from the supplicant [10].

Second, to maximize the net Internet access bandwidth, an optimal traffic scheduling mechanism needs to be defined in a wireless adapter. This scheduling mechanism determines which channel (AP) the adapter should associate with at different time intervals over a given time period. This schedule critically depends on the end-to-end available bandwidths on each channel. Consequently, the key challenge is how to efficiently estimate available end-to-end bandwidth on each channel. Current bandwidth measurement methodologies are not suitable for this purpose, because they either rely on a coordination between the two end hosts (IGI[11], Spruce[12], Pathload[13], TOPP[14], SLoPS[15]), or transferring large files over the network ([16][17]). We need suitable bandwidth measurement methodologies that can aid in designing this traffic scheduling mechanism.

ARBOR addresses these two challenges, and enables a WiFi user to (in)directly stripe traffic over multiple APs and maximize Internet access bandwidth in a secure manner. ARBOR has four important characteristics. First, it can sustain a much longer switching cycle without losing packets queued at different APs. Second, it can schedule traffic loads and (in)directly aggregate AP backhaul bandwidths. Third, ARBOR designs and implements a light-weight authentication mechanism that provides sufficient amount of security, and at the same time, ensures fast switching time. Finally, ARBOR is transparent to the upper layers of the network stack. A prototype of ARBOR has been implemented and extensively evaluated. Experiment results show that ARBOR provides

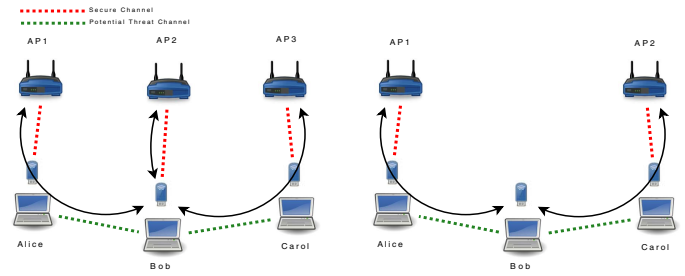


Fig. 2. Bob aggregates bandwidth via a direct connection to AP2 and indirect connections to AP1 and AP3 via his neighbors, Alice and Carol (left); Bob aggregates bandwidth using indirect connections to AP1 and AP3 via his neighbors, Alice and Carol without any direct connection to any AP (right).

significantly better throughput gains and lower Internet access delays.

The rest of the paper is organized as follow. Section II presents the motivation of ARBOR. In Section III, we describe the design of ARBOR in detail. In Section IV, we describe our prototype implementation and performance evaluation. Section V presents our discussion and future work. Finally, Section VI concludes the paper.

II. MOTIVATION

In order to aggregate Internet bandwidth over multiple APs, maximize throughput and avoid losing queued packets on each AP, the connectivity approach shown in Figure 1 (b) requires a wireless adapter to frequently switch between different channels in short time scales. To better understand the reason, let us look at a simple example shown in Figure 1 (b). Assume that the buffer capacity of the APs is 125 KB, each AP connects to a DSL cable with 10 Mbps and wireless available bandwidth is 30 Mbps on both WiFi networks. Ignoring the switching overhead, buffers at one AP may overflow if the wireless adapter continuously spends more than 100 ms on another AP³. So, to prevent buffer overflow, the switching cycle must be less than 200 milliseconds.

We performed an experiment to measure the time it takes for a wireless adapter to authenticate and associate with an AP for different 802.11i specification. Results are shown in Table I. Notice that even in the simplest case of ‘EAP-TLS over LAN & AS and AP connecting to the same switch’ a wireless adapter will have only 86 ms of time left, i.e. only 43% for data transfer while switching from associated AP1 to not-associated AP2 within 200 ms of switching cycle. Furthermore, switching times in cases of EAP-TLS over Internet are so large that FatVAP is simply infeasible in those cases. Therefore, the key of maximizing Internet bandwidth significantly depends on the capacity of queues where packets are buffered and the amount of switching overhead that includes security delays. A solution such as FatVAP is not suitable due to limited buffer size on an AP, e.g. D-Link wireless G router WBR-1310 can only buffer approximately 200 KB for a WiFi user that enters the power save mode.

³For simplicity, assume 1 Mbps = 1000 Kbps

TABLE I
 LATENCY OVERHEAD OF VARIOUS OPERATIONS IN MILLISECONDS FOR DIVERSE 802.11 AUTHENTICATION & ASSOCIATION MECHANISMS

Operation	Mean (ms)	Std. Dev. (ms)	Min. (ms)	Max. (ms)
Open mode	3.456	0.41317	3.022	4.341
64-bit WEP	5.17642	0.98522	3.382	7.581
128-bit WEP	5.26367	1.51298	3.166	8.816
EAP-TLS over LAN & AS and AP connecting to the same switch	113.583	14.829	104.152	139.077
EAP-TLS over LAN & AS and AP connecting to the same LAN	183.003	56.506	125.127	268.560
EAP-TLS over INTERNET & AP and AP in UIUC region	995.231	1223.06	267.144	6314.26
EAP-TLS over INTERNET & AP in UIUC and AS in America region	1718.37	4150	355.464	45738
EAP-TLS over INTERNET & AP in UIUC and AS in Asia region	5788.53	6969.88	1363.82	26279.4
EAP-TLS over INTERNET & AP in UIUC and AS in Europe region	2227.47	2973.44	929.967	24868.8

III. ARBOR

The primary goal of ARBOR is to schedule traffic over multiple APs and maximize Internet access bandwidth in a secure manner. While ARBOR uses a single wireless adapter for bandwidth aggregation, it differs from earlier approaches in that it connects to multiple neighboring hosts that in turn connect to different APs (see Figure 2). Because the length of the queues on neighboring hosts is much larger than the queue on an AP, ARBOR is able to sustain a much longer switching cycle during which a user completes authentication procedures and has plenty of usable time left in his schedule for data transmission. In addition, ARBOR is able to use a customized security mechanism between neighboring hosts that is significantly lighter weight than EAP-TLS. This security mechanism employs public-key cryptography to establish a pair of symmetric keys and thus protects data transmission between hosts.

A. Overview of ARBOR

At a high level, ARBOR works as follows.

1) *Probing passively*: Neighboring hosts (contributors) willing to contribute their Internet access bandwidths probe corresponding APs, estimate their Internet access bandwidth and then announce their backhaul bandwidths. An ARBOR host scans various channels searching for contributors⁴. In addition, the ARBOR host probes its AP and estimates its AP backhaul Internet bandwidth if it connects to an AP.

2) *Scheduling in a neighborhood*: Based on Internet access bandwidth that ARBOR obtains from each contributor and the AP that the ARBOR host connects to, it calculates and announces a schedule that indicates when it will connect to different contributors and for how long.

3) *Adapter switching and striping traffic*: Based on this schedule, the ARBOR host switches its wireless adapter between different channels and stripes traffic over different contributors. If the ARBOR host connects directly to an AP, some data will be transmitted through the AP. Furthermore, to make an ARBOR host obtain Internet access bandwidth correctly, each contributor also switches its own wireless adapter between the ARBOR host and its AP by following the schedule that the ARBOR host announced.

⁴This paper assumes the sum of Internet available bandwidth is less than wireless available bandwidth.

4) *Establishing a secure channel*: To establish a secure channel between an ARBOR host and each contributor, ARBOR performs a light-weight authentication procedure when the ARBOR host connects to a contributor for the first time. This authentication procedure assumes that a public/private keys have been installed on the ARBOR host and contributors. By adopting asymmetric cryptography techniques, a pair of symmetric keys are established between an ARBOR host and each contributor.

B. Probing passively

Before contributing Internet access bandwidth to ARBOR hosts nearby, contributors have to probe their corresponding APs and estimate their Internet access bandwidth. In addition, the ARBOR host also needs to probe its AP and estimate its backhaul Internet access bandwidth as well as the available bandwidth over wireless medium. We use the methodology introduced in [7] for passive measurement of wireless available bandwidth. However, estimating AP backhaul Internet access bandwidth is challenging for ARBOR hosts and contributors. Due to limited switching cycle on ARBOR, earlier approaches for estimating end-to-end available bandwidth (Pathload, Spruce and abget etc.) are not suitable. For example, we used abget to estimate end-to-end available bandwidth of 123 websites. As shown in Figure 4(a), none of the measurements completed with in one second. The methodology used in FatVAP measures an average of available bandwidths over all paths through an AP, and is again not suitable. To address this problem, ARBOR harnesses a novel methodology - ABODE [18] - to estimate backhaul Internet access bandwidth accurately and efficiently.

Based on our observation that the tight link of the Internet in the context of a wireless 802.11 environment is usually on the edge of the Internet [18], the estimation methodology - ABODE - works by having the sender (either a contributor or an ARBOR host) send ICMP probe packets starting at a certain rate and increasing this rate at regular intervals. Round trip time (RTT) is measured for every send rate. When the send and reply rates of ICMP messages is less than the Internet access bandwidth, the RTT remains approximately constant (see Figure 3(a)). On the other hand, when the send rate goes above the Internet access bandwidth, RTT starts increasing (see Figure 3(b)). Therefore, ABODE estimates the available bandwidth by observing sending echo request

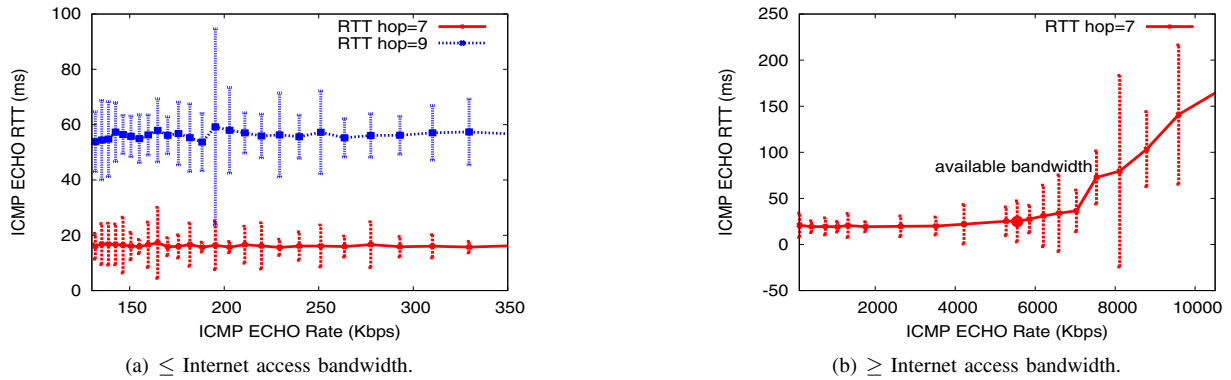


Fig. 3. Variation in RTT with varying ICMP send rate.

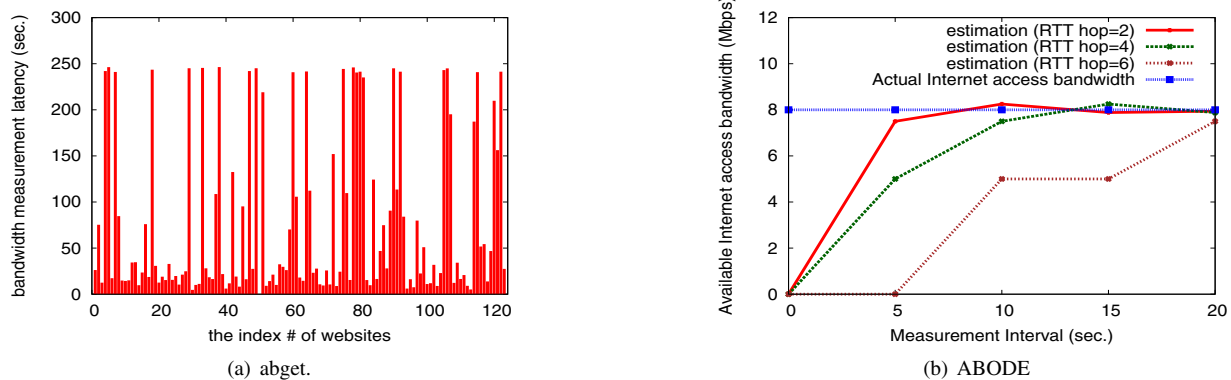


Fig. 4. Bandwidth measurement in abget and ABODE.

messages at increasing rates and observing the value of RTT. The available bandwidth is estimated to be the send rate at which RTT starts to increase. ABODE utilizes the binary search algorithm ($O(N \cdot \log N)$) to locate this turning point. There are two important advantages of ABODE over the earlier bandwidth measurement technologies that make it well-suited for ARBOR. First, it does not require access to hosts at both ends of a path and it does not rely on an end-to-end TCP connection. Second, it does not require large data transfer over an entire communication path. We have experimented extensively with ABODE. Figure 4(b) illustrates one such experiment, where the number of hops is two and available Internet access bandwidth was set to 8 Mbps. For comparison, we have also included how abget performs under the same scenario (recall that abget requires large file transfer over the entire communication path).

C. Scheduling in a neighborhood

Based on the backhaul Internet access bandwidths of different contributors, the backhaul Internet bandwidth of AP that the ARBOR host directly connects to, and available wireless bandwidth, ARBOR calculates a schedule. This main goal of this schedule is to ensure that the buffer at the AP that the ARBOR host directly connects to doesn't overflow, and at the same time the ARBOR host is able to associate with each contributor long enough to download all data from it. Given a

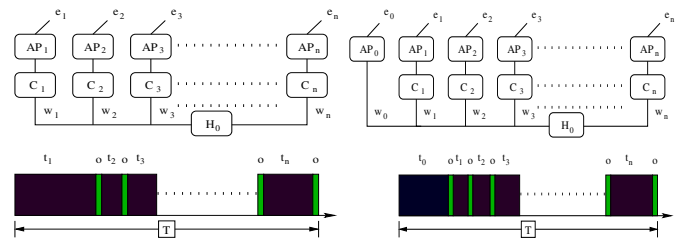


Fig. 5. Illustration of scheduling traffic over multiple APs (in)directly.

set of contributors $\{C_1, C_2, \dots, C_n\}$ and the corresponding AP set $\{AP_1, AP_2, \dots, AP_n\}$, we want to aggregate Internet access bandwidths over all the APs. Let e_i represent Internet access bandwidth of AP_i , and w_i be the wireless bandwidth between the ARBOR host H_0 and contributor C_i ($i = 1, 2, \dots, n$). In addition, let t_i be the time interval during which H_0 connects to C_i . If H_0 has a direct connection to an AP (say AP_0), e_0 , w_0 and t_0 then denote the Internet access bandwidth of AP_0 , wireless bandwidth between H_0 and AP_0 and the time interval during which H_0 connects to AP_0 respectively. When a direct connection occurs between H_0 and AP_0 , we need to further consider the size of the buffer on AP_0 where incoming packets are buffered while H_0 enters the power saving mode. Let Q_0 be the size of this buffer. Also, o be the switching overhead of switching from one device to another. Recall that there will

be less time left for data transfer in the whole switching cycle T when the switching overhead o is large.

Suppose that H_0 connects to multiple contributors one by one following the sequence number of the contributors (see Figure 5 left). Then, the whole switching cycle can be formalized as

$$T = \sum_{i=1}^n (t_i + o) \quad (1)$$

If H_0 directly connects to AP_0 (see Figure 5 right), then the whole switching cycle is

$$T = \sum_{i=0}^n (t_i + o) \quad (2)$$

H_0 needs to associate with each contributor for sufficient time to be able to download all data from that contributor. This means the amount of data downloaded by a contributor C_i from the Internet should not exceed $t_i \cdot w_i$. When an ARBOR host connects to a contributor or an AP, the subsequent contributors continue to maintain data download from the Internet. For example, when H_0 is connected to C_i , the subsequent contributors ($C_{i+1}, C_{i+2}, \dots, C_n$) will keep downloading data from the Internet via their respective APs. Therefore, the time interval during which H_0 connects to C_i can be calculated by using the following equation:

$$t_i = \frac{e_i \cdot \sum_{j=1}^{j=i-1} (t_j + o)}{w_i} \quad (3)$$

Similarly, if there exists a wireless connection between H_0 and AP_0 , the time interval is

$$t_i = \frac{e_i \cdot \sum_{j=0}^{j=i-1} (t_j + o)}{w_i} \quad (4)$$

Also, to ensure that the buffer on AP_0 doesn't overflow, the following condition must hold:

$$\frac{Q_0}{e_0} \geq \sum_{j=1}^n t_j + o \quad (5)$$

Using these equations, ARBOR calculates a schedule and announces it to all contributors. It waits for a time interval Δt which is equal to the maximum transmission delay, i.e. $\Delta t = \max t_{d_i}$, where t_{d_i} denotes the transmission delay between H_0 and C_i , to ensure that all contributors have received the schedule. Note that wireless bandwidth measurement that ARBOR uses can indirectly estimate transmission delays [7]. In addition, since the contributors and the ARBOR host have to follow the same schedule, they must have their clocks synchronized⁵. Also, note that the switching overhead o contains overheads due to active bandwidth measurement overhead (III-B), switching overhead (III-D) and authentication overhead (III-E). Switching cycle in ARBOR is usually set to $\leq 1000ms$ to ensure that TCP connection doesn't time out.

⁵A number of techniques exist to synchronize the hosts in a mesh network. These techniques are outside the scope of this paper.

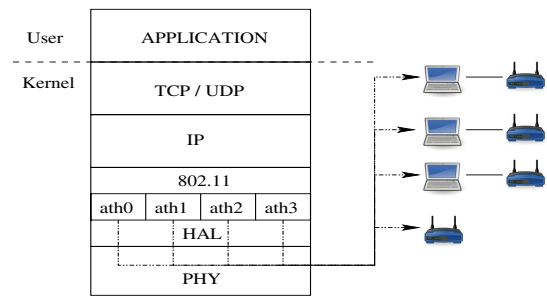


Fig. 6. ARBOR: a wireless adapter connects to three neighbor hosts and one AP via four different interfaces (ath0-ath2 work in adhoc mode, whereas ath3 operates in managed mode).

D. Switching a wireless adapter and striping traffic

Based on this schedule, an ARBOR host not only stripes traffic over different contributors and AP_0 , but also enables their wireless adapters to toggle between different channels so as to obtain AP backhaul Internet bandwidth (in)directly.

1) *Striping traffic*: In general, a TCP-based application, especially an HTTP application, attaches a unique destination IP address at a time. Therefore, splitting traffic based on different destination IP addresses is not practical, especially when an Internet user just attempts to request a web page from a website. ARBOR allocates traffic on a flow basis. A flow is identified by its destination IP address and its ports. An HTTP response usually contains several different flows, e.g. an HTTP response from cnn.com includes 7 different flows.

An 802.11a/b/g adapter has multiple interfaces and each interface (in)directly associates with a different AP (see Figure 6). Assigning different flows to different interfaces means (in)directly assigning different flows to different APs. ARBOR creates a flow-to-interface mapping in a hash table. Each entry in the hash table is a 2-tuple that contains information about an interface and a flow. Initially, this table contains information about all interfaces. When a new flow arrives, ARBOR randomly decides which interface to assign the flow to and records the assignment in the hash table. Subsequent packets in the flow are simply sent through the interface recorded in the hash table. Because ARBOR splits traffic randomly, the probability of assigning a group of same flows to each interface is approximately equal.

2) *Switching a wireless adapter*: An ARBOR host and its contributors have to enable their wireless adapters to toggle between channels. Based on the schedule that the ARBOR host generated, not only a contributor guarantees to stay on the corresponding channel when the ARBOR host is in the contributor's network, but also it ensures its wireless adapter to immediately switch back to managed mode when the ARBOR host is out of the network. In our implementation, ARBOR harnesses two different strategies to achieve loss-free switching: inside the kernel and at the user level.

At the user level, ARBOR creates an interface working in a certain operation mode (adhoc or managed) at boot time. When ARBOR switches the interface from one mode to other, or switches the interface from one contributor to another, it

TABLE II
 SWITCHING OVERHEAD OF A WIRELESS 802.11 ADAPTER.

Operation	Mean (ms)	Std. Dev. (ms)
Managed→Adhoc (User level)	28.333	1.168
Adhoc→Managed (user level)	24.121	2.441
Adhoc→Adhoc (user level)	24.879	1.485
Managed→Adhoc (Kernel level)	3.749	0.425
Adhoc→Managed (Kernel level)	3.045	0.115
Adhoc→Adhoc (Kernel level)	3.376	0.362

retrieves the buffered data on the queues which attach to the interface, informs the device that it is entering the power save mode, destroys the interface as well as the corresponding queues, and creates a new interface and corresponding queues. In our implementation, this involves approximately 24-28 milliseconds of switching delays (see Table II).

ARBOR implementation inside the kernel creates different interfaces and queues at boot time. One interface works in managed mode and others operate in adhoc modes. Generally, each of the interfaces associates with a different machine (a contributor or an AP) (see Figure 6). Note that some of the interfaces may appear unassociated if the number of interfaces is greater than the number of machines. When ARBOR switches from one interface to another interface, it retrieves the buffered data from the queues attached to the interface, informs the machine that it is entering the power save mode, detaches the interface and the corresponding queues, and attaches to another interface and its queues. This implementation inside the kernel incurs significantly low switching delays which are approximately 3-4 milliseconds (see Table II). Therefore, we have adopted the switching strategy inside the kernel in our prototype implementation of ARBOR.

In addition to the switching delays, some minor, non-negligible delays are incurred. A contributor has to properly forward packets when it receives incoming packets from its AP or outgoing packets from the ARBOR host. As shown in Figure 6, each interface is associated with a different contributor or AP, and uses a different IP address. Therefore, when receiving a packet either from an ARBOR host or from an AP, ARBOR has to change the address in the packet so that the packet can arrive at its destination. IP checksum and the TCP/UDP checksum are recomputed after ARBOR replaces the original IP address with a new one. In addition, searching in hash table and running scheduling algorithm results in slight increase in switching overhead. We measured these delays to be less than 3 milliseconds on average.

E. Establishing a secure channel

Finally, as shown in Figure 2, encryption-free channels appear between the ARBOR host and each contributor. Due to the fact that ARBOR can sustain a longer switching cycle, most authentication mechanisms defined in 802.11i specification can be used. However, these authentication mechanisms are expensive. Consider the following scenario. The ARBOR host switches its wireless adapter between three neighbor hosts. Ignoring other switching overheads, if ARBOR utilizes

EAP-TLS authentication mechanism over LAN (AP and AS connecting to the same LAN) and the switching cycle is $T = 1000ms$, ARBOR would only have approximately 451 ms of usable time left. To address this, ARBOR we incorporate a light-weight authentication mechanism that harnesses public-key cryptography to jointly establish a shared secure key over an insecure communication channel. This authentication mechanism consists of four steps.

Step 1: The ARBOR host sends to a contributor an authentication request frame to perform asymmetric-cryptography authentication. This request frame encapsulates the identification of the ARBOR host (id_{ARBOR}), Authentication Transaction Sequence Number (ATSN), and Timestamp.

Step 2: The contributor responds with an authentication response frame. The response frame encapsulates the certificate of the contributor ($Cert_{Contributor}$) and a ticket. The ticket consists of three elements: The identification of the contributor ($id_{Contributor}$), the timestamp shown in the authentication request frame that the contributor received from the ARBOR host and a Message Integrity Check (MIC).

Step 3: Based on the response frame that it received, the ARBOR host verifies the identification of the contributor and generates a Pre-Session Key (PSK) encrypted using the public key of the contributor (PK). The ARBOR host then sends the contributor a frame that encapsulates the encrypted pre-session key and the ticket.

Step 4: On receiving the ticket and pre-session key, the contributor verifies the ticket through the timestamp and MIC. If it is valid, the contributor calculates the symmetric session key and sends a response frame containing an Authentication State Code (ASC) with the value "success". Otherwise, it sends a response frame with authentication failure message.

We have implemented this authentication mechanism in ARBOR. It consumes about 37.545 ms. Compared with the authentication delays of EAP-TLS over LAN/Internet, our light-weight authentication mechanism allows ARBOR to utilize a much larger time period for data transfer. This mechanism can still obtain lower authentication latency despite the fact that the computation of cryptographic keys is more complicated in our mechanism. The main reason for this is that the number of messages exchanged in our mechanism is much less than the number of messages EAP-TLS over LAN/Internet exchanges (4 vs. >14).

Two primary points are worth noting: (1) We assume that each contributor has installed a public-private key pair. The installation of the public-private keys can be achieved by using either a Certificate-Authentication-signed certificate or a self-signed certificate. We limit the scope of this paper to the study of the self-signed certificate only. (2) The light-weight authentication mechanism can guarantee security against spoofing and eavesdropping because different symmetric key pairs are generated between the ARBOR host and each contributor. However, there is a threat of insider attack due to self-signed certificate. This threat can be eliminated by using a Certificate-Authentication-signed certificate for each contributor.

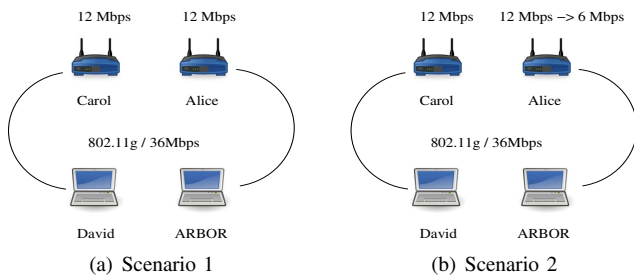


Fig. 7. ARBOR simultaneously connects to David and Alice to aggregate AP backhaul Internet bandwidth.

IV. EVALUATION

To evaluate the performance of ARBOR, we have performed a number of experiments. These experiments focus on two factors - exploring throughput behaviors and comparing the latency of TCP-based applications. Experimental results demonstrate the correctness and benefits of ARBOR.

A. Experimental setup

Based on the madwifi v0.9.4 [19] driver, we implemented ARBOR on 3com wireless LAN PC adapters (Atheros AR5213). Each wireless adapter is deployed on a laptop with Linux v2.6.24 installation. Some laptops serve as contributors and others as ARBOR hosts. To make all laptops connect to different wireless networks, D-Link WBR-1310 wireless G routers are deployed in our experimental environment. Each of the wireless routers is assigned to different 802.11g channels. A D-Link WBT-1310 router can buffer up to 200 KB of data for a WiFi user that enters power save mode.

In general, DSL broadband services provide 1.5 - 24 Mbps Internet access bandwidth. To emulate this AP backhaul link, we deployed a traffic shaper behind each of wireless routers. The traffic shaper is equipped with two gigabit Ethernet NICs, one of which connects to our D-Link wireless router and the other is connected to the Internet through an Ethernet cable at Gigabit speeds. We simply shaped download and upload links to an equal value.

B. Testbed

To demonstrate the benefits of ARBOR, we deployed an indoor testbed in our system lab at CU-Boulder. The main components are two laptops - ThinkPad R52 and DELL 9300 - equipped with Atheros wireless adapters, two PCs - HP Compaq dx 2000 Microtower and DELL OptiPlex 960 - serving as traffic shapers, and two 802.11 wireless routers. Note that since the raw bit rate of Atheros wireless cards is set to 36 Mbps, the theoretical maximum throughput is 24 Mbps [20]. Based on the parameters shown in Figure 7, we configured two different experimental environments.

C. Exploring throughput behaviors

In the first experimental environment, we perform an experiment with an ARBOR client, and repeat it with an unmodified client that connects to Alice. In both cases, we use iperf to generate large TCP flows, each of which lasts for 5

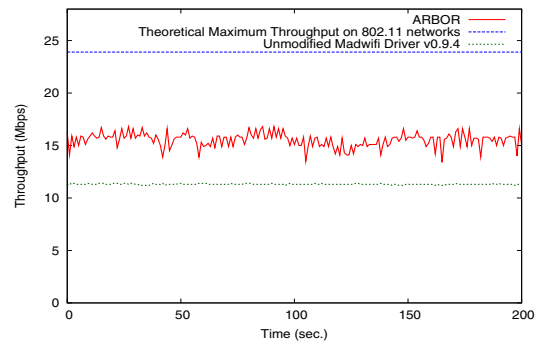


Fig. 8. Throughput comparison (an unmodified madwifi driver limits the throughput to 12 Mbps, whereas ARBOR provides 15.5 Mbps via aggregation.)

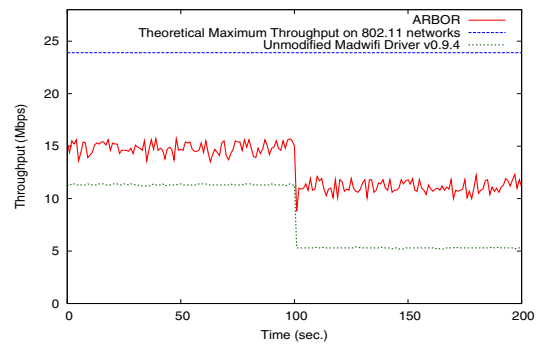


Fig. 9. Throughput comparison (As Alice's backhaul Internet access bandwidth varies from 12 to 6 Mbps, ARBOR is still able to provide 11.1Mbps throughput.)

minutes. As illustrated in Figure 8, ARBOR allows a client to aggregate 15.5 Mbps of Internet access bandwidth on average when switching cycle is $T = 400ms$, whereas unmodified madwifi driver limits the client to 12 Mbps of Alice's Internet access bandwidth. Also, notice that the throughput of ARBOR fluctuates slightly over the 200-second interval compared with the throughput behavior of unmodified madwifi. This is due to the impact of switching overheads.

In our second experiment, we use the same configuration as our first experiment, except that Alice's backhaul Internet ac-

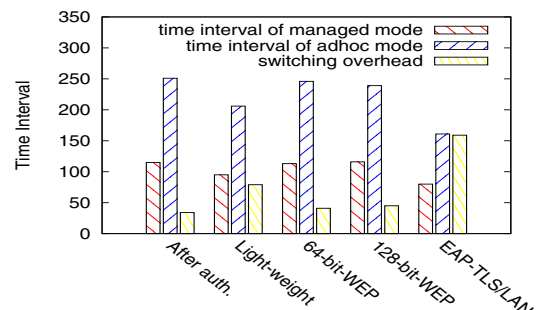


Fig. 10. Allocation of switching cycle under different authentication mechanisms.

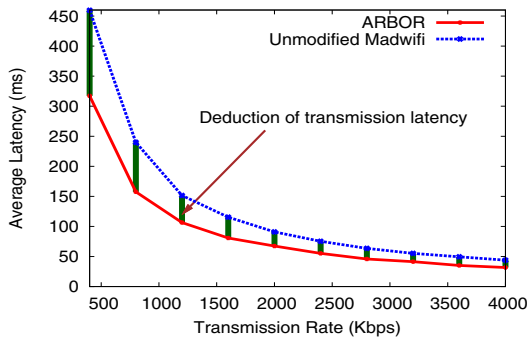


Fig. 11. A client downloads a 20 MB file from a remote server by using an unmodified madwifi and ARBOR respectively.

cess bandwidth is changed to 6 Mbps at time $t = 100s$. Simply repeating the experimental procedure in the first experiment, Figure 9 shows that if the client utilizes an unmodified driver, his throughput changes from 12 Mbps to 6 Mbps. However, ARBOR still provides 11.1 Mbps Internet access bandwidth on average.

While exploring the throughput behavior, we also recorded the actual allocation of switching cycle. When ARBOR host first connects to David’s network, an authentication and association procedure is performed. By using different authentication mechanisms, we notice that light-weight authentication mechanism provides larger time for useful data transmission in client’s schedule when compared with EAP/TLS over LAN (see Figure 10). Since both 614-bit and 128-bit WEP just involve slight delays, total switching overhead for both mechanisms are significantly lower than the light-weight and EAP/TLS over LAN authentication mechanisms. After connecting to David’s network, ARBOR does not require authentication and association any more; therefore, the total switching overhead tends to be a lower value.

D. Latency of TCP-based applications

Based on the testbed we deployed and a switching cycle ($T = 80ms$), we further conducted two experiments to compare the latency of TCP-based applications on ARBOR with the latency on an unmodified madwifi driver.

In our first experiment, a TCP client repeatedly downloads a 20 MB file from a remote file server by utilizing an unmodified madwifi and ARBOR respectively. Tuning Alice and Carol’s backhaul bandwidth from 400 Kbps up to 4000 Kbps, we find that the average latency for downloading a 20 MB file is significantly lower for ARBOR (see Figure 11). Latency difference is more pronounced that when the end-to-end bandwidth is low.

In the second experiment, we picked six websites at random from the top 100 websites listed at Alexa.com and tuned our experimental environment in accordance to the configuration shown in Figure 7(a). We then measured the number of flows for each main page of the websites by using tcpdump. As shown in Table III, each main page has more than one flow. So, we expect a lower latency if we request a webpage by

TABLE III
 LATENCY OF TCP-BASED APPLICATIONS (IN MILLISECONDS)

Website	Madwifi	ARBOR	# of Flow	% Reduction
craigslist	896	588	5	34.38%
cnn	276	149	3	46.01%
ebay	1686	987	8	41.46%
youtube	455	276	4	39.34%
facebook	2271	1232	5	45.75%
wikipedia	3171	2739	3	13.62%

using ARBOR. Table III shows that ARBOR does result in reduced latency, 14% wikipedia.org to 46% in cnn.com and facebook.com. To understand this further, we analyzed the traces of tcpdump. Figure 12 illustrates the assignment of flows for each website and the volume of each flow. Taking facebook.com as an example, when about half of the packets are directly assigned to Alice and rest of them indirectly traverse via Carol, the latency of requesting a webpage from facebook.com is approximately reduced by 50%. The possible causes of the slight difference between 45.75% and 50% are ARBOR’s switching and a handful of packet retransmissions. Furthermore, the number of packets in a certain flow only approximately demonstrates the volume of data in the flow, i.e. a direct result of the slight difference in latency. Therefore, the causes of latency reduction are flow assignment as well as the volume of data stuffed into each flow.

V. DISCUSSION AND FUTURE WORK

1) *Additional functions* The primary function of ARBOR is to aggregate AP backhaul Internet bandwidth in a secure manner. In addition, it can be used for several other purposes. First, like Multinet [6], a user can connect his/her wireless adapter to an adhoc network while staying on an infrastructure network. This can help a client debug poor connectivity [21][22]. Second, the range of an infrastructure network can be extended using ARBOR. Third, a user can connect different virtual machines to physically different wireless networks. Finally, a host that is part of an adhoc network and is close to an AP connected to the Internet can serve as a gateway for the adhoc network.

2) *Heterogeneous platforms* ARBOR utilizes a single WiFi card to maximize bandwidth aggregation while running on diverse WiFi-supported devices. This is in contrast to using multiple WiFi cards on a single device. As discussed earlier, this latter approach suffers from several drawbacks, such as device size and power limitations to support multiple cards in handheld devices, and RF interference and possibility of power leakage, particularly if the antennas are placed very close. In this respect, ARBOR provides an opportunity to extend throughput aggregation to heterogeneous platforms in a secure manner.

3) *Robust incentive mechanisms* An important concern in using ARBOR is “What is the incentive for different users to share their bandwidth, and more importantly how can we ensure that all users participate in bandwidth sharing in a fair manner?”. Large number of selfish WiFi users can

