Context Caching using Neighbor Graphs for Fast Handoffs in a Wireless Network [*]

Arunesh Mishra, Min-ho Shin, and William A. Arbaugh

Computer Science Technical Report CS-TR-4477, and
UMIACS Technical Report UMIACS-TR-2003-46
Department of Computer Science
University of Maryland
College Park, Maryland 20742
waa@cs.umd.edu

Abstract. User mobility in wireless data networks is increasing because of technological advances, and the desire for voice and multimedia applications. These applications, however, require handoffs between base stations to be fast to maintain the quality of the connections. Previous work on context transfer for fast handoffs has focused on reactive, *i.e.* the context transfer occurs after the mobile station has associated with the next base station or access router, methods. In this paper, we describe the use of a novel and efficient data structure, neighbor graphs, which captures dynamically the mobility topology of a wireless network as a means for pre-positioning the station's context at the potential next base stations– ensuring that the station's context remains one hop ahead. From experimental and simulation results, we find that the use of neighbor graphs reduces the layer 2 handoff latency due to reassociation by an order of magnitude from $15.37ms$ to $1.69ms$, and that the effectiveness of the approach improves dramatically as user mobility increases.

1 Introduction

Wireless networks, specifically those based on the IEEE 802.11 standard (Wi-Fi), are experiencing rapid growth due to their low cost and unregulated bandwidth. As a result of this tremendous growth, pockets of connectivity have been created not unlike those created during the first few years of the cellular systems. The logical next step for Wi-Fi based networks is support for fast roaming within the same administrative domain and then eventually between overlapping pockets of connectivity or different administrative domains. Thus, we expect users to become more mobile once technological advances such as multi-mode (Wi-Fi and GSM/CDMA cellular) handsets become more available much as users became more mobile in the traditional cellular networks once handsets became smaller and more affordable.

Previous studies of wireless network mobility have shown that users tend to roam in what we call *discrete mobility* where the user utilizes the network while stationary (or connected to the same base station) and before moving the user ceases operation only to continue using the network after moving to a new location [1–4]. That is the users do not usually move while using the network because the majority of current network applications and equipment do not easily lend themselves to what we call *continuous mobility* where the user moves while utilizing the network.

Voice based applications are the usual application in *continuous mobility* as seen in the current cellular networks, and we expect voice and multimedia applications will serve as the catalyst for *continuous mobility* in Wi-Fi networks much as they did for the cellular networks once multi-mode handsets and end-user applications become more widely available.

Supporting voice and multimedia with continuous mobility implies that the total latency (layer 2 and layer 3) of handoffs between base stations must be fast. Specifically, the overall latency should not exceed $50\ ms$ to prevent excessive jitter [5]. Unfortunately, the vast majority of Wi-Fi based networks do not currently meet this goal with the layer 2 latencies contributing approximately 90% of the overall latency which exceeds $100\ ms$ [6, 7]. Handoffs involve transfer of station *context*, which is the stations's session and QoS related state information, via inter-access point communication. This transfer only furthers the handoff delay by an average 15.37 ms.

One method of reducing the overall latency of handoffs is to transfer or cache context ahead of a mobile station in a *pro-active* fashion. Unfortunately, the previous work on context transfer has focused solely on *reactive* context transfers, *i.e.* the context transfer is initiated only after the mobile station associates with the next base station or access router resulting in an overall increase in the latency of the handoff rather than reducing it [7, 8]. The problem with *pro-active* approaches, however, is how to determine the set of potential next base stations without examining the network topology and manually creating the set.

In this paper, we introduce a novel and efficient data structure, the neighbor graph, which dynamically captures the mobility topology of a wireless network through real-time examination of the handoffs occurring in the network in either a distributed fashion, *e.g.* at a base station or access point, or in a centralized fashion, *e.g.* at the authentication server.

A neighbor graph is an undirected graph with each edge representing a mobility path between the vertices, or access points. Therefore, given any edge, e, the *neighbors* of e represent the set of potential next access points. While there are numerous uses for this information, we focus in this paper on using it for *pro-active* context transfers ensuring that a mobile station's context is always one hop ahead.

To demonstrate the utility of neighbor graphs, we have implemented neighbor graphs on top of the IEEE's Inter-access point protocol and our approach has been included into the latest IEEE draft (IAPP) [9], and we find that using neighbor graphs to pro-actively cache a station's context reduces the latency of reassociation from an average of 23.58 ms (with a single outlier) and 15.37 ms (without the outlier) to 1.69 ms. Further, we find through simulations that as users become more mobile the effectiveness of our solution increases, *i.e.* the context cache hit ratio increases to over 90% in most cases with reasonable cache sizes.

2 Related Work

The related work is broken into two distinct categories: context transfers, and algorithms that dynamically generate the topology of wireless networks.

The previous work on context transfers has mostly focused on the IP layer using *reactive* transfer mechanisms [7], and general purpose transfer mechanisms without detailing transfer triggers [10]. The only previous work on link layer context caching was also originally *reactive* until neighbor graphs were recently added [9].

The IP layer context transfer mechanisms focus solely on the transfer of context from access router to access router, and while Koodli [7] mentions access points briefly– indicating that access routers and access points can be co-located. The context transfer mechanisms are designed solely for access routers and are reactive rather than pro-active as in neighbor graphs [7]. In the case of the SEAMOBY context transfer protocol, the protocol provides a generic framework for either reactive or pro-active context transfers [10]. The framework, however, does not define methods for implementing either reactive or pro-active context transfers. As a result, our approach can easily be integrated into the SEAMOBY protocol providing a pro-active context transfer mechanism as it was with IAPP [6].

IAPP was originally only reactive in nature– creating an additional delay in a handoff. As a result of our early results, the neighbor graph notion was included in the latest IAPP draft recommended practice [9].

The previous work on topology algorithms has focused on pre-authentication, automated bridge learning, and sharing of public key certificates [11–13].

Pack proposes pre-authentication be performed to the k most likely next access points. The k stations are selected using a weighted matrix representing the likelihood (based on the analysis of past network behavior) that a station, associated to AP_i, will move to AP_j. The mobile station may select only the most likely next access points to pre-authenticate, or it may select all of the potential next access points [11, 12]. Pack uses the notion of a frequent handoff region (FHR) to represent the adjacent access points, or neighbors, which is obtained by examining the weighted matrix. The weights within the matrix are based on an $O(n^2)$ analysis of RADIUS log information using the inverse of the ratio the number of handoffs from AP_i to AP_j to the time spent by the mobile station at AP_i prior to the handoff. While the FHR notion represents neighboring access points, it requires $O(n^2)$ computation and space, where n is the number of access points in the network, and must be created at the authentication server (AS). Furthermore, the FHR notion does not quickly adapt to changes in the network topology changes. This is in contrast to neighbor graphs which require $O(degree(ap))$ computation and storage space per AP [1] and which quickly adapt to changes in the network topology. Additionally, neighbor graphs can be utilized either in a distributed fashion at each access point, or client, and in a centralized fashion at the AS.

Capkun et. al. leverage station mobility to create an *ad-hoc* public key infrastructure by neighboring stations exchanging public key certificates to create a certificate graph [13]. The idea is that a neighboring station is most likely able to verify the identity of another station, and after successfully doing so add the certificate to their graph. The resultant graph represents the mobility pattern with respect to other stations. While this mobility graph has a different focus and use than neighbor graphs, mobile stations rather than access points or base stations, and caching certificates rather than context. It none-the-less uses the notion of neighbors, and we include a discussion of it for completeness.

In 1980's, to overcome the geographic limitation of a LAN, LANs began to be interconnected with other LANs using *bridges*. In this approach, a bridge, connected to two or more links, listens promiscuously to all packets and forwards them to a link on which the destination station is

---

[1] The cache consumes an $O(1)$ storage and computation.

known to reside. A bridge also learns of the locations of stations so that it forwards traffic to the correct link. In [14], Perlman proposed a *self-configuring* and *distributed* algorithm to allow bridges to learn the loop-free subset of the topology that connects all LANs, by communicating with other bridges. This subset is required to be loop-free (a spanning tree) to avoid unnecessary congestion caused by infinitely circulating packets. This *Spanning Tree Algorithm* or *Spanning Tree Protocol* in [15] is self-configuring because the only priori information necessary in a bridge is its own unique ID (MAC address). The algorithm requires very small bounded amount of memory per bridge, and bounded amount of communications bandwidth on each LAN, both are independent of the total number of LANs. Furthermore, there is no requirement for modifications to stations and the algorithm interoperates with simpler bridges, is another advantage of the algorithm. Neighbor graphs are also self-configuring operate in the same manner– examining network traffic, specifically layer 2 management frames, to create the wireless network topology dynamically. The two algorithms, and their purpose is different.

3 Background

3.1 IEEE 802.11 Handoffs

The IEEE 802.11 [16] MAC specification allows for two modes of operation: *ad hoc* and *infrastructure* mode. In *ad hoc* mode, two or more wireless stations (STAs) recognize each other (through beacons) and establish a peer-to-peer relationship. In infrastructure mode, an *access point* (AP) provides network connectivity to its *associated* STAs which forms a *Basic Service Set*(BSS). Multiple APs as a part of the same wireless network form an *Extended Service Set*(ESS). Because of mobility, load conditions, or degrading signal strength, a STA might *move* to another AP within the same wireless network. This process is referred to as a *handoff*. The mechanism or sequence of messages between a STA and the APs resulting in a transfer of physical layer connectivity and state information from one AP to another with respect to the STA is referred to as a handoff. While the process involves various MAC and network layer functions, we only focus on the layer 2 aspects in this paper.

We use the following terms in the paper: STA, station, client or user refers to a computing device capable of performing the role of an 802.11 mobile station. We use *old-AP* to refer to the AP to which a STA was associated prior to a handoff, and *new-AP* to refer to the AP to which the STA is associated after the handoff. The term *current-AP* refers to the AP to which a STA is currently associated to. The term *distribution system* (DS) refers to the interconnection

architecture for communication between the APs and other network devices (authentication server, routers, etc) which together form the ESS.

Figure 1 shows the sequence of steps that are designed to occur during a handoff. The first step (not indicated in the figure) is the termination of a STA's association to current AP. Either entities can initiate a *disassociation* for various reasons ([16], page 53). Due to mobility or degradation of physical connectivity (signal strength), it might not be possible for the STA or the AP to send an 802.11 disassociate message. In such cases, a timeout on inactivity or communication between APs or the receipt of an IAPP *Move-Notify* (discussed later) terminates the association.

During the second step, the STA *scans* for APs by either sending *probe request* messages (*active scan*) or by listening for beacon messages (*passive scan*) broadcast by APs on channels of interest. Messages A through D in the figure show the active scan. For each channel, a probe request (messages A and C) is sent by the STA and *probe responses* (messages B and D) are received from the APs in the vicinity of the STA. After scanning all intended channels, the STA selects the new-AP based on the data rates and signal strength. *Probe Delay* is the time spent by the STA in scanning and selecting the next AP. After the probe, the STA and new-AP exchange 802.11 *authentication* frames, and the latency incurred is the *authentication delay* (messages E and F). After authentication, the STA sends an 802.11 *reassociation request* to the AP (message G) and receives a *reassociation response* from the AP (message H) which completes the handoff process. The latency incurred during this exchange is the *reassociation delay* and this process is called the *reassociation process*. [2] During reassociation, the APs involved exchange station context information. This is achieved through the use of the *Inter Access Point Protocol* (IAPP). The next subsection discusses the broader role of IAPP in managing the distribution system (DS).

3.2 Inter Access Point Protocol

The IEEE 802.11f-recommended best practice specifies two types of interaction for completing context transfer [8]. The first form of interaction occurs between APs during a handoff and is achieved by the IAPP protocol, and the second form of interaction is between an AP and the RADIUS server.

IAPP plays a significant role during a handoff. The two main objectives achieved by inter-access point communication are : (a) *Single Association Invariant:* Maintaining a single association of a station with the wireless network,

---

[2] For a detailed analysis of the probe and authentication process, see [6].
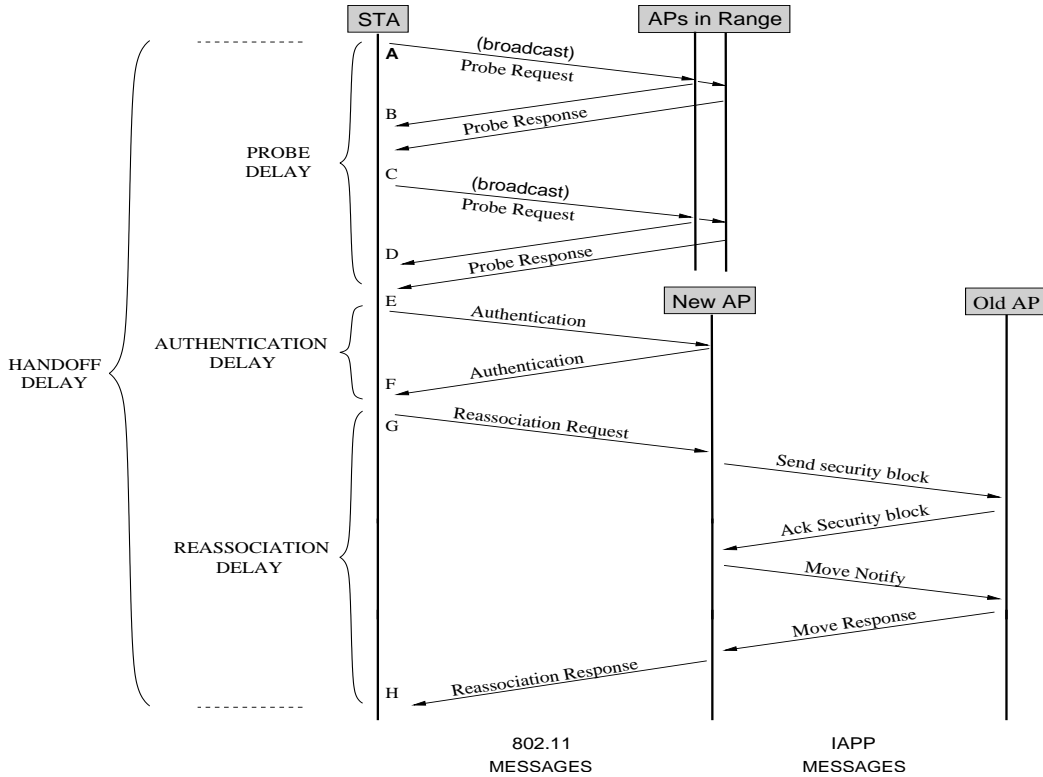
Fig. 1. *The handoff procedure by the IEEE 802.11 and IEEE 802.1f.*

and (b) the secure transfer of state and context information between APs involved in a reassociation reducing or eliminating the latency of due to context transfer.

Association and reassociation events change a station's point of access to the network. When a station first associates to an AP, the AP broadcasts an *Add-Notify* message notifying all APs of the station's association. Upon receiving an *Add-Notify*, the APs clear all stale associations and state for the station. This enforces a unique association for the station with respect to the network. When a station reassociates to a *new-AP*, it informs the *old-AP* of the reassociation using IAPP messages. Figure 1 shows the sequence of messages involved in the reassociation.

At the beginning of a reassociation, the new-AP can optionally send a *Security Block* message to the old-AP, each of which acknowledges with an *Ack-Security-Block* message. This message contains security information to establish a secure communication channel between the APs. The new-AP sends a *Move-Notify* message to the old-AP requesting station context information and notifying the old-AP of the reassociation. The old-AP responds by sending a *Move-Response* message.

For confidentiality of the context information, IAPP draft recommends the use of a RADIUS server (to obtain shared keys) to secure the communication between APs. The RADIUS server can also provide the address mapping between the MAC addresses and the IP addresses of the APs, which is necessary for IAPP communication at the network layer.

Although the IAPP communications serve to fulfil the mandatory DS functions, they invariably increase the overall handoff latency because of their reactive nature.

4 Neighbor Graphs

In this section, we describe the notion and motivation for neighbor graphs, and the abstractions they provide. As seen in figure 1, the reassociation phase primarily involves the transfer of station context from the old-AP to the new-AP. In order to improve the reassociation latency, the context transfer process (using IAPP) must be separated from the reassociation process. This can be achieved by providing the new-AP with the client-context prior to the handoff, or proactively. Since we are unable to predict the mobile station's movement, we need a method for determining the *candidate*

set of potential new-APs to perform the transfer prior to the handoff. The neighbor graph datastructure provides the basis for identifying this candidate set.

4.1 Definitions

Given a wireless network, we construct a datastructure called a *neighbor graph* which captures the *reassociation relationship* between access points.

Reassociation Relationship: Two APs, say, ap_i and ap_j are said to have a reassociation relationship if it is possible for an STA to perform an 802.11 reassociation through some path of motion between the physical locations of ap_i and ap_j, see the dotted lines in figure 2.

The reassociation relationship depends on the placement of APs, signal strength and other topological factors and in most cases corresponds to the physical distance (vicinity) between the APs.

AP Neighbor Graph: Define a undirected graph $G = (V, E)$ where $V = \{ap_1, ap_2, \ldots, ap_n\}$ is the set of all APs (constituting the wireless network under consideration), and there is an edge $e = (ap_i, ap_j)$ between ap_i and ap_j if they have a reassociation relationship. Define $Neighbor(ap_i) = \{ap_{i_k} : ap_{i_k} \in V, (ap_i, ap_{i_k}) \in E\}$, i.e. it is the set of all *neighbors* of ap_i in G.

While, the neighbor graph can be implemented either in a centralized or a distributed manner. In this work, we are implementing it in a distributed fashion, with each AP storing its set of neighbors. Each AP *learns* of a neighbor when it receives an 802.11 reassociation request, or an IAPP *Move-Notify* message. The construction and maintenance of this datastructure (in a distributed fashion) is discussed further in section 4.4.

Association Pattern: Define the *association pattern* $\Gamma(c)$ for client c as $\{(ap_1, t_1), (ap_2, t_2), \ldots, (ap_n, t_n)\}$, where ap_i is the AP to which the client reassociates (new-AP) at time t_i and $\{(ap_{i+1}, t_{i+1}) , (ap_i, t_i)\}$ is such that the handoff occurs from ap_i to ap_{i+1} at time t_{i+1}; the client maintains continuous logical network connectivity from time t_1 to t_n.

4.2 Proactive Caching and Locality of Mobility

Caching strategies are based on some *locality* principle, eg: locality of reference, execution etc. In this environment, we have locality in the client's *association pattern*. In this section we discuss the proactive caching strategy, based on *locality of mobility*.

We define the *Locality of Mobility* principle to state that for a client c, with association pattern $\Gamma(c)$ as defined above, for any two successive APs according to $\Gamma(c)$, say, ap_i and ap_{i+1} should satisfy the reassociation relationship.

This concept of locality is the abstraction captured by the neighbor graph as a datastructure.

The following functions/notations are used to describe the algorithm:

1. $Context(c)$: Denotes the context information related to client c.
2. $Cache(ap_k)$: Denotes the cache datastructure maintained at ap_k.
3. $Propagate\_Context(ap_i, c, ap_j)$: denotes the propagation of client c's context information from ap_i to ap_j. This can be achieved by sending a *Context-Notify* message from ap_i to ap_j (as discussed later in section 4.6).
4. $Obtain\_Context(ap_{from}, c, ap_{to})$: ap_{to} obtains Context(c) from ap_{from} using IAPP *Move-Notify* messsage as discussed in section 3.2.
5. $Insert\_Cache(ap_j, Context(c))$: Insert the context of client, c, in to the cache datastructure at ap_j. Perform an *LRU* replacement if necessary.

The Proactive Caching Algorithm: The access points use the following algorithm for proactive caching:

1. When a client c associates to ap_j:
   (i) For every $ap_i \in Neighbor(ap_j)$: $Propagate\_Context(ap_j, c, ap_i)$
2. When a client c reassociates to ap_j from ap_k :
   (i) If Context(c) not in Cache(ap_j): $Obtain\_Context(ap_k, c, ap_j)$
   (ii) For every $ap_i \in Neighbor(ap_j)$: $Propagate\_Context(ap_j, c, ap_i)$
3. On receipt of Context(c) at ap_j: $Insert\_Cache(ap_j, Context(c))$.

At each AP, the cache replacement algorithm we are using a *least recently used* approach.

The cache can be implemented as a hash table over a sorted linked list (according to the insertion time). This would give a cache lookup of $O(1)$ and a cache replacement of $O(1)$ as well. The method $Propagate\_Context$ requires sending the context to each neighbor and hence would incur an execution cost of $O(degree(ap_j)*propagation\_time)$, where *propagation_time* is the round-trip time for communication between the two APs under consideration.

4.3 An Example

Figure 2 shows the physical topology of a wireless network, and the corresponding neighbor graph. There are five APs $A \ldots E$, with their placement as shown. The dashed lines show the potential paths of motion for a client in the vicinity of A, and associated to it.

Fig. 2. *Figure shows an example placement of APs and the corresponding neighbor graph.*

When a client associates to A, its context information is propagated to neighbors B and E. Thus if the client is associated to AP A, regardless of the path of motion (as shown in the figure), the new-AP has to be either B or E when the client moves continously. In other words, there is no edge nor a reassociation relationship, for example, between A and D, because through all possible paths of motion allowed by the physical topology, a client cannot directly reassociate from A to D without (at least temporarily) going through B or E. By proactively sending the context to B and E, the client is guaranteed a fast handoff (assuming context remains in cache). Thus the neighbor graph (on the right in figure 2) captures this *locality* information in the form of a datastructure. Thus, the graph on the right of figure 2) is sufficient to abstract out this relationship from the given network topology.

4.4 Generation of the Neighbor Graph

The neighbor graph can be automatically generated (i.e. learned) by the individual access points over time. There are two ways that APs can learn the edges in the graph. Firstly, when an AP receives an 802.11 reassociation request frame from a STA, the message contains the MAC (BSSID) of the old-AP and hence establishes the reassociation relationship between the two APs. Secondly, receipt of a *Move-Notify* message from another AP via IAPP also establishes the relationship. These two methods of adding edges are complementary, and the graph will remain undirected.

Each AP maintains the edges locally in an LRU fashion. This is necessary in order to eliminate the outliers, i.e. incorrectly added edges. One situation where this would happen is a client that goes into the power save mode, and can

potentially wake up to reassociate to any other AP on the wireless network. Thus a timestamp based LRU approach would guarantee the freshness of the neighbor graph, and eliminate the outlier edges over time. The effect of the outliers on the performance of the algorithm would be nominal, as it would just result in an additional caching of a client's context for a short amount of time (LRU freshness).

The autonomous generation also eliminates the need for any survey or other manual based construction methods. As a result, this also makes the datastructure adaptive to dynamism in the reassociation relationship (i.e. changes in AP placements, physical topology changes, etc).

The graph is generated by executing the following pseudocode at each AP. ap_{host} is the AP on which the algorithm is assumed to be executing:

1. *Receipt of a reassociation request:* When a client *c* reassociates to ap_{host} from ap_i, add edge ap_i as a neighbor of ap_{host} (i.e. ap_{host} adds ap_i to its list of neighbors).
2. *Receipt of a Move-Notify:* When ap_{host} receives a *Move-Notify* from ap_i, add ap_i to the list of neighbors.

Thus the graph is constructed independently by each AP. The first client to traverse an edge incurs, a high handoff latency. But the edge is added to the graph, and the cost is amortized over subsequent handoffs. Thus after $O(|E|)$ *high* latency handoffs, the algorithm converges to its expected performance. The algorithm has a $O(1)$ running time per reassociation.

4.5 Expected Performance: Characterizing the Cache Misses

As discussed earlier, the caching algorithm is based on the locality of mobility principle. Since reassociation relationships are captured in the neighbor graphs and client-context

is forwarded to all neighbor APs, technically we would expect a 100% cache hit ratio for the reassociations. This assumes that the neighbor graph has been learned and the cache size is unlimited.

The above assumption takes us to the two kinds of cache misses possible during a reassociation:

1. *Reassociation between non-neighbor APs:* When a reassociation occurs between two APs that are not neighbors, the station-context does not get forwarded and results in a cache miss. The edge subsequently is added to the graph through the learning process. Thus when a wireless network is first brought up (or rebooted), the initial reassociations in the network would be cache misses.

2. *Context evicted by LRU replacement:* This happens when the client-context is evicted at the new-AP because of other clients reassociating to neighboring APs.

As discussed in the previous section, the first type of cache miss would occur only once per edge and has a nominal effect towards the performance in the long run. The second type of cache miss depends on mobility of other users, and hence dictates the performance of the algorithm. Presented below is a simple analysis of the expected performance perceived by a client (i.e. hit ratio observed by a client), with regard to its mobility.

Let $\Gamma(c) = \{(ap_1, t_1), (ap_2, t_2), \ldots, (ap_n, t_n)\}$, be the association pattern observed for a client c. Consider a reassociation $\{(ap_i, t_i), (ap_{i+1}, t_{i+1})\}$. Client c reassociated to ap_i at time t_i. Also at t_i, the client's context was inserted into the cache at ap_{i+1}. The time spent by c at ap_i, would be $T(c, ap_i) = t_{i+1} - t_i$. The longer the client stays at ap_i, the greater the chance of its context being removed by other clients reassociating to the neighbors of ap_{i+1}. Thus the probability $P(c, ap_{i+1})$ that the client's context is evicted from the cache at ap_{i+1} would be directly proportional to the time spent by the client at ap_i. Thus :

$$P(c, ap_{i+1}) \propto T(c, ap_i)$$

A faster client (i.e. higher mobility) would spend less time at each AP and hence would have a higher probability of a cache hit. Thus the performance of the algorithm perceived by a client would be expected to improve with its mobility.

4.6   IAPP and Proactive Caching

In this section, we discuss the modifications to IAPP to incorporate proactive caching using neighbor graphs. The modifications consist of two new messages; *Cache-Notify*, and *Cache-Response* for the purposes of implementing the *Propagate_Context()* method discussed in section 4.2. These

changes were included in the *IAPP draft recommended practice* version 5 [9].

Figure 3 shows the modified reassociation process (compared to figure 1). For the sake of clarity, the probe and authentication messages are not shown.

1. *Cache-Notify:* This message is sent from an AP to its neighbor and carries the context information pertaining to the client. It is sent following a reassociation or an association request.

2. *Cache-Response:* This is sent in order to acknowledge the receipt of *Cache-Notify*. A timeout on this message results in removal of the edge, as the neighbor AP might not be alive.

As can be seen from figure 3, a cache-hit avoids the *Move-Notify* and *Security-Block* communication latency during reassociation resulting in a faster handoff.

The knowledge of neighboring APs at each APs is essential for the effective operation of proactive caching. To avoid the management overhead of manually maintained neighbor graphs, the current draft of IAPP now includes the algorithms from section 4.2.

5   Experiments and Simulations

We present both simulation and implementation results to demonstrate the performance of proactive caching. Section 5.1 discusses the implementation results and simulation results are presented in section 5.2

5.1   Experiments

In this section, we discuss the implementation of the caching algorithm over a custom wireless testbed. We describe the testbed configuration, the process of the experiments, and the results. In brief, we measured 114 reassociations in the testbed resulting in an average reassociation latency of 15.37 ms for a cache-miss without an outlier and 23.58 ms with the outlier (which is the traditional IAPP communication latency) and 1.7 ms for a cache-hit– achieving an order of magnitude improvement in the reassociation latency.

The Wireless Testbed The wireless testbed spans a section of two floors (2nd and 3rd) of an office building. There were five APs on the third floor and four on the second. The geometry of the floors (L-shape and the dimensions) and topology of nine access points are shown in figure 4. The gray circles in the figure represent APs, labeled by an identifier. Three channels, namely 1, 6 and 11 were used by
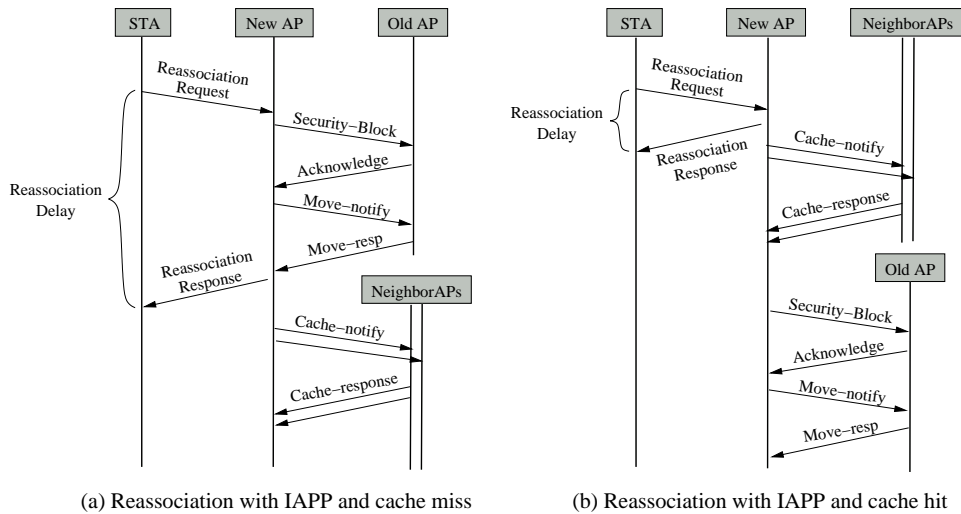
(a) Reassociation with IAPP and cache miss   (b) Reassociation with IAPP and cache hit

Fig. 3. *Message sequences during a handoff with context caching.*
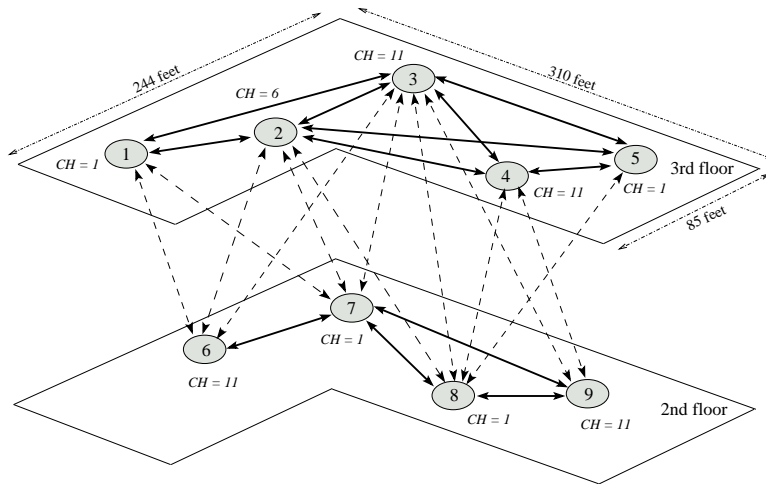

Fig. 4. *Experiment Environment and Neighbor Graph.*

the APs. There were 4 APs on channel 1 and 11 each and one AP on channel 6. These channels were assigned in a fashion to avoid interference with other wireless networks operating in the building resulting in less than optimal RF design.

The APs used for the experiments were based on a *Soekris* [17] board NET4521 which has a 133 MHz AMD processor, 64MB SDRAM, two PC-Card/Cardbus slots for wireless adapters and one *CompactFlash* socket. A 200mW Prism 2.5 based wireless card was used as the AP interface with a 1ft *yagi* antenna. OpenBSD 3.1 with access point functionality was used as the operating system.

The IAPP protocol, neighbor graphs, and the caching algorithm were implemented in the driver (for the wireless interface) along with the AP functionality.

Experiment Process To preclude possible interference, we shutdown the other wireless networks in the building during the experiment. Figure 5 shows the setup of the experiment. The *mobile unit* consisted of a client laptop, and a sniffer. A laptop with Pentium III 750 MHz CPU and 256 MB RAM and a Prism 2.5 based *ZoomAir* wireless card was used as the client. The reassociation latencies were
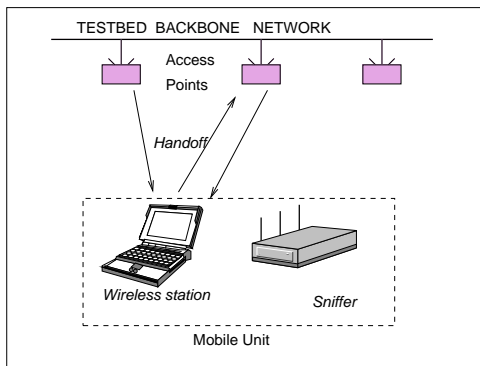
Fig. 5. *The experiment setup: the sniffer and the mobile client move together through the testbed in a random path of motion.*

measured by capturing management frames on channels 1, 6 and 11. This was done by the sniffer which had a wireless card dedicated to capturing traffic on each channel (1, 6, and 11). Since the APs were configured only on the above three channels, it was guaranteed that the sniffer would capture all management frames destined to or transmitted by an AP in the testbed (with respect to the STA) (primarily *reassociation request* and *response* frames). Three wireless interfaces in two laptops constituted the sniffer.

Two experiments were conducted. The first experiment was conducted with *fresh* APs, i.e. there were no neighbor relationships prior to the start of the experiment. The goal of this experiment was to study the effect of the learning process on the reassociation latencies with time. The second experiment (following the first) was to confirm guaranteed cache hits once the neighbor graph had been learnt by the APs. We discuss the detailed setup of each experiment below.

Experiment A: The first experiment consisted of a random walk with the mobile unit, through the physical span of the testbed. There were no neighbor relationships existing among APs prior to the start of the experiment. The experiment started with the client associating to AP-2 (refer figure 4), and a random path of motion covered all APs on third floor. The unit then moved to the second floor, covered all APs, and returned to the initial point of association (AP-2). This was one round of the experiment and nine rounds were conducted for statistical confidence in the measurements. This resulted in one association, and 114 reassociations during the entire experiment.

Experiment B: The second experiment, followed the first, consisted of two short rounds using a different client. The purpose of this experiment was to verify the existence of neighbor graphs (i.e learned from the first experiment) at each AP by observing a cache hit on all reassociations.

Experiment Results Figure 4 depicts the (3D) neighbor graph created during the experiment. The graph was constructed by observing the *reassociation request* frames captured by the sniffer. The directed edges indicate the direction of the reassociation (from the old-AP to the new-AP). The solid edges are intra-floor edges and the rest are inter-floor edges. The graph shows 23 distinct pairs of APs, between which the STA could reassociate.

Experiment A: Figure 6 shows the reassociation latencies at each AP [3]. The Y-axis is the latency in log base 2. The *circular* points represent reassociation with a cache-miss and *cross* points are the cache-hits. Most of cache-miss latencies reside around 16 *ms* except an outlier of 81 *ms* at AP-8. The cache-hit latencies are clustered around an average of 1.69 *ms*. There are a few cache-hits with latencies more than 4 *ms*. We reason that these outliers (involved with AP-4 and 5) are due to poor coverage design with respect to the building topology. AP-4 and AP-5 had relatively small transmission range when compared to other APs and they were physically close to each other (causing interference). There was another extreme outlier of 2.36 seconds latency with a cache-hit. We are convinced that this is caused by a sniffing error due to contention at the moment. This value is excluded in the analysis of the experiment.

Figure 7 shows the reassociation latencies observed with time. During the experiment, there was a cache-miss for the first reassociation to each AP (except AP-2). Since during the initial rounds of the experiment, the APs had not discovered their neighbors, the initial reassociations were cache misses. The graph visualizes how context caching decreases reassociation latencies as experiment time progresses. Except the very first reassociations and a few outliers, most reassociation latencies lie below 2 *ms*. In total, there were 8 cache-misses with average of 15.37 *ms* [4] and 105 cache-hits with average latency of 1.69 *ms*.

Experiment B: The second experiment, was done with a different client. The APs had learnt the neighbor graph, and hence during the experiment there were cache misses. Each association/reassociation forwarded the context to the neighbors, and hence the client's context was always found in cache during a reassociation [5]. This experiment had 18 reassociation, all cache hits, resulting in an average latency of 1.5 *ms*.

---

[3] The reassociation latency is attributed to the new-AP

[4] The outlier of 81 *ms* has been excluded from the average calculation. We eliminated it since it would unfairly distort our result by making it higher, *i.e.* better, than what is clearly the average of 16 *ms*.

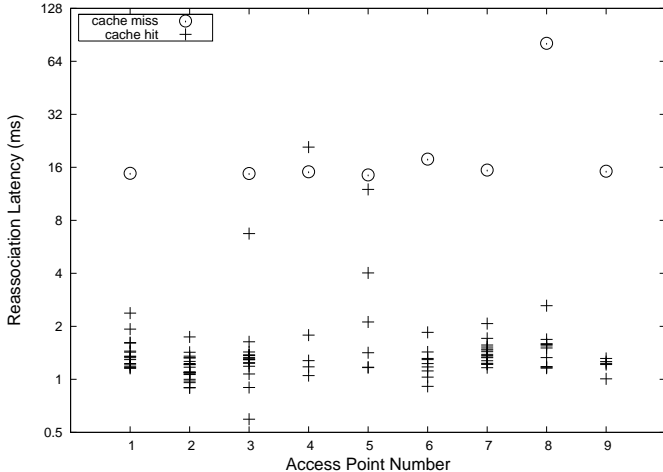[5] Since we had only one client in the experiment, there were no cache evictions.

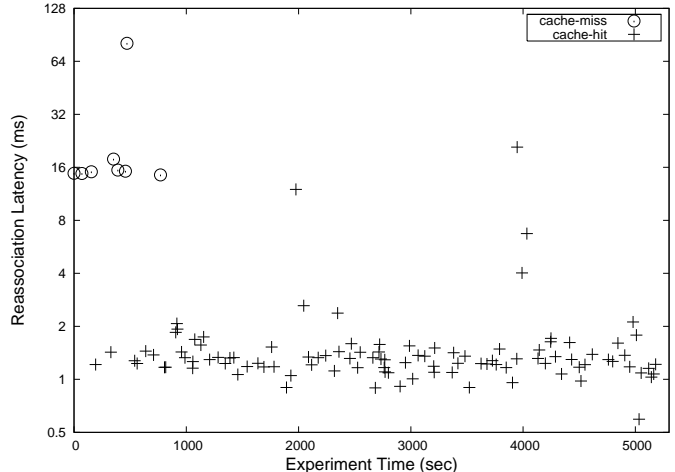**Fig. 6.** *Reassociation latencies at each access point.*

Fig. 7. *Reassociation Latencies with Time.*

Thus the experiment results show that proactive caching with neighbor graphs reduces the reassociation latency by an order of magnitude.

5.2 Simulations

Access points are embedded systems with limited resources (computing power and memory). A typical access point has around 4MB of RAM and 1 MB of flash. Client context information could potentially consist of security credentials, QoS information etc. Thus an AP can store only a limited number of contexts in its cache (LRU cache replacement). In this section we present results on how the algorithm performs under varying mobility and the number of clients and APs in the network.

Simulation Objectives:

1. To observe the effect of cache size, number of clients and mobility of clients on the cache hit ratio.
2. To observe the performance of caching with various neighbor graphs.

Each simulation starts with a set of APs, a neighbor graph structure connecting them, a set of clients and their initial distribution on the APs. Each client is assigned a *mobility index* (defined later), which dictates the mobility of the client throughout the simulation. We discuss below the assumptions and the model used.

Simulation Model and Assumptions

1. *AP Neighbor Graph does not change during the simulation:* As noted earlier, changes in the AP neighbor graph would cost (in the worst case), one high latency handoff per edge, and has a nominal effect on the overall cache performance.

2. *Correctness and completeness of the Neighbor Graphs:* We are assuming that the neighbor graphs are correct and complete, i.e. the simulations do not consider any reassociations which are not covered as edges in the graph [6]. This makes it sufficient to simulate reassociations according to the neighbor graph without maintaining any correspondence with the physical placement of APs (that would produce the neighbor graph).

3. *Initial User-AP distribution:* We have assumed a uniform distribution of clients across APs to at the start of the simulation. Figure 8 shows the distribution of the maximum number of users associated to each AP during a simulation with 100 APs, and 500 clients.

4. *Roaming Model:* The client roams according to the following model:

   (a) Let client c have an association pattern $\Gamma(c) = \{(ap_1, t_1), (ap_2, t_2), \ldots, (ap_n, t_n)\}$. The client c is said to roam from ap_1 to ap_n if (i) the time associated at each $ap_i, (1 < i < n) : t_{i+1} - ti$ is of the order of a typical reassociation latency (around 100 ms, [6]) and (ii) the time the client spends on ap_1 and ap_n is of the order of a typical client session [4]. Thus the client stays for a session-duration

---

[6] As discussed earlier, such reassociations would have resulted in the edge being added to the graph

with an AP, roams to another AP (according to an association pattern), and stays for another session.

(b) At any given point of time during the simulation, the client is either *roaming* (according to definition above) or staying associated to its current AP.

(c) The association pattern of a roam is decided randomly: If the client c is associated to ap_i, it can move to any one of its neighbors $(ap_{i_1}, ap_{i_2}, \ldots ap_{i_k})$ with equal probability.

5. *User Mobility:* Define *mobility index* of a client as the probability that the client is *roaming* at any given point of time during the simulation. At the end of the simulation it converges to the (Total time spent in roaming/Total simulation time). Mobility indices are assigned to clients on a scale of $1 \ldots 100$. The distribution of mobility indices on clients is uniform.

Fig. 9. *Plot of clients mobility and the cache hit ratio achieved.*

Fig. 8. *Distribution of Maximum number of clients associated to an AP during a simulation with 100 APs and 500 users.*

Simulation Environment

1. The simulation uses random and connected neighbor graphs with 10, 20, 50 and 100 vertices.

2. *Duration of the Simulation:* The simulation runs for one million reassociation events uniformly distributed over the users according to their mobility indices. This makes the duration of the simulation large enough for statistical confidence in the results.

Simulation Results

1. *Mobility Improves Proactive Caching Performance:* Figure 9 shows the cache hit ratio achieved by clients according to their mobility index. The figure compares the hit ratio performance over neighbor graphs of size 10, 20, 50 vertices keeping the cache size constant. In all
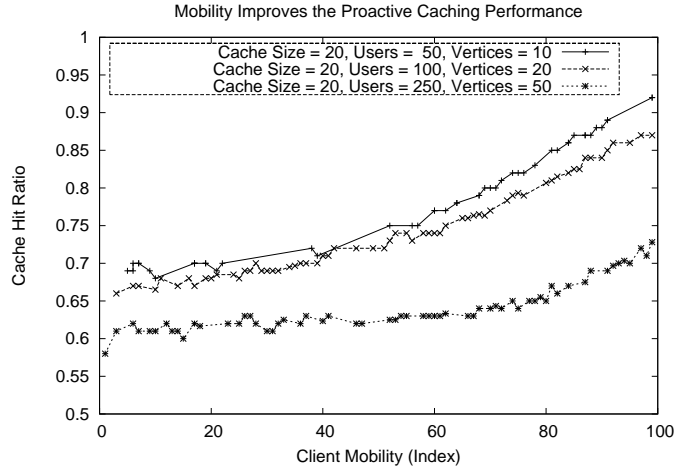
three curves, the hit ratio increases with client mobility. The relative improvement diminishes with increasing number of vertices in the NG graph, and the prime reason for this being the constant cache size. Later plots elucidate this observation.

2. *Effect of Cache Size and Client Mobility on Hit Ratio:* Figure 10 shows the effect of cache size on the hit ratio keeping the number of clients, and the NG graph the same. The graph had 100 vertices, and 200 users. Clearly an increase in the cache size has a direct impact on the cache hit ratio, to the extent that for a cache size of 50, all client with mobility indices 70 or higher did not have any cache misses (slow handoffs) at all.

3. *Effect of Cache Size and Number of Users on Hit Ratio:* Number of clients in the network has a direct impact on the performance. Figure 11 shows the effect of the two parameters on hit ratio. Figure 12 shows the effect of the cache size as a percentage of the number of users on the hit ratio. The data points were taken for cache sizes varying from 20 to 50 and the number of users varying from 200 to 500 in increments of 100. Thus a 15 percent cache size is sufficient for a hit ratio of 88 percent while a cache size of 25 percent gives a hit ratio of around 93 percent.

6 Performance Improvements

In this section, we discuss a modification to the caching algorithm, and the initial results that we obtained that show an improvement in the average hit ratio.

The improvement is the following. When an STA moves from an old-AP to a new-AP, in addition to the new-AP
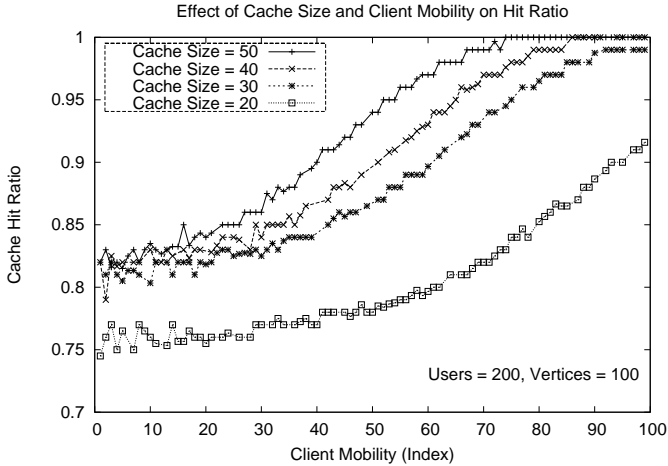
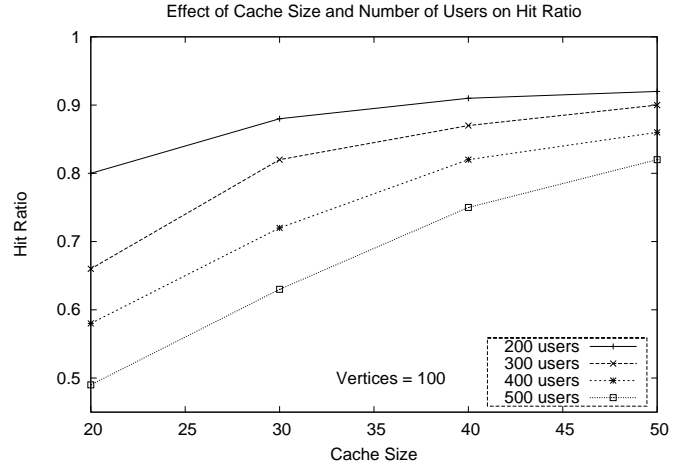**Fig. 10.** *Effect of Cache Size and Client Mobility on Hit Ratio.*

Fig. 11. *Effect of Cache Size and Number of Users on Hit Ratio.*

forwarding context to its neighbors, the old-AP *removes* context from *its* neighbors (other than the new-AP). As an example, consider the NG graph shown in figure 15. Let the STA roam from D to E. According to the algorithm detailed in section 4.2, after the STA moves to E, it propagates the context to its neighbors F,G and H. With the modification, D sends a *remove-context* message to its neighbors A,B and C (other than E), which remove the cache entry for the STA if present. This brings us to the following *invariant* as being satisfied (we state without proof):

Neighborhood-users Invariant: If Context(c) is in Cache(ap_i), for a client c and an AP ap_i, this implies c is associated to an AP ap_k such that $ap_k \in Neighbor(ap_i)$.

In other words, if an entry (client-context) is present in the cache of particular AP, this would imply that the corresponding client is associated to one of the neighbors (of this AP). Figure 14 shows the improvement in the hit ratio with the cache invalidation. Shown is the hit ratio observed by clients according to their mobility index with and without cache invalidation. The results show an average improvement of 17.4% for 500 clients on a 100 vertex graph and a cache size of 50. Figure 13 shows the effect of cache size (as a percentage of the number of users) on the average hit ratio (over all clients) with and without cache invalidation. As can be seen from the above plots, cache invalidation always has an effect of improving the cache hit ratio. The logical reason behind this observation is that fact that the cache-invalidation method removes the *unnecessary* entries in the cache of an AP.

7 Conclusions and Future Work

In this paper, we have introduced a novel, efficient and a dynamic data structure, *neighbor graphs*, which captures the topology of a wireless network by autonomously monitoring the handoffs. This datastructure abstracts the physical topology of the network into a neighbor relationship which can be used as a vehicle for numerous applications. Neighbor graphs provide more *structure* to the distribution system (DS) interconnecting the APs forming the wireless network. This structure, which provides the DS information about the physical topology of the APs, can be leveraged for optimizations on existing algorithms (load balancing, network management, and key pre-distribution) or could lay the foundation for interesting and novel applications.

As an application of the neighbor graphs, we implemented and studied the performance of the *proactive caching algorithm* for faster wireless handoffs. The caching algorithm uses neighbor graphs to send station-context to its neighbors prior to the handoff and hence separates the context transfer process from reassociation. We have implemented the approach over IAPP [9] running on a dedicated wireless testbed and presented results from experiements conducted on the testbed.

In our experiments, 114 reassociations occurred with an average reassociation latency of 23.58 ms (including the one outlier) and 15.37 ms (without the outlier) for a cache-miss (traditional handoff), and 1.69 ms for a cache-hit. The results show an order of magnitude improvement due to proactive caching. In our simulations, we study the performance of the algorithm under varying network characteristics : user mobility, number of users associated to the network, and number of APs forming the network. We conlude
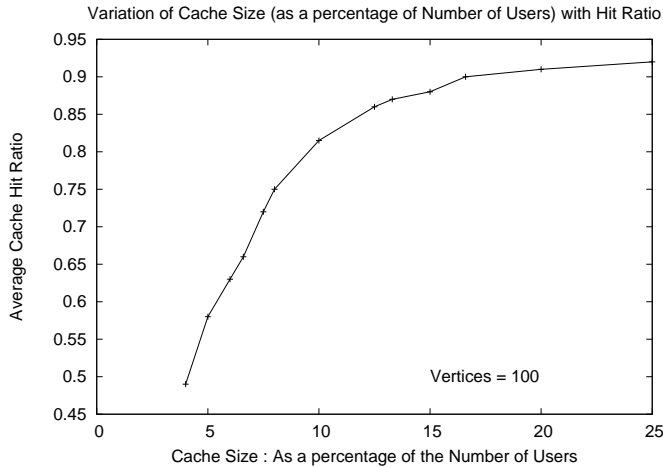
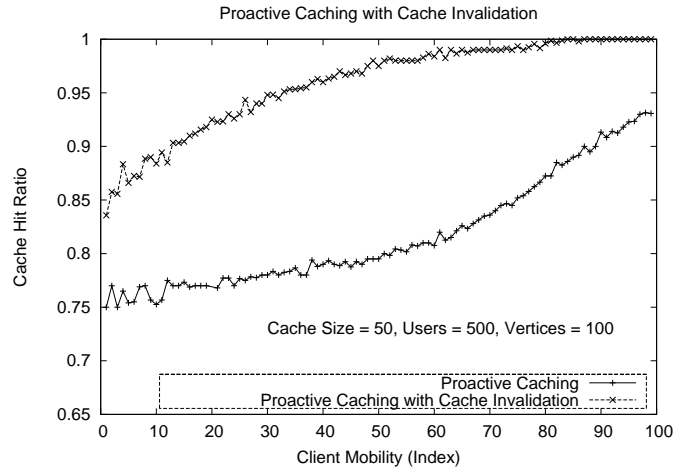**Fig. 12.** *Variation of Cache Size (as a percentage of Number of Users) with Hit Ratio.*

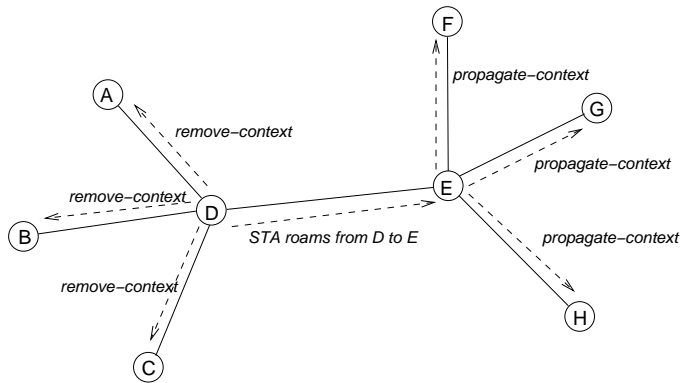**Fig. 14.** *Effect of cache invalidation on the performance of the proactive caching method.*

Fig. 13. *An example NG graph to show effect of the modification.*

network management of APs. As a special appliation, neighbor graphs could potentially lead to a scalable method of organizing and managing a large scale cooperative wireless network which interconnects APs from different network domains and with different characteristics (network bandwidth, cost etc). Neighbor graphs can also be used to eliminate the expensive scanning operation for faster MAC layer handoffs by making an intelligent guess about the list of APs on a particular channel.

References

1. D. Tang and M. Baker, "Analysis of a metropolitan-area wireless network," in *Mobile Computing and Networking*, pp. 13–23, 1999.
2. K. Lai, M. Roussopoulos, D. Tang, X. Zhao, and M. Baker, "Experiences with a mobile testbed," in *Proceedings of The Second International Conference on Worldwide Computing and its Applications (WWCA '98)*, Mar 1998.
3. A. Balachandran, G. Voelker, P. Bahl, and P. Rangan, "Characterizing user behavior and network performance in a public wireless lan," 2002.
4. M. Balazinska and P. Castro, "Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network," in *International Conference on Mobile Systems, Applications, and Services(To Appear)*, May 2003.
5. International Telecommunication Union, "General Characteristics of International Telephone Connections and International Telephone Circuits." ITU-TG.114, 1988.
6. Anonymous for Reviewing Purposes, "No title given,"
7. R. Koodli and C. Perkins, "Fast Handover and Context Relocation in Mobile Networks," *ACM SIGCOMM Computer Communication Review*, vol. 31, October 2001.
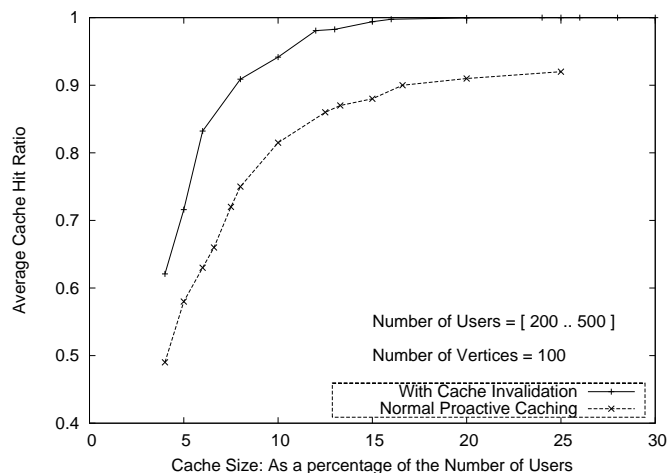
that the performance of the algorithm (hit-ratio) improves as the user mobility increases eventually reaching a 100% hit-ratio under certain network configurations. We find that the cache size plays an important role in the performance of the algorithm. A cache size of 15% (of the number of users associated to the network) gives a minimum cache hit-ratio of 88%.

In the future, we plan to pursue a formalization of the proactive caching algorithm with cache invalidation. As observed from the initial simulation results, (figure 14) there is an average improvement of 17.4% for 500 clients on a 100 vertex graph and a cache size of 50.

Extending the neighbor graphs, we are working on a comprehensive key distribution scheme for secure inter-network and intra-network roaming. We plan to investigate application of neighbor graphs to perform load balancing, and

Fig. 15. *Effect of cache invalidation on the hit ratio as a function of the cache size (percentage of number of users).*

8. IEEE, "Draft 4 Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation," *IEEE Draft 802.1f/D4*, July 2002.

9. IEEE, "Draft 5 Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11 Operation," *IEEE Draft 802.1f/D5*, January 2003.

10. M. Nakhjiri, C. Perkins, and R. Koodli, "Context Transfer Protocol," *Internet Draft : draft-ietf-seamoby-ctp-01.txt*, March 2003.

11. S. Pack and Y. Choi, "Fast Inter-AP Handoff using Predictive-Authentication Scheme in a Public Wireless LAN," *IEEE Networks 2002 (To Appear)*, August 2002.

12. S. Pack and Y. Choi, "Pre-Authenticated Fast Handoff in a Public Wireless LAN based on IEEE 802.1x Model," *IFIP TC6 Personal Wireless Communications 2002 (To Appear)*, October 2002.

13. S. Capkun, L. Buttyan, and J.-P. Hubaux, "Self-Organized Public-Key Management for Mobile Ad Hoc Networks," *To appear n IEEE Transactions on Mobile Computings 2003*.

14. R. Perlman, "An algorithm for distributed computation of a spanningtree in an extended lan," pp. 44–53, 1985.

15. R. Perlman, *Interconnections, Second Edition: Bridges, Routers, Switches and Internetworking Protocols.* Pearson Education, September 1999.

16. IEEE, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," *IEEE Standard 802.11*, 1999.

17. "Soekris Engineering." URL: http://www.soekris.com.