Hierarchical Temporal Mobility Analysis with Semantic Labeling

Adele Hedrick
Faculty of Science
UOIT
Oshawa, ON, Canada
Email: adele.hedrick@uoit.net

Ken Q. Pu
Faculty of Science
UOIT
Oshawa, ON, Canada
Email: ken.pu@uoit.ca

Ying Zhu
Faculty of Business and IT
UOIT
Oshawa, ON, Canada
Email: ying.zhu@uoit.ca

***Abstract*—Abstract—User mobile devices are nowadays constantly in a sea of WiFi hotspots. By recording the observed WiFi signals over time, it is possible for the mobile phone to deduce the salient locations in its environment, and the mobility patterns of the user. In real life, fluctuations of WiFi hotspots and unreliability of mobile phone antenna necessarily create false or missed readings, making problems related to location identification particularly challenging.**

We propose a family of algorithms to perform the tasks of location identification, mobility inference, and localization. Our algorithms are able to handle real-life noisy readings, and construct a hierarchy of salient locations providing a multiresolution model of the environment.

1. Introduction

With the omnipresence of WiFi hotspots throughout the urban and suburban areas, it has become possible to perform location identification and mobility analysis on any mobile device. Given the nature of WiFi networks, sensing the surrounding WiFi basic service set identifiers (BSSID) can be done accurately with insignificant power requirement and latency, making it an ideal way generate spatio-temporal mobility data streams. The *raw* mobility data stream consists of a time series of *readings*. Each reading has a timestamp, and a collection of triples $\langle \mathrm{bssid}, \mathrm{ssid}, \mathrm{strength} \rangle$, where bssid is a universally unique identifier of a WiFi hotspot, ssid is a user-defined name, and finally $\mathrm{strength}$ is a numerical measure of the signal intensity of that particular WiFi hotspot.

The problem of location inference based on WiFi scans has received a great deal of attention (e.g., [2], [4], [6], [7], [9], [12]). Our work distinguishes from the existing literature in several important ways. We do not make any assumptions on the signal propagation model, nor any prior knowledge of the ambient environment. In the *location identification phase*, our algorithm continuously constructs a set of physical locations based on the observation of the BSSID's and their signal strength. Our emphasis is that location identification is performed in an unsupervised, ad-hoc and incremental fashion. To make it suitable for the mobile runtime environment, the *online* algorithm runs with sublinear time and space complexity.

Our algorithm consists of a series of pipelined stages of WiFi reading processing. Each stage is a non-blocking online algorithm whose output is consumed by the next stage.

Movements

In the first stage, we detect *movements* of the mobile device by maintaining a multiresolution timeline of the readings. We hierarchically partition the timeline to detect mobility at various time scales. The multiresolution is used to reject false positives and cope with false negatives in the WiFi readings.

Physical Locations

The segments generated by the movement detection phase are further processed to build a database of *physical locations* based on the BSSID signatures of the segments. This allows us to map out the ambient environment over time. We note that movement detection and physical location identification are both non-blocking, so that physical locations are identified instantly, and incrementally refined.

Semantic Clustering

While the BSSID information provides accurate physical location identification, they do not carry sufficient *semantic* value to the end user. We do not use GPS for the sake of minimizing power consumption. It turns out that user-defined SSID names are a useful source of semantic knowledge of the physical locations. Our algorithm analyzes the SSID names to organize the physical locations into a semantically meaningful hierarchy.

Location Inference

Finally, the algorithm uses maximal likelihood estimation to infer locations with semantic labels to the raw WiFi readings.

2. Related Work

Radio frequency and ultrasonic signals are used in conjunction [13] to estimate the location of mobile devices, with the overhead of installing beacon devices throughout the building and listener devices to user mobiles. Received signal strength indication (RSSI) measurements between the cellular base stations and the user device S [10] are processed by algorithms combining an extended Kalman filter, approximate pattern matching and velocity vectors to

predict future movements of users across cell boundaries. It is suitable for only coarse-scale outdoor environments.

Indoor localization algorithms that use RSSI have been extensively investigated [2], [4], [6], [9], [12] because there is no overhead cost of additional hardware required before deployment. In [12], positioning and tracking of users depend on the locations of known WiFi access points which may not be realistic in the general case. Our work does not require such prior knowledge, we use readings from any available access points that are detected. Some work [1], [3] focus on the problems arising from the unpredictable effect of physical factors on the received signals and use a propagation model for the relationship between signal and location. An alternative is the fingerprinting model [5], [8], [11] where offline readings are recorded first and the online readings are then compared to them. The k nearest neighbor algorithm is used in [8], [11] to estimate the user location. It is an efficient algorithm but suffers in estimation accuracy. The localization system in [2] has two phases: an offline phase in which a radio map is built by collecting readings on a grid of reference points and decomposed into clusters; an online phase in which signal readings are first identified to belong to certain clusters and then more finely localized using compression sensing theory. This method has high accuracy but still requires an initial offline phase which is not realistic in general, especially in outdoor environments.

In works such as [14], [16], the indoor area is divided into a grid and signal data is recorded from a set of fixed known points which is used as training data, generating a probability distribution of signal strengths given location values. With new signal observations, the posterior distribution is computed and the location with the highest probability is chosen.

The works of [15] and [7] both model user locations as states of a dynamic system with the noisy observations of RSSI data, and apply an implementation of Bayesian filters called particle filters to probabilistically estimate the system state (i.e., user location). An initial training phase builds a wireless sensor map of the environment divided into cells by sampling at predefined points; this step is dependent on the assumed WiFi sensor model, either signal propagation or fingerprinting models. Then location is estimated on a spatial connectivity graph; each motion update step of the Bayesian filter moves the user along an edge of this graph. These probabilistic methods are powerful but suffers the drawback of potential high computational complexity inherent in the particle filters method whose worst-case complexity grows exponentially in dimensions of the state space. Moreover, all these above-mentioned approaches all require an initial training phase, and therefore are not online and ad-hoc.

3. Movement Segmentation

We study the problem of online processing of WiFi scans collected by a mobile device. We refer to each scan as a *reading*. A reading is defined as $\langle t(r), \mathbf{B}(r), \mathrm{SSID}, s(\cdot|r)\rangle$ where $t(r)$ is the timestamp of the reading, and $\mathbf{B}(r) \subseteq$ BSSID is a set of BSSID of the wifi hotspots that the scan detected. The $\mathrm{SSID} : \mathbf{B}(r) \to \mathrm{Names}$ is a mapping of BSSID names to user-defined name of the WiFi hotspot. $s(\cdot|r) : \mathbf{B}(r) \to R^+$ is a mapping of BSSID of the reading r to a signal strength; $s(b|r)$ is the intensity of b in the reading r.

func **TimelineClustering**(T)
$X = T$
while $\lvert X\rvert > 1$ {
$\quad i^* = \mathrm{argmax}\{\mathrm{sim}(T_i, T_{i+1}) : i \in [1, \lvert X\rvert]\}$
$\quad r = \mathbf{merge}(T_i, T_j)$
$\quad X = \mathbf{replace}\ [T_i, T_{i+1}]\ \mathbf{with}\ r$
}
return Tree with root X

Algorithm 1: Bottom-up timeline clustering

3.1. Movement detection

In this section we describe an online algorithm to organize the timeline of readings into multiresolution segments. The timeline is partitioned such that within each segment,

Definition 1. A *timeline* T is a sequence of readings. We denote T_i as the i-th reading of the timeline T.
A *segment* of the timeline S is a contiguous subsequence of T.

The $\mathbf{B}(T_i)$ induces a similarity measure among the readings.

Definition 2 (Reading similarity).

$$\mathrm{sim}(T_i, T_j) = \mathrm{Jaccard}(\mathbf{B}(T_i), \mathbf{B}(T_j)) = \frac{|\mathbf{B}(T_i) \cap \mathbf{B}(T_j)|}{|\mathbf{B}(T_i) \cup \mathbf{B}(T_j)|}$$

This allows us to organize T using hierarchical clustering. Unlike the traditional hierarchical clustering of sets of items with a similarity measure, clustering a timeline has the added constraint that each cluster must only contain adjacent readings. Algorithm 1 is an adaptation of the bottom-up agglomerative to compute hierarchical timeline clustering. In $\mathbf{TimelineClustering}(T)$, $\mathbf{merge}(T_i, T_{i+1})$ creates a new node in the hierarchical timeline, and its BSSID set is simply the union of the BSSID sets of its children:

$$\mathbf{B}(\mathbf{merge}(x, y)) = \mathbf{B}(x) \cup \mathbf{B}(y)$$

We remark that **TimelineClustering** is certainly *not* an online algorithm, as it requires $\mathcal{O}(|T|^2)$ complexity to build the hierarchy. This will be remedied by the online version to be presented later.

Let $H = \mathbf{TimelineClustering}(T)$, where H is a binary tree with the leaf nodes as the readings in T. H presents a multiresolution segmentation.

We use the following notations: The leaf-nodes of H, written $\mathrm{leaf}(H)$, is just the readings T. The nodes of H

```
func Segments(v, ε)
if minsim(v) > ε {
    return [v]
} else {
    return Segments(left(v), ε) ∪ Segments(left(v), ε)
}
```

Algorithm 2: Identifying homogeneous nodes

is writtn $\mathrm{Nodes}(H)$. The descendants of $v \in \mathrm{Nodes}(H)$ is written $\mathrm{descendants}(v)$. The readings of v is defined as:

$$\mathbf{R}(v) = \mathrm{descendants}(v) \cap \mathrm{leaf}(H)$$

Since H is a binary tree, each interior node v has two children, written $\mathrm{left}(v)$ and $\mathrm{right}(v)$.

The challenge is the determine the nodes in H whose readings are all at the *same* physical location. This is determined by the *minimum similarity*.

Definition 3 (Minimum Similarity). Given a set, R, of readings, the minimum similarity of R, written $\mathrm{minsim}(R)$ is defined as:

$$\mathrm{minsim}(R) = \min\{\mathrm{sim}(r, r') : (r, r') \in R \times R\}$$

The minsim provides a measure if a set of readings R span over more than one physical location. We observe that if R contains readings taken from two distinct physical locations, then, due to the spatial distance there will be at least one pair of readings, r and r', that are quite dissimilar: $\mathrm{sim}(r, r') \leq \epsilon$ for some small constant $\epsilon \ll 1$. This means that $\mathrm{minsim}(R) \leq \epsilon$.

For nodes in the hierarchy, $v \in \mathrm{Nodes}(H)$, its minsim is simply the minsim of its readings.

Definition 4 (Minsim for nodes).

$$\mathrm{minsim}(v) = \mathrm{minsim}(\mathbf{R}(v))$$

We will use minsim and ϵ to perform the segmentation of T using the hierarchy H. Algorithm 2 performs a top-down traversal of H to identify the top-most nodes in H with $\mathrm{minsim}(R) > \epsilon$, which we call *homogeneous nodes*.

Each homogeneous node in corresponds to a segment of the timeline in which all the readings are at the same physical location according to the BSSIDs.

There are two shortcomings with the approach: (1) The timeline clustering is not an online algorithm. (2) The segmentation requires the threshold measure ϵ. We will address these issues in the coming sections.

3.2. Online hierarchical timeline clustering

Our objective is to update a hierarchical timeline cluster H incrementally when more readings are appended to the timeline T.

```
func OnlineTimelineCluster(T)
H = Tree(first(T))
S = Segments(root(H), ε)
for r ∈ T
    T' = update(T, r)
    for each v ∈ Nodes(T') − Nodes(T) {
        if v is a top-level homogeneous node in T' {
            S = S ∪ v
    }
    for each v ∈ Nodes(T) − Nodes(T') {
        S = S − {v}
    }
```

Algorithm 3: Online timeline clustering

We define a procedure $\mathrm{update}(H, v_{\mathrm{insert}}, v_{\mathrm{new}})$ where H is the timeline cluster to be updated, and we wish to add a new node v_{new} *after* the v_{insert}.

We analyze the behaviour of $\mathrm{update}(\dots)$ by different cases. Let $H' = \mathrm{update}(H, v_{\mathrm{insert}}, v_{\mathrm{new}})$

- If
$$v_{\mathrm{insert}} = \mathrm{root}(H)$$
Then
$$H' = \mathrm{newnode}(v_{\mathrm{insert}}, v_{\mathrm{new}})$$
- Otherwise, define
$$v_{\mathrm{parent}} = \mathrm{parent}(v),\ v_{\mathrm{prev}} = \mathrm{left}(v_{\mathrm{parent}})$$

Next week check if v_{new} is closer to v_{insert} or not, compared to v_{prev}.

- If $\mathrm{sim}(v_{\mathrm{new}}, v_{\mathrm{insert}}) \geq \mathrm{sim}(v_{\mathrm{insert}}, v_{\mathrm{prev}})$, then we create a new node $v' = \mathrm{newnode}(v_{\mathrm{insert}}, v_{\mathrm{new}})$, and promote v_{prev} to replace v_{parent}. Then, we recursively invoke:
$$H' = \mathrm{update}(H, v_{\mathrm{prev}}, v')$$
- If $\mathrm{sim}(v_{\mathrm{new}}, v_{\mathrm{insert}}) < \mathrm{sim}(v_{\mathrm{insert}}, v_{\mathrm{prev}})$, then we try to insert v_{new} after v_{parent}.
$$H' = \mathrm{update}(H, v_{\mathrm{parent}}, v_{\mathrm{new}})$$

The online version of the hierarchical timeline clustering is shown in Algorithm 3. For each new reading, it updates the hierarchical timeline H and the homogeneous nodes based on the differences between the new H and the old H.

3.3. Unsupervised segmentation

In practice, the WiFi readings will encounter several differen types of *noises*.

- False positives - it is possible that a reading will detect one or more BSSIDs which are not local to the physical location. False positives may be the result

of faulty WiFi scan, or by moving WiFi hotspots created by other mobile devices.
- False negatives - a reading my miss certain BSSIDs which are local to the physical location. This may be the result of signal blockage or momentary interference.
- Moving BSSIDs - moving mobile devices often create transient WiFi hotspots, making it possible for a common WiFi BSSID appear at multiple physical locations.

By choosing the right threshold during the segmentation ϵ, we can eject errors due to noise. With the right choice of ϵ, we classify readings with sufficient difference (as low similarity between their BSSID sets) as distinct locations even if some readings contain errorenous BSSIDs.

We make the assumption that all the segments in H are either homogeneous or not. At the leaf level, we have $\mathrm{minsim}(v) = 1$, and as we traverse upwards toward the root, $\mathrm{minsim}(v) \to 0$. A sudden drop in minsim from level i to $i+1$ suggests that at the higher level (and coarser timeline partition), nodes are covering more than one physical location.

The selection of ϵ is minsim corresponding to the level with the largest drop in average $\mathrm{minsim}(v)$ for v in the level.

3.4. Approximate segmentation using sampling

The definition of minsim (Definition 3) can be costly as it requires computation in $\mathcal{O}(|R|^2)$. As we compute the root node in the hierarchical timeline, $|R|$ grows rapidly.

Given a node in the hierarchy H, We can accurately estimate $\mathrm{minsim}(v)$ using sampling as follows.

$$
\begin{aligned}
v_L &= \mathrm{left}(v), v_R = \mathrm{right}(v) \\
R_L &= \mathrm{sample}(\mathbf{R}(v_L), N) \\
R_R &= \mathrm{sample}(\mathbf{R}(v_R), N) \\
\mu &= \min\{\mathrm{sim}(r, r') : r \in R_L, r' \in R_R
\end{aligned}
$$

Finally the estimation of $\mathrm{minsim}(v)$ is given by:

$$\mathrm{minsim} \simeq \min\{\mathrm{minsim}(v_L), \mathrm{minsim}(v_R), \mu\}$$

4. Location Identification

In Section 3, we described a unsupervised online algorithm to identify a sequence of movements, $\mathbf{S}$, based on the BSSID readings. The result of the algorithm is a sequence of segments, each of which is guaranteed to be present a single physical location.

4.1. Constructing location index

In this section, we describe an efficient algorithm to identify the set of *unique* locations based on BSSID signatures of each segment. Given a segment (a sequence of consecutive readings in the timeline) S, $\mathbf{R}(S)$ is the readings belonging to S, and $\mathbf{B}(S) = \bigcup\{\mathbf{B}(r) : r \in \mathbf{R}(S)\}$.

The unique locations are characterized by repeated occurrences of segments with similar BSSID signatures.

While we can use k-mean clustering to cluster the segments based on the BSSID signatures, our application requies an efficient online method that does not require a prescribed value of k.

Recall that the segments $\mathbf{S}$ is being produced in a streaming fashion. will propose an algorithm to incrementally build a database of physical locations $\mathbf{L}$. Each physical location $L \in \mathbf{L}$ is characterized by a set of *weighted* BSSIDs.

The algorithm begins with an empty set. As a segment S is generated in $\mathbf{S}$, we look for a location $L \in \mathbf{L}$ that is sufficiently similar S by the measure of $\mathrm{sim}(\mathbf{B}(L), \mathbf{B}(S))$. If no such L exists, then we have discovered a new location, and we create a new location in $\mathbf{L}$ based on $\mathbf{B}(S)$.

The algorithm of location identification is given in Algorithm 4. It has a parameter c which is the minimal similarity for a segment to belong to a physical location.

```
func IdentifyLocations(S)
L = new index of locations
foreach S ∈ S {
   find L ∈ L such that sim(B(L), B(S)) > c
   if not found {
      L = L ∪ newloc(S)
   } else {
      Update readings of L with readings of S
   }
}
```

Algorithm 4: Algorithm for identifying distinct locations from a stream of movements.

4.2. Transient versus persistent locations

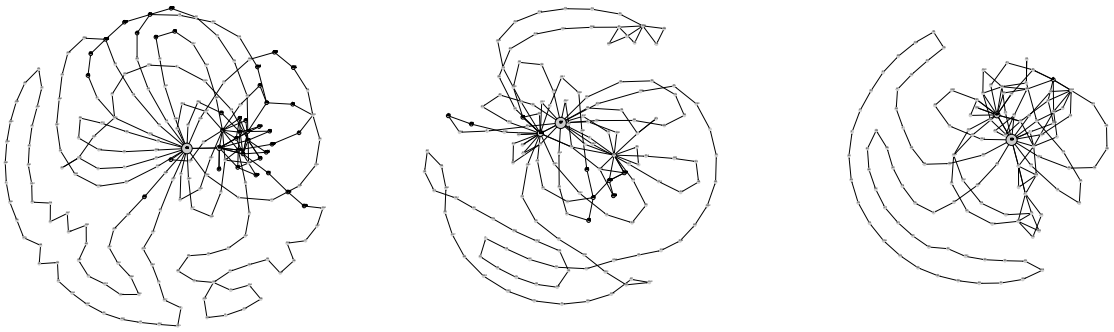
Algorithm 4 produces an index of *all* locations visited by the mobile device. $\mathbf{L}$ also maintains the association between readings/segments and the identified physical locations.

Each location $L \in \mathbf{L}$ has several attributes: (1) $\mathrm{freq}(L)$ is the number of times that L appeared in the timeline T. This is the number of segments associated with L. (2) $\mathrm{duration}(L)$ is the total duration that the mobile device was present at location L in the timeline T. This is the sum of all the durations of segments associated to L. The two metrics $\mathrm{freq}(L)$ and $\mathrm{duration}(L)$ allow us to classify the physical locations into two distinct classes: *transcient* and *persistent* locations.

For some minimal duration τ, we can define the transient locations as:

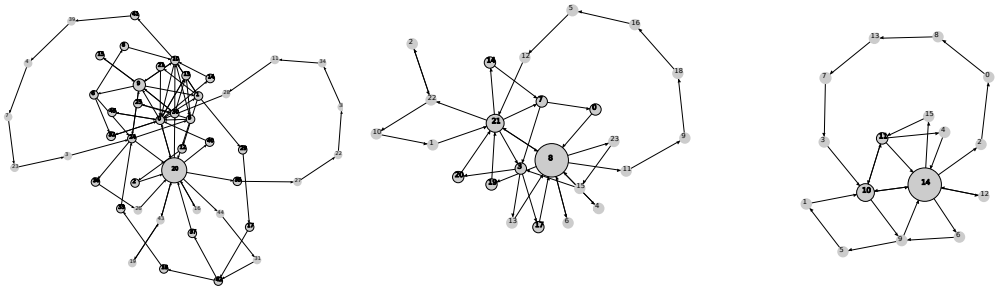
$$\mathbf{L}_{\mathrm{Transient}} = \{L \in \mathbf{L} : \mathrm{freq}(L) = 1 \text{ and } \mathrm{duration}(L) < \tau\}$$

The persistent locations are $\mathbf{L}_{\mathrm{Persistent}} = \mathbf{L} - \mathbf{L}_{\mathrm{Transient}}$.



(a) (b) (c)

Figure 1. Transition diagrams of timelines of all locations labeled by: (a) physical locations, semantic labels with $i = 2$, semantic labels with $i = 1$.



(a) (b) (c)

Figure 2. Transition diagrams of timelines of *persistent* locations labeled by: (a) physical locations, semantic labels with $i = 2$, semantic labels with $i = 1$.

func merge($\mathbf{L}$, K)
  $\mathbf{L}$: the locations
  K: a set of keywords used to merge locations
  returns: the set of semantic locations after merging

L_{new} = newlocation($\emptyset$)
foreach $L \in \mathbf{L}$ {
  if $\forall k \in K\ \exists k' \in \text{SSID}(L)$ such that $k \preceq k'$ {
    $L_{\text{new}} = L_{\text{new}} \cup L$
    delete L from $\mathbf{L}$
  }
}
if L_{new} is not empty {
  $\mathbf{L} = \mathbf{L} \cup \{L_{\text{new}}\}$
}
return $\mathbf{L}$

Algorithm 5: Merging locations based on a key SSID set.

5. Semantic Grouping of Physical Locations

In Section 3 and Section 4, we presented two online algorithms **OnlineTimelineCluster** and **IdentifyLocations**. We compose the two algorithms forming a pipeline that automatically identifies movements based on incoming WiFi hotspot readings.

The limitation so far is that the algorithms have only been working with BSSID signatures. An example of a BSSID (for a Starbucks outlet) looks like `00:24:6c:46:c5:f0`. Its use is limited to a universal identifier of the WiFi hotspot, not as a humanly readable label. For this reason, users can assign semantically meaningful strings to BSSIDs. For instance, `00:24:6c:46:c5:f0` is better known as "`Starbucks WiFi`" at this particular location.

For the application of mobility analysis, SSID names are a valuable source of semantic information about the physical location. If two distinct physical locations both contain WiFi hotspots labeled by "`Startbucks Wifi`" and "`Starbucks Wifi 4893`", then it is reasonable to infer that the mobile device has visited two different coffee venues: both being `Starbucks`, but at two different physical locations. We would like to create succinct and informative summarization of mobility in terms of semantically meaningful location groups to the user by means of *semantic grouping* of locations.

By semantic grouping, we are to build a semantic location tree whose leaf nodes are the physical locations, and the intermediate nodes are labeled. An intermediate x in the location tree signify that the leaf nodes (physical locations)

under x are semantically related.

Our assumption of semantic relevance is that two WiFi hotpots are considered semantically related if they share some common features in their user assigned SSID names. For example, the `Starbucks4230` and `Starbucks WiFi2039` share the common prefix of `Starbucks`. This can be formalized by *semantic labels*.

Definition 5 (Semantic labels of locations). Let $K = \{k_1, k_2, \ldots, k_m\}$ be a set of keywords. We say that K is can serve as a semantic label of a location L if every keyword in K is a prefix of a SSID label of WiFi hotspots of L.
Namely,

$$\forall k \in K, \exists x \in \mathrm{SSID}(L), k \preceq x$$

Two locations L_1 and L_2 are *mergable* by K if K is a semantic label for both L_1 and L_2.

func SemanticMerge($\mathbf{L}$, n) $\mathbf{L}$: the list of locations i: the maximal of keywords to use as key *returns*: set of semantic locations

$K^* = \mathrm{argmax}\{\lvert\mathbf{L}\rvert - \lvert\mathrm{merge}(\mathbf{L}, K)\rvert : K \in \mathrm{SSID}(\mathbf{L})^i\}$
$\delta = \max\{\lvert\mathbf{L}\rvert - \lvert\mathrm{merge}(\mathbf{L}, K)\rvert : K \in \mathrm{SSID}(\mathbf{L})^i\}$
while $\delta > 0$ {
$\mathbf{L} = \mathrm{merge}(\mathbf{L}, K)$
$K^* = \mathrm{argmax}\{\lvert\mathbf{L}\rvert - \lvert\mathrm{merge}(\mathbf{L}, K)\rvert : K \in \mathrm{SSID}(\mathbf{L})^i\}$
$\delta = \max\{\lvert\mathbf{L}\rvert - \lvert\mathrm{merge}(\mathbf{L}, K)\rvert : K \in \mathrm{SSID}(\mathbf{L})^i\}$
}
return $\mathbf{L}$

Algorithm 6: An algorithm to aggregate physical locations to groups based on their semantic information.

In the example of two locations containing SSID names of `Starbucks4230` and `Starbucks WiFi2039` in their respective SSID sets, the keyword `Starbucks` can be used as part of a semantic label to represent both locations.

Given a semantic label, K, and a set of locations $\mathbf{L}$, we can merge all the locations $L \in \mathbf{L}$ of which K is a semantic label. The merge function is shown in Algorithm 5. The function `merge(`$\mathbf{L}$`,` K`)` creates a new semantic location L_{new} that contains all the mergable locations.

By building semantic locations using successively coarser keyword sets as semantic labels, we can build a tree of semantic locations.

Definition 6 (Semantic Location Tree). A *semantic location tree* is a tree with levels $0, 1, \ldots n+1$, with leaf nodes at level 0 as the physical locations $\mathbf{L}$. At level $i > 0$, the intermediate nodes are labeled by $n - i + 1$ distinct names.

For $n = 3$, nodes at level 1 are labeled by $n-1+1 = 3$ distinct names. At level 2, nodes are labeled by $n-2+1 = 2$ two distinct names. The top-level is $n + 1 = 4$, which is labeled by $n-(n+1)+1 = 0$ names, so it has no labels. The labels are used to reflect the commonality of the physical locations. The root encompasses all the physical locations, and thus carries no semantic labels.

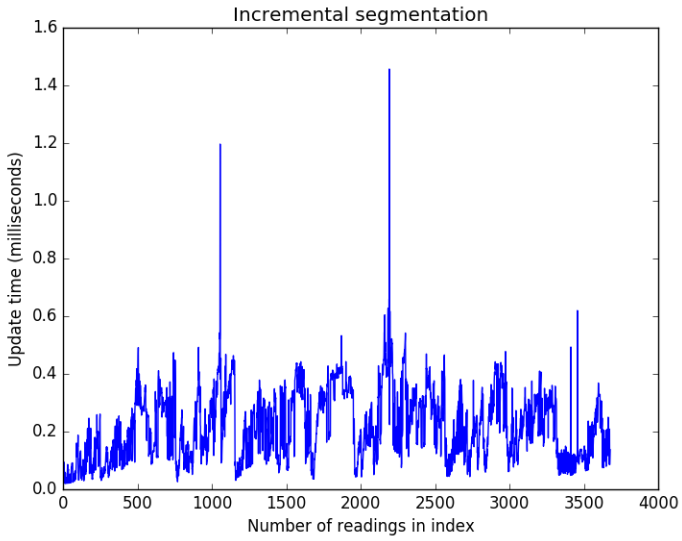


Figure 3. Performance of the online location identification

At each level, we iteratively merge locations using keyword sets of size i. We choose to select the keyword sets to maximize the reduction in the number of semantic locations after merge.

Given a set of location $\mathbf{L}$ with each location L with a SSID set $\mathrm{SSID}(L)$, the optimal semantic label $K*$ of length i is defined as:

$$K^* = \mathrm{argmax}\{\lvert\mathbf{L}\rvert - \lvert\mathrm{merge}(\mathbf{L}, K)\rvert : K \in \mathrm{SSID}(\mathbf{L})^i\}$$

The semantic label K^* is then used to merge the locations $\mathbf{L}$ to $\mathbf{L}'$. We then iteratively pick the optimal semantic label of $\mathbf{L}'$, and repeat the merge step. The iteration terminates if K^* can no longer reduce the number of locations. This is formally described in Algorithm 6.

6. Implementation and Experimental Evaluation

References

[1] P. Bahl and V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 775–784. Ieee, 2000.

[2] C. Feng, W. S. A. Au, S. Valaee, and Z. Tan. Received-signal-strength-based indoor positioning using compressive sensing. *Mobile Computing, IEEE Transactions on*, 11(12):1983–1993, 2012.

[3] B. Ferris, D. Haehnel, and D. Fox. Gaussian processes for signal strength-based location estimation. In *In proc. of robotics science and systems*. Citeseer, 2006.

[4] A. Hatami and K. Pahlavan. A comparative performance evaluation of rss-based positioning algorithms used in wlan networks. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, pages 2331–2337. IEEE, 2005.

[5] K. Kaemarungsi and P. Krishnamurthy. Modeling of indoor positioning systems based on location fingerprinting. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1012–1022. IEEE, 2004.

[6] A. Kushki, K. N. Plataniotis, and A. N. Venetsanopoulos. Kernel-based positioning in wireless local area networks. *Mobile Computing, IEEE Transactions on*, 6(6):689–705, 2007.

[7] J. Letchner, D. Fox, and A. LaMarca. Large-scale localization from wireless signal strength. In *Proceedings of the national conference on artificial intelligence*, volume 20, page 15. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[8] B. Li, J. Salter, A. G. Dempster, and C. Rizos. Indoor positioning techniques based on wireless lan. In *LAN, First IEEE International Conference on Wireless Broadband and Ultra Wideband Communications*. Citeseer, 2006.

[9] K. R. Liu. Signal processing techniques in network-aided positioning. *IEEE Signal Processing Magazine*, 1053(5888/05), 2005.

[10] T. Liu, P. Bahl, and I. Chlamtac. A hierarchical position-prediction algorithm for efficient management of resources in cellular networks. In *Global Telecommunications Conference, 1997. GLOBECOM'97., IEEE*, volume 2, pages 982–986. IEEE, 1997.

[11] J. Ma, X. Li, X. Tao, and J. Lu. Cluster filtered knn: A wlan-based indoor positioning scheme. In *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, pages 1–8. IEEE, 2008.

[12] A. S. Paul and E. A. Wan. Wi-fi based indoor localization and tracking using sigma-point kalman filtering methods. In *Position, Location and Navigation Symposium, 2008 IEEE/ION*, pages 646–659. IEEE, 2008.

[13] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan. The cricket location-support system. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43. ACM, 2000.

[14] T. Roos, P. Myllymäki, H. Tirri, P. Misikangas, and J. Sievänen. A probabilistic approach to wlan user location estimation. *International Journal of Wireless Information Networks*, 9(3):155–164, 2002.

[15] V. Seshadri, G. V. Zaruba, and M. Huber. A bayesian sampling approach to in-door localization of wireless devices using received signal strength indication. In *Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on*, pages 75–84. IEEE, 2005.

[16] A. Smailagic and D. Kogan. Location sensing and privacy in a context-aware computing environment. *Wireless Communications, IEEE*, 9(5):10–17, 2002.