

# Predicting mobility events on personal devices

Arjan Peddemors,<sup>ab\*</sup> Henk Eertink,<sup>a</sup> Ignas Niemegeers<sup>b</sup>

<sup>a</sup>Novay, INCA Group, Enschede, The Netherlands

<sup>b</sup>TU Delft, WMC Group, Delft, The Netherlands

---

## Abstract

High-end mobile phones are quickly becoming versatile sensing platforms, capable of continuously capturing the dynamic context of their owners through various sensors. A change in this context is often caused by the fact that owners – and therefore the devices they carry – are moving from one place to another. In this paper, we model the sensed environment as a *stream of events*, and assume, given that people are creatures of habit, that time correlations exist between successive events. We propose a method for the *prediction in time* of the next occurrence of an event of interest, such as ‘arriving at a certain location’ or ‘meeting with another person’, with a focus on the prediction of *network visibility events* as observed through the wireless network interfaces of the device. Our approach is based on using other events in the stream as predictors for the event we are interested in, and, in case of multiple predictors, applying *different strategies* for the *selection of the best predictor*. Using two real world data sets, we found that including predictors of infrequently occurring events results in better predictions using the best selection strategy. Also, we found that cross-sensor (cross-interface) information in most cases improves the prediction performance.

*Keywords:* event prediction; personalization; kernel density estimation; time series; mobility modeling; context-awareness

---

## 1. Introduction

This paper focuses on a method to predict the visibility in time of wireless network entities as they are observed from the perspective of a personal mobile device. Knowing when a network will get in range, or knowing when a network will be out of range, is important information for various data communication applications. A data synchronization job, for instance, may require uninterrupted connectivity to sync local data with data at a remote location, in which case it benefits from an accurate estimate on how long a candidate network remains visible – and therefore staying available to carry the synchronization data traffic.

The approach taken here relies on the continuously updated information describing the visibility over time of network entities as seen on the wireless interfaces of a personal mobile device. When examining this stream of information, the patterns in the visibility of a network entity correspond with the day-to-day patterns in mobility of the owner of the device, for those network entities that are stationary. At any time, multiple network entities may be visible, and often entities are visible in a certain order, reflecting, for instance, the travelling from home to work or vice versa. We can therefore expect that a time correlation exists between the visibility of network entities seen in-order, and that the visibility of one such entity may be used to predict the visibility of another. Although we use examples, a synthetic data set, and real user traces in this paper that stay close to the problem of predicting network visibility, we describe our method in generic terms so that it may be applied to events through other sensors as well.

---

\* Corresponding author. Tel. +31 53 4850421

*Email addresses:* Arjan.Peddemors@novay.nl, Henk.Eertink@novay.nl, I.Niemegeers@ewi.tudelft.nl

We want to predict *in time* the next occurrence of an event of interest in a stream consisting of multiple reoccurring events. This stream can be thought of as comprised of a number of sequences equal to the number of different events, with every sequence marking the times at which the particular event occurs. To predict an event taking place in the future, we may use different models. A prediction may be based on estimating the probability of the occurrence of an event at a point in time relative to the beginning of the current period in the stream, assuming that the period duration is known. Or, it may be based on estimating the probability of the occurrence of the event relative in time to the previous occurrence of this event. The *main focus* of this paper, however, is on estimating the probability of the time of the occurrence of a *future event of interest* given the *previous time of occurrence of any of the defined events* in the stream (including the event of interest itself).

In general terms, our approach can now be described as follows. We estimate the probability of an event occurrence based on another event, using a non-parametric density estimation method called *kernel density estimation*. We use both the straightforward form and the conditional form of kernel density estimation. As there may be more than one previous (preceding) event, each predicting the event of interest, a part of the method consists of selecting the best predictor. We define a number of different strategies to select the best preceding event to be used as predictor. The moment of obtaining visibility and the moment of losing visibility of a particular network entity each corresponds with a separate event. Therefore, the number of different events in the stream is twice the number of different network entities. As we will see, the proposed method requires the stream to be periodic and the events in the stream to be relatively sparse to assure tractability.

A common approach to predicting events in a system is to associate events with system states and then model the state transitions with a Markov model. This approach has a number of drawbacks when predicting mobile device sensor data. It may be hard to define states for a stream with different events, especially when the number of events is large and keeps growing, and when they may occur at the same time. Various data clustering techniques may help to extract stream states, although these techniques must be applied with care (see [8]). Furthermore, we are interested in the future time that another state becomes active for the first time after the current state, which is a classical Markovian problem, known as the *first passage time*. It is in general difficult to solve analytically, even for Markov models with time-homogeneous state transitions, and requires exponential time to compute recursively (see [13], pp. 25) for discrete time models. The approach presented here does not require the definition of system states and needs polynomial computing time, but lacks the appealing simplicity of Markov models.

The contributions of this paper are the following. It provides a novel method to predict the time of the next occurrence of an event of interest given a stream consisting of multiple reoccurring events, including the event of interest itself. It formulates a number of different strategies to select amongst available predictors in the method, which are evaluated on real data. It studies the event prediction performance using the best strategy from different angles, using real user traces.

This paper is organized as follows. Section 2 introduces the definitions and the outline of our approach. Section 3 provides an overview of kernel density estimation and kernel conditional density estimation and its applicability to our problem. Section 4 introduces strategies for best predictor selection, followed by section 5 discussing the prediction results. Section 6 describes computational aspects and section 7 provides a discussion on various aspects of our approach. Section 8 discusses related work. The final section 9 gives a summary of the conclusions.

## 2. Events and predictors

In this paper we consider a situation where the data from mobile device sensors as observed over time is represented as a *stream* of different *events*, with every event occurring once or multiple times. An example event is the ‘arrival within range of the home 802.11 network’, which, under normal circumstances, takes places multiple times when observing the 802.11 interface on a personal device longer than a few days. We denote an event as  $z_i$  and the time of occurrence  $x$  of this event as  $t_{i,x}$ . The stream can be regarded as a multidimensional or multivariate time series with one dimension for every event, as depicted in Figure 1, although it does not have the uniformly spaced time intervals typically used for time series analysis. Our objective is to predict the first occurrence of an event after the current time  $t_p$ , so we are interested in the probabilities associated with the future time that the next occurrence of event  $z_i$  takes place, given the past data on all other events.

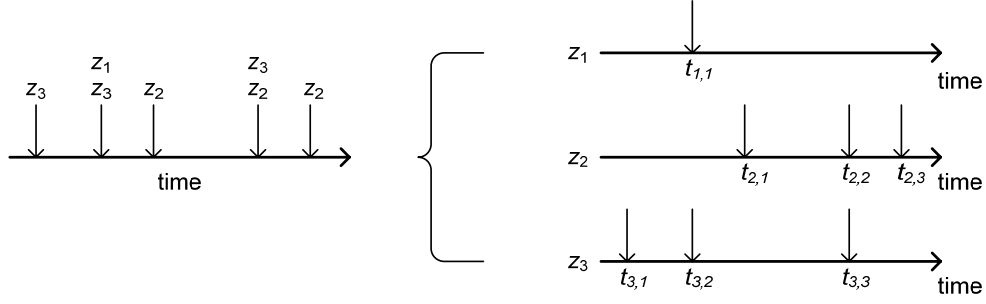


Figure 1: decomposition of a stream of three different events, occurring at different moments in time

The outline of our approach is as follows. We assume no prior knowledge on the relation in time between occurrences of different events, but use only the historical information observed in the stream so far. Let  $z_i$  be the event we want to predict and let  $z_j$  be any other event in the stream, including  $z_i$  itself. We are interested in the *waiting time*  $T_{w,i}$  of event  $z_i$  at the moment of the occurrence of an event  $z_j$ , which means that the most recent occurrence of  $z_j$  (at time  $t_{j,c}$ ) coincides with the present time, so that  $t_{j,c} = t_p$ . Let  $b_{p,i,j}$  be all pairs  $\{t_{j,y}, t_{i,x}\}$  in the stream with  $t_{j,y} < t_{i,x}$  that mark the first occurrence of  $z_i$  after an occurrence of  $z_j$ . In other words,  $b_{p,i,j}$  essentially holds the time values for all previous cases with the first occurrence of event  $z_j$  after  $z_j$ . Now, extract  $b_{p,i,j}$  from the stream observed so far and use the pairs to estimate the probability density function of the waiting time. Note that the waiting time is *relative* to the time of the occurrence of  $z_j$ , whereas the  $\{t_{j,y}, t_{i,x}\}$  pairs are relative to the time origin of the stream, as depicted in Figure 2. The *waiting time estimate* is denoted as  $\hat{T}_{w,i}$  and the associated estimated probability density function is  $\hat{f}_{w,i}$  so that the probability of having a waiting time between the future times  $t_a$  and  $t_b$  is

$$P(t_a < \hat{T}_{w,i} < t_b \mid b_{p,i,j}, t_{j,c} = t_p) = \int_{t_a}^{t_b} \hat{f}_{w,i}(t \mid b_{p,i,j}, t_{j,c} = t_p) dt, \quad (1)$$

for all  $t_p < t_a < t_b$ .

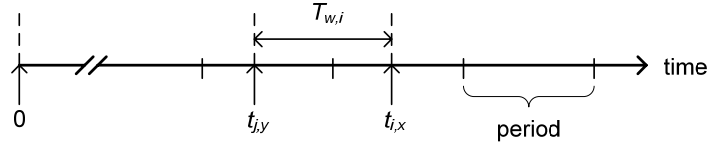


Figure 2: Event times  $t_{j,y}$  and  $t_{i,x}$  are relative to the time origin of the stream, while the waiting time  $T_{w,i}$  is relative to the time of event  $z_j$  (denoted as  $t_{j,y}$ )

A straightforward way to calculate  $\hat{f}_{w,i}$  is by only looking at the previous waiting times (the time delta between each  $\{t_{j,y}, t_{i,x}\}$  pair). Alternatively, when assuming that the stream is periodic, i.e., is showing dominant patterns in a cyclic manner, we can also calculate  $\hat{f}_{w,i}$  depending on the value of  $t_{j,y}$  relative to the current period. In that case,  $\hat{f}_{w,i}$  is a conditional probability density function and we need to translate the  $t_{j,y}$  values in  $b_{p,i,j}$  relative to their periods.

For instance, network entities are often visible in daily reoccurring patterns – the period is 24 hours – and it may matter highly when during this period event  $z_j$  occurs. When  $z_j$  is the event of getting in range of a GSM cell on the road in between home and work, the waiting time for getting in range of the home 802.11 network ( $z_i$ ) is considerably different when this cell is seen in the morning than when it is seen in the late afternoon. A substantial part of our method deals with estimating conditional probability densities. For that part to apply, the stream must be periodic.

**Definition 1:** Waiting time probability distribution *The waiting time probability distribution of the occurrence of event  $z_i$ , given the occurrence of event  $z_j$ , describes the probabilities of observing a waiting time for event  $z_i$  at the moment that  $z_j$  occurs. It is only valid at that moment.*

When observing the stream, with every occurrence of event  $z_j$  we have a predictor for the next occurrence of  $z_i$ .

**Definition 2:** Predictor *A predictor assigns probabilities to the future time of the next occurrence of an event  $z_i$  given the occurrence of an event  $z_j$ .*

A predictor is equivalent to the waiting time probability density  $\hat{f}_{w,i}(t \mid b_{p,i,j}, t_{j,c} = t_p)$ . To simplify notation, we will also use  $f_{w,i}(t \mid z_j)$  for  $f_{w,i}(t \mid b_{p,i,j}, t_{j,c} = t_p)$ . When repeating the density estimation

for multiple conditional events  $z_j$ , we have a set of predictors for  $z_i$  so that at a given time multiple predictors may apply because their associated events have occurred in the stream. In that case, we intend to select the best predictor for event  $z_j$ .

Figure 3 illustrates the dynamics in predictor availability in an example stream. We keep track of all applicable predictors using a list. At the occurrence of event  $z_i$ , this list of predictors is cleared, because previous predictors do not apply anymore. Note that by choosing  $z_j = z_i$ , there is always at least one predictor available; predictor  $f_{w,i}(t | z_i)$  is then immediately inserted in the cleared list. We will address density estimation and conditional density estimation in Section 3 and best predictor selection in Section 4.

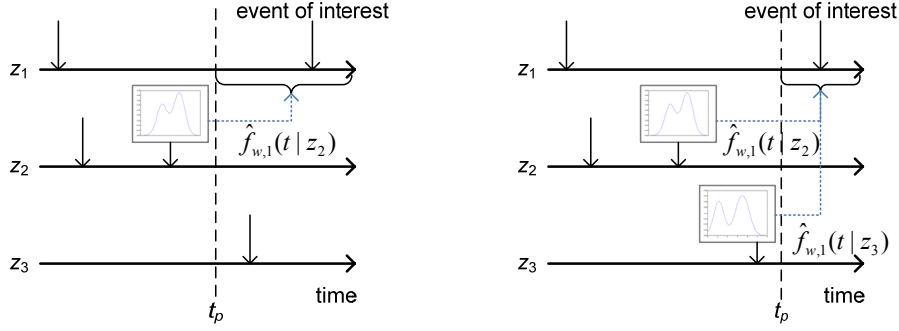


Figure 3: Increasing predictor availability with time for an example stream

With a predictor  $\hat{f}_{w,i}(t | z_j)$ , we can determine the  $z_i$  waiting time probabilities at the time of the occurrence of event  $z_j$ . Now, as long as event  $z_j$  has not occurred yet, we want to be able to predict its next occurrence not only at the times that events such as  $z_j$  occur, but rather at arbitrary times when we require such a prediction – we can not assume that these times coincide. Therefore, we are interested in the *remaining waiting time* or the *residual waiting time* ( $T_{r,i}$ ). This is also illustrated in Figure 3. In this figure we show the occurrence of three events  $z_1$ ,  $z_2$ , and  $z_3$ . At time  $t_p$ , we want to predict when  $z_1$  will occur the next time. The current time  $t_p$  is in between the event  $z_2$  and the event of interest  $z_1$  (left) and similarly, in between the events  $z_2$  and  $z_3$  and the event of interest  $z_1$  (right).

**Definition 3:** Remaining waiting time probability distribution *The remaining waiting time probability distribution of the occurrence of event  $z_i$ , given a previous occurrence of event  $z_j$ , describes the probabilities of observing a waiting time for event  $z_i$  at a moment after the occurrence of  $z_j$ , provided that  $z_i$  has not occurred since the occurrence of  $z_j$ .*

The relation between the waiting time density  $f_{w,i}$  and the remaining waiting time density  $f_{r,i}$ , with  $t_p$  the present time, is given by

$$f_{r,i}(t | b_{p,i,j}, t_{j,c} < t_p) = \frac{f_{w,i}(t | b_{p,i,j}, t_{j,c} = t_p)}{\int_{t_p}^{\infty} f_{w,i}(u | b_{p,i,j}, t_{j,c} = t_p) du} \quad (2)$$

It shows that the remaining waiting time can be derived from the waiting time. This is a result known from, amongst others, reliability theory, where the remaining waiting time is called the residual lifetime. Note that the remaining waiting time can be the same as the waiting time, when the density function has a ‘memoryless’ exponential shape. We will also use  $f_{r,i}(t | z_j)$  to denote  $f_{r,i}(t | b_{p,i,j}, t_{j,c} < t_p)$ .

The density estimation for a predictor may involve a considerable computational cost when the stream history is large. This may force us to use only a part of the history to reduce this cost. Moreover, exhaustive computation of all predictors in a stream may be intractable. In that case, only the predictors that apply most frequently may be used, or the predictors that are within a time window together with the event of interest. We discuss computational issues in section 6.

We want to apply this approach in on-line scenarios, i.e., in situations where the density estimation and best predictor selection take place while the stream data continues to be collected. We do assume that the process generating the data is not completely stationary, so we are interested in the convergence properties of the density estimates and the dynamics in the order of the best predictor list. In some situations, the number of dimensions in the stream grows with time. The prediction evolution is studied in section 5.

### 3. Density estimation

Estimating the probability density function of a random variable from a set of observed values is important for a wide range of tasks. Two main approaches exist for density estimation: *parametric* and *nonparametric*. The parametric approach assumes that the probability density function has a known shape, and selects the parameters for this known shape such that it best matches with the data. This approach is useful when there are strong indications that the shape assumption is justified. The nonparametric approach does not make this assumption and aims to capture the distribution of values more directly from the data instead of matching a rigid shape to the data. We use the nonparametric approach here because we do not have strong hints on the shape of the waiting time distribution. Nonparametric density estimation has received considerable attention from statisticians [23][22][25].

The most simple nonparametric density estimation is a *histogram*. This well known method subdivides a value range in intervals or *bins* and counts the number of observations that fall into each bin. Although easy to construct, a histogram has the disadvantage of providing a discontinuous (not smooth) density function and a density shape that may depend strongly on the choice of the origin of the bins, even when using the same bin width.

#### 3.1. Kernel density estimation

Another well known method for nonparametric density estimation – the one we use for our event prediction – is *kernel density estimation* (KDE). We give a short overview of kernel density estimation in this section because it is one of the basic tools we use for on-line event prediction in a multi-dimensional event stream. For a more elaborate discussion on kernel density estimation, see [23] and [11].

Consider a sample generated by observing a single random variable. From this data set, construct a set of functions, one for each observation, so that the sum of all functions in this set defines the probability density estimation for the variable. Each function is essentially weighing the contribution of a single observation to the overall density. Such a function is called a *kernel function*, often notated as  $K$ , and has the following property

$$\int_{-\infty}^{\infty} K(x)dx = 1, \quad (3)$$

so that a kernel function is a probability density function itself. A kernel function typically is symmetric and smooth, resulting in a smooth overall density function. Common shapes for  $K$  are Gaussian, Epanechnikov [7] and triangular.

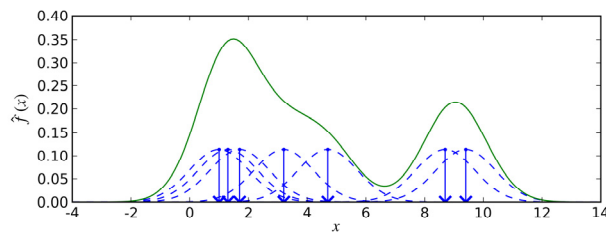


Figure 4: Construction of a density estimate (continuous line) by summation over a set of weighted Gaussian kernels (dashed lines), each associated with an observation (marked on the x-axis). The smoothing parameter  $h$  is set to 0.5.

Figure 4 shows the construction of a density function from the kernel functions associated with individual observations, using a Gaussian kernel. The probability density  $f(x)$  is estimated from a data set  $\{x_i\}$  by

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right), \quad (4)$$

where  $n$  is the number of values in the data set and  $h$  is the smoothing parameter, also called the *window width* or *bandwidth*.

The above equation has two parameters to choose: the kernel and the smoothing parameter. As illustrated in [23], the kernel shape has remarkable little influence on the approximation performance and therefore may be chosen based on other criteria such as the computational cost. Unless specified

otherwise, we use the Gaussian kernel. The smoothing parameter, in contrast, has a large impact on the performance of the estimator and therefore must be selected with care. It determines the width of the kernel which delivers a highly smooth density for large values and a less smooth density for low values. With a large data set, the optimal smoothing parameter is expected to become small.

A frequently used, elegant method for determining the optimal smoothing parameter is called *likelihood cross-validation*. This method is based on the idea of using the likelihood of the parameter values of a statistical model or hypothesis given the data, to serve as a measure for comparison with other values of the parameters; then the values that deliver the maximum likelihood provide the best fit. The principle of maximum likelihood has applications in a wide range of domains.

Suppose we have a density estimate  $\hat{f}_h$  for a random variable and a set of observations  $\{x_i\}$ . The likelihood of this sample depends on  $h$  and is given by  $\prod_i \hat{f}_h(x_i)$ . The estimate itself, however, also requires a sample to determine, and usually we do not have multiple sets of observations. This problem can be solved by using the same set as input to the density estimate and as input to calculate the likelihood. Applying the likelihood cross-validation method to find the optimal smoothing parameter now consists of the following steps. First, determine the likelihood of each observation in the data set with a density estimate that uses all  $(n-1)$  *other* observations. This measures how well an independent observation matches with an estimate based on  $(n-1)$  other observations, repeated for all observations in the sample because no single observation has more relevance than another. Second, calculate the logarithm of the likelihood of the entire data set as the sum of the logarithm of the likelihoods of each observation. Using the logarithm of the likelihood here is more convenient than the likelihood, because the likelihood tends to become very small, being a product rather than a sum. The normalized log likelihood of the data set is then

$$L(h) = \frac{1}{n} \sum_{i=1}^n \log(\hat{f}_{-i,h}(x_i)) , \quad (5)$$

where  $\hat{f}_{-i,h}$  denotes the *leave-one-out* kernel density estimate with smoothing parameter  $h$  using all observations except observation  $i$ . The third step consists of using  $L$  to compare different values for  $h$ : the  $h$  value that provides the maximum log likelihood value is optimal.

Kernel densities based on likelihood cross-validation have a number of disadvantages. A density estimate may become oversmooth in case the data set contains an outlier value, especially for kernel functions with finite support. The likelihood of an outlier  $x_i$  calculated with  $\hat{f}_{-i,h}(x_i)$  may easily become 0, and therefore result in  $L(h) = -\infty$ , unless  $h$  is sufficiently increased. Maximizing the likelihood therefore obtains a smoothing parameter that is large enough to bridge the gaps between data points, even if some of the gaps are disproportionately large. Related to this, and of a more fundamental theoretical concern, are the inconsistent estimates of the density in case of  $n \rightarrow \infty$ : the expected value for  $h$  would then go to 0. For many real distributions with a tail, however, this is not the case, because also for large  $n$  there may be large gaps in between the data points in the tail. Loosely speaking, using kernels with a ‘light tail’ will result in an oversmooth density estimate for densities with a ‘long tail’. A possible solution to this problem is the usage of a variable width kernel which applies less smoothing if the local density of the data is high and vice versa [1].

We use the likelihood cross-validation method here, primarily because it has significant computational benefits compared to other methods such as the *least squares cross-validation*, and has a known computational optimization for the case of kernel conditional density estimates [12] – discussed later in section 6. Furthermore, using likelihood cross validation on bounded densities does result in consistent estimates in the asymptotic case [3]. Our waiting time observations have a natural limit because the total observation time is always bounded, which means that even if the real distribution of waiting times has a long tail, the values far in the tail will never be observed. Nevertheless, outliers may still occur and therefore it is good to carefully consider the range of possible  $h$  values.

For the illustration of the density estimation methods discussed in this section, we introduce a synthetic example data set consisting of 15 visibility intervals of a network entity, as observed on the mobile device of a person. Each interval has a begin time and an end time, expressed as time *relative* to the day of the begin time, and each interval has a duration of less than 24 hours. This entity is visible most of the time during the evening and all of the time during the night, and can be thought of as a wireless home network. We call this data set `SynthSet-I` (see Figure 5).

The method of kernel density estimation using likelihood cross-validation can be applied on this data set. Figure 6 shows the estimated densities for the interval begin time  $t_b$  (a) and the interval end time  $t_e$  (b),

expressed as time relative to a period of 24 hours, as well as the estimated density of the duration of intervals ( $t_e - t_b$ ) in hours (c). The kernel is Gaussian and the optimal value of the smoothing parameter is 0.7, 0.9 and 1.1 respectively, calculated by comparing  $h$  values in the range  $[0.1, 10.0]$  with a step size of 0.1.

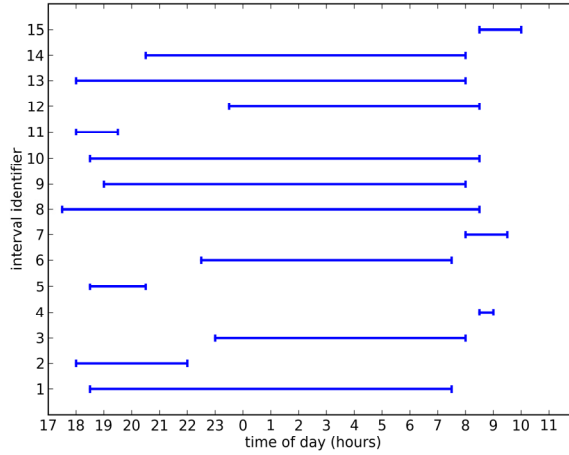


Figure 5: Synthetic data set SynthSet-I consisting of 15 visibility intervals

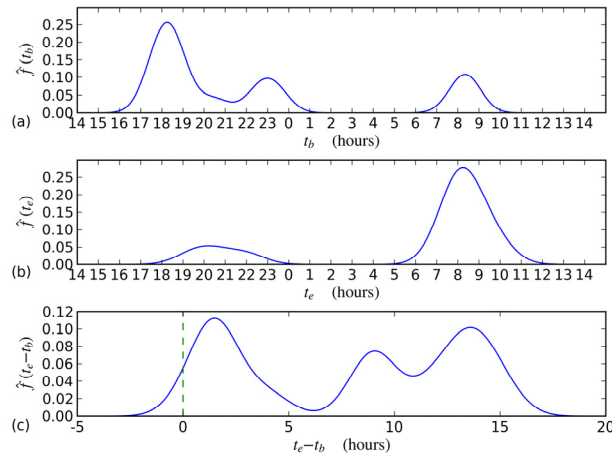


Figure 6: Probability density estimates for  $t_b$  (a),  $t_e$  (b) and  $t_e - t_b$  (c) for the SynthSet-I data set intervals, using a kernel density estimation with a Gaussian kernel, and corresponding optimal  $h$  values of 0.7, 0.9, and 1.1

The duration density in Figure 6c shows a shortcoming of kernel estimates for densities that are naturally bounded on one or both sides. It is clear that the duration of an interval can not have a negative value, and consequently, the density estimate should be 0 for  $(t_e - t_b) < 0$ . The estimate in Figure 6c, however, erroneously shows a positive density for negative durations. This shortcoming normally occurs only when there are observations close to the boundary and the smoothing parameter is not sufficiently small. Although existing approaches tackle this problem successfully, they come at a cost of, for instance, a more elaborate kernel or the insertion of phantom data (e.g., [5]).

So far, our discussion has focused largely on determining the optimal smoothing parameter. An open issue that remains is the difference between a calculated estimate with an optimal smoothing parameter and the real density. Unfortunately, there is little theoretical work on computing confidence bands using only the available observations, so instead of focusing on the difference between the real and the estimate density, our attention goes to formulating multiple models and then select the best model using its likelihood.

### 3.2. Kernel conditional density estimation

As explained in the outline of our approach on predicting events in a stream, we are interested in the waiting time distribution for an event given the occurrence of another event relative to the stream period. To estimate the waiting time density in a nonparametric fashion, we therefore need the conditional variant of kernel density estimation, called *kernel conditional density estimation* (KCDE). Although the body of work covering the theory on conditional estimates is much smaller than regular kernel densities, and only recent advances have provided an outlook on efficient computational approaches, it is now well enough established to be used here as a tool.

To a large extent, the mechanisms found in KDE are also used for KCDE. The conditional density is estimated with a double kernel, so that for a given set of observations  $\{x_i, y_i\}$  it is written as

$$\hat{f}_{h_1 h_2}(y | x) = \frac{\sum_{i=1}^n K((y - y_i)/h_1)K((x - x_i)/h_2)}{\sum_{i=1}^n K((x - x_i)/h_2)}, \quad (6)$$

where  $n$  is the number of value pairs in the data set and  $h_1$  and  $h_2$  are the smoothing parameters for  $y$  and  $x$ . This form is known as the Nadaraya-Watson estimator (see [22], pp. 220). Other more elaborate forms exist with potentially better properties [6]: for simplicity, however, we use this form here and in the remaining sections. Now, finding the best conditional density translates to selecting the optimal value of these two parameters. As a method to do this, we again turn to likelihood cross-validation which – for the conditional case – has the following normalized log likelihood of the data set

$$L(h_1, h_2) = \frac{1}{n} \sum_{i=1}^n \log(\hat{f}_{-i, h_1, h_2}(y_i | x_i) \hat{f}_{-i, h_2}(x_i)), \quad (7)$$

with  $\hat{f}_{-i, h_1, h_2}$  the leave-one-out kernel conditional density estimate with smoothing parameters  $h_1$  and  $h_2$  using all observations except observation  $i$ , and  $\hat{f}_{-i, h_2}$  the regular leave-one-out kernel density estimate using smoothing parameter  $h_2$  (Equation 5). It is based on the probability  $P(x_i, y_i)$  of the combined occurrence of the two events.

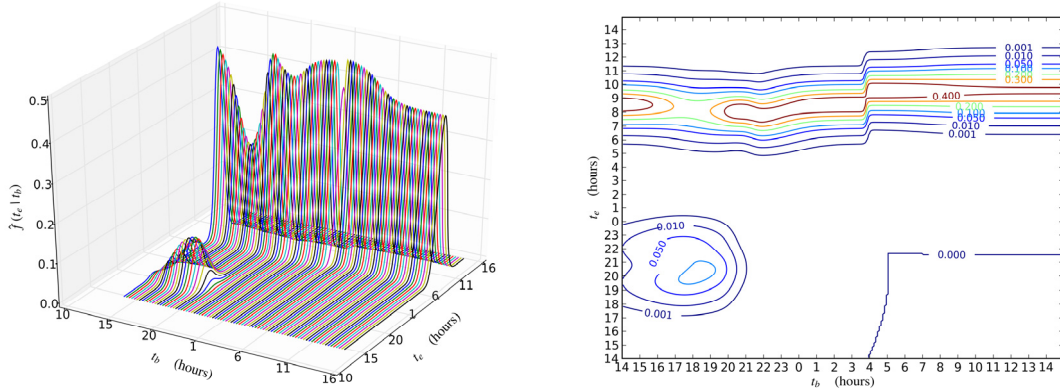


Figure 7: 3D plot (left) and contour plot (right) of the kernel conditional density estimation of  $f(t_e | t_b)$  for the SynthSet-I data set intervals, using a Gaussian kernel and optimal parameters  $h_1 = 0.8$  and  $h_2 = 0.7$

When applying this equation to the data set SynthSet-I, the optimal parameter values are  $h_1 = 0.8$  and  $h_2 = 0.7$ , resulting in a conditional density as depicted in Figure 7 – in a three dimensional form and a contour form. These plots clearly show that the shape of the interval end time event density varies considerably given different begin times and therefore offers more information than can be extracted from the accumulated density in Figure 6b. The density is estimated using a conditional begin time value relative to a 24 hour period in the input stream.

Note that the conditional density at the left-hand side is not the same as the density at the right-hand side of the plots in Figure 7 – when wrapping the plots around in the horizontal direction we would expect a smooth period boundary transition. This anomaly happens because we map all observations to a single

period without compensating for the kernel bandwidth looking beyond the boundaries of this period. By mirroring the data set to period -1 and period +1, however, at least for those observations that are within the bandwidth, we obtain a conditional density estimate that can be wrapped around the conditional axis (not depicted).

#### 4. Predictor selection

In this section, we assume we have a set of simple models (or predictors) in the form of probability densities that predict the time of a future occurrence of an event we are interested in. These models estimate the waiting time given the occurrence of a *single* preceding event in the input stream. Using KCDE, we can construct a set of conditional density estimates – one for every conditional preceding event – to predict this event. Or, using straightforward KDE, we can generate density estimates for the (unconditional) duration between the preceding event and the event of interest. This section discusses the dynamic selection over time of the best predictor from this set of available models. It introduces a number of strategies to dynamically select the best predictor (i.e., reevaluating predictor performance during the interval between two occurrences of the event of interest), and defines metrics to compare these strategies.

In the previous section, a number of different models were introduced that predict the end time of the intervals in the `SynthSet-I` data set. The model in Figure 6b simply shows the probability density of the occurrence of the interval end time on a 24 hour period. Assuming that intervals never have a duration longer than one day, we can estimate, at the moment the interval begins, when during the next 24 hours the interval will end – irrespective of the time value of the beginning. The model in Figure 6c uses the relative duration of an interval, again not depending on the begin time value. The model in Figure 7 predicts the end time value given – as a conditional – the begin time value on a 24 hour period.

A fundamental statistical method to compare different models is to determine the likelihood of each model given available data and then select the model with the highest likelihood as the preferred model. When applying this method of *maximum likelihood* to our models, the data in `SynthSet-I` is used to calculate the density of the end time for every model, at the moment the interval begins: the likelihood of the data is then the product of all end time densities. To prevent working with extreme small values, however, most commonly the logarithm of the likelihood is used so that the log likelihood is the *sum* of the log of – in this case – the end time densities. Now, when calculating the log likelihood of the data set for the available models, the first model has a value of -27.1, the second a value of -37.5 and the third a value of -19.8. This suggests that the conditional density of the third model delivers the best prediction performance, although for a more careful evaluation we may need to use two separate data sets: one to estimate the densities and another to calculate the log likelihood. This example shows the use of the maximum likelihood method to select the best model out of a set of available models.

##### 4.1. Strategies

Considering the more general case, with multiple preceding events such as illustrated in Figure 3, a density estimation associated with a preceding event only applies when this preceding event has occurred. So, contrary to the example above, the available models are not always applicable; they can be considered only at the moment that the associating event occurs. As time proceeds and more preceding events take place, the set of applicable models is therefore expected to grow. The question we want to address now is: given a set of applicable models, each associated with a single preceding event, which one of these models is the best predictor of the event of interest? Or, more precisely, without determining the multivariate remaining waiting time conditional density  $f_{r,i}(t|z_k, z_l, z_m, \dots)$  for event of interest  $z_i$  with preceding events  $z_k, z_l, z_m, \dots$ , which of the univariate estimated conditionals densities  $\hat{f}_{r,i}(t|z_k)$ ,  $\hat{f}_{r,i}(t|z_l)$ ,  $\hat{f}_{r,i}(t|z_m)$ , ... is closest to the multivariate conditional density?

From a theoretical perspective, it is possible to select between two models in the special case when one event  $z_k$  always occurs in conjunction with another  $z_l$  (but not necessarily vice versa). In that case, the univariate  $\hat{f}_{r,i}(t|z_k)$  is the same as the multivariate  $\hat{f}_{r,i}(t|z_k z_l)$  so that it provides the desired model. The number of observations in the input stream of event  $z_k$  is then always less than the number of observations of event  $z_l$ . This principle of *single-sided concurrence* can be extended to more than two events, requiring that if one event is observed, also all other events are observed. If events in an input

stream do have many of these particular inter-relationships, this characteristic could be exploited for model selection.

When considering an input stream reflecting human mobility, the time between two often observed events during the traveling of a person can be highly constant (i.e., can have a low variance). For example, the occurrence of an event observed on the way from work to home may be a good predictor for the arrival at home. So, the variance of a predictor may be a good indicator of the performance of that predictor – being closer to the multivariate conditional density than predictors with high variance. Also, in this case the most recent preceding events may be better predictors than those happened longer ago. Note, however, that both low variance and event recentness are *not guaranteed* to be good indicators for prediction performance: if they are, it is a heuristic that applies only to certain kinds of input streams. With single-sided concurrence, for instance, the least frequent event may predict with a higher variance than the most frequent event, even though it is the better predictor.

In the above maximum likelihood example, the prediction of the end time of an interval is measured with a data set consisting of begin and end time pairs. For every data pair  $\{t_b, t_e\}$ , it essentially answers the question: given this begin time  $t_b$ , what is the probability of seeing the end time  $t_e$  for each of the models? This implies that we are only interested in predicting the end time at the moment an interval begins. In reality, however, we may want to predict the end time at several moments before the end of the interval, using the residual waiting time of Equation 2. Model performance may change considerably with increasing time in an interval, because the shape of the residual waiting time may change considerably, for instance by bringing forward features in the tail of the waiting time density that only appear when scaled up sufficiently. Now, translating to the general case with multiple preceding events, this adds another reason to selecting models dynamically as time proceeds.

When calculating the performance of the models at different moments in time (i.e., not only at times when events occur), we can build up a history of model performance, for instance based on the likelihood. Then, for every  $f_{r,i}(t | z_j)$ , we keep track of the mean likelihood, or, alternatively, the median likelihood.

Taking the above discussion in consideration, we define the following strategies for dynamic model selection for the prediction of event  $z_{i,x}$ :

1. *Self conditional*. This strategy serves as a baseline for comparison with the other more sophisticated strategies. It uses the previous occurrence of the event of interest ( $z_{i,x-1}$ ) as a conditional event. As such, it uses the minimal amount of historical information available from the input stream.
2. *Last conditional*. Using the most recent preceding event as a predictor: the event  $z_j$  for which  $t_{i,x} - t_{j,y}$  is minimized. Contrary to self conditional, this strategy uses different conditional events as time progresses and new events occur.
3. *Lowest variance*. From the set of available predictors, select the one which has, at the time of evaluation, a residual waiting time with the lowest variance. This is event  $z_j$  for which  $\text{Var}(\hat{T}_{r,j})$  is minimal.
4. *Event concurrence*. Filter out the predictors based on events that always concur with other preceding events (see text above for a more precise definition). Use lowest variance selection on the reduced set of predictors.
5. *Best likelihood*. For every  $f_{r,i}(t | z_j)$ , keep track of the likelihood at previous predictor evaluation moments. Use the predictor which has the highest mean likelihood, or, alternatively, the highest median likelihood.

These strategies are evaluated in the next section, using real-world traces.

#### 4.2. Prediction performance metrics

A common metric to compare the prediction performance of different models is, as discussed in the example above, the likelihood of a relevant data set for each model. This method has as practical disadvantage that for some observations the probability might be zero, which would result in a log likelihood with a negative infinite value. To deal with this problem, a minimum probability is used during likelihood calculation. Still, a few observations with zero probability may considerably influence the overall likelihood of a data set for a given model.

An alternative usage of the likelihood as a metric is to compare for every model the probability of each observation and mark the model with the highest probability as the winner. Then, the model with the highest winner fraction – the number of times the model is a winner divided by the total number of

observation – is the best model. This metric does not provide a ‘weighted’ likelihood, but is less susceptible to zero probability influences.

Another very common metric to determine the prediction performance of a model is the *mean squared error* (MSE) which is defined as the average of the square of the error. Here, the error is the difference in time between the expected occurrence time of the event of interest and the observed occurrence time. Using Equation 2, the expected occurrence time is given by

$$E(T_{r,i}) = \int_{t_p}^{\infty} t \cdot f_{r,i}(t | z_j) dt \quad (8)$$

By taking the square over the error, outlier values of the expected error strongly influence the overall MSE value for a model. Also, if the residual waiting time is far in the future, the error is likely to be larger than with a short waiting time, which makes MSE more suitable to measure long term prediction performance than short time prediction performance: the long term errors overshadow the short term errors. As an alternative we can use the *mean absolute percentage error* (MAPE), defined as

$$\frac{1}{n} \sum_{i=1}^n \left| \frac{\hat{x}_i - x_i}{x_i} \right|, \quad (9)$$

which measures the relative error and therefore balances long and short term errors. Like MSE, it uses the difference in time between the expected occurrence time of the event of interest and the observed occurrence time.

#### 4.3. Measurement points

A final aspect to take into account when selecting the best predictor – next to the applied selection strategies and the used performance metric – is the *distribution of points on the timeline where the prediction performance is measured*. A straightforward approach is to measure at regular intervals, until the event of interest has occurred. If, however, prediction performance is more important when the residual waiting time is becoming shorter, i.e., we want our model to predict better if the event of interest is imminent, more measurement points close to the event of interest provides a stronger emphasis on short term prediction performance. In that case, a set of measurement points with increasing distance between them when counting backwards from the observed event time is more suitable. Ideally, the measurement points are determined by the application using the predictions.

## 5. Prediction results

In this section, we provide the results on predicting network entity visibility in data streams collected on the personal devices of real users, using the approach described in the previous sections. These streams represent the visibility events of heterogeneous network entities, i.e., entities such as access points and base stations as seen through the wireless interfaces available on the devices. One objective of this section is to compare the different predictor selection strategies introduced in the previous section, using real world data. Another objective is to study the influence of the amount of data available to construct a predictor: if a predictor is based on a pair set with a small size, is it still worth to take into consideration? A third objective is to investigate whether the prediction of events stemming from a single sensor (i.e., a single network interface) benefits from taking into account predictors that use information from other sensors (other network interfaces).

We focus on the prediction in time of network entities getting in range and getting out of range, using two data sets: the data collected during our own CoSphere trial and the public data set from Rice University, described in [21]. The CoSphere data set [19] was collected with the middleware software implementing the CoSphere NAL [18], gathering data on cellular, 802.11 Wireless LAN, and Bluetooth networks in range. The duration of this trial was approximately one month and had 12 participants carrying their mobile device in their every day activities. The data set can be obtained from [4]. The Rice data set was generated in a similar experiment with almost the same duration and number of participants, gathering cellular and 802.11 visibility data at a higher scan rate than in the CoSphere trial. It does not, however, contain Bluetooth information. Another reason for using information from all network

interfaces is to study whether events related to one network technology can be used as predictors for events of another technology.

As described in the previous section, many parameters play a role in the calculation of the prediction of events and evaluation of the prediction performance. The kernel density estimation needs smoothing parameter boundaries and can be calculated with different (optimizing) algorithms. The data stream may be preprocessed in various ways. The measurement points used to calculate a prediction may be selected in many ways. It is clear that we can not exercise all parameters values in all combinations, and therefore, we make choices on the settings of these parameters in the different parts of this section.

The outline of the section is as follows. Section 5.1 presents the main results of the strategy comparison – based on cross validation – and provides a description of the method and algorithm used to compare the prediction performance. It exercises all the strategies and metrics defined in the previous section. Section 5.2 shows the influence of using different sets of preceding events on the prediction performance. It addresses the question whether it is worthwhile in term of prediction performance to add more event data – also those events that occur infrequently. Section 5.3 discusses the influence of cross-technology information on the prediction performance, making a distinction between cellular, 802.11 and Bluetooth events. Section 5.4 shows the prediction performance of a number of strategies as it evolves over time; this is relevant for most real world scenarios.

### 5.1. Comparing strategies for predictor selection

We want to determine which predictor selection strategy, as described in the previous section, delivers the best prediction performance for the visibility of network entities as observed over time by a single person. When comparing strategies on a real-world data set, we are interested in the *aggregated performance* over all potential events of interest, not just one of them. As with the selection of measurement points, the selection of potential events of interest is application specific: some applications may need predictions for a highly selective set of network events that are best predicted with one strategy, while other applications require predictions for a different set of events that are best predicted with another strategy. Here, we determine the prediction performance based on broad sets of events of interest and broad sets of preceding events, providing a strategy comparison based on large portions of the data set. We vary the set sizes by working with different threshold values representing the minimum number of times an event has occurred during the time of the experiment. It is good, however, to keep in mind that this provides a generalized comparison of the strategies, and that application specific event sets may lead to different results.

#### 5.1.1. Single event prediction performance

In a running system environment, the calculation of the prediction performance for a *single* event  $z_i$  at a *single* measurement point  $t_p$ , based on a selection of possible preceding events, is determined by executing a number of steps.

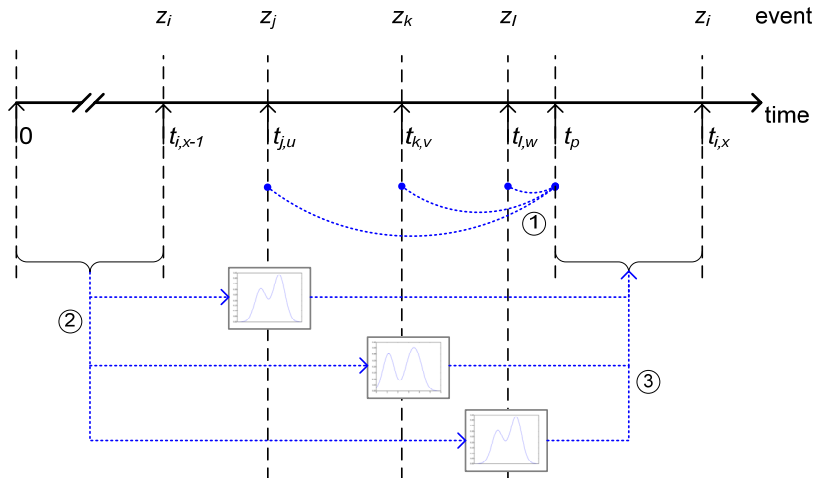


Figure 8: First three steps of calculation of the performance of a single event occurrence prediction

The description of the steps is given below. Steps 1, 2 and 3 are illustrated by Figure 8.

1. First, identify the preceding events that have occurred since the previous occurrence of the event of interest  $z_i$  (between the times  $t_{i,x-1}$  and  $t_p$ ).
2. Then, if not done already at a previous measurement point in between  $t_{i,x-1}$  and  $t_{i,x}$ , calculate the best smoothing parameters for the waiting time probability density estimations  $\hat{f}_{w,i}(t | z_j)$  for every preceding event  $z_j$ . (see Definition 1). Per  $z_j$ , there are two estimates: one KDE estimate for the relative time between the two event times  $t_{j,y} - t_{i,x}$  (the *relative predictor*, Equation 4), and one KCDE estimate for the period time of  $z_i$  expressed as time relative to the current period of  $z_j$ , using the time of  $z_j$  on its own period as a conditional (the *periodic predictor*, Equation 6).
3. For all occurred preceding events, compute the relative and periodic *remaining* waiting time estimates at time  $t_p$  of the measurement point, using their waiting time estimates (Equation 2).
4. For each strategy under consideration, select the best predictor according to that strategy. For the strategies ‘lowest variance’, ‘event concurrence’, and ‘best likelihood’, this requires additional computation.
5. Per best predictor, calculate the performance using the set of defined metrics. The metrics ‘maximum likelihood’ and ‘likelihood winner’ only need one calculation of the estimated probability density function, but metrics MSE and MAPE use the expected occurrence time of  $z_i$ , by taking the integral as in Equation 8.

In a running system environment, the smoothing parameter calculation in step 2 can only take into account the data seen so far in the stream, i.e., the data observed up to the point in time of the previous occurrence of the event of interest (at  $z_{i-1}$ ). Going over the stream in this way gives an indication of the prediction performance evolution in time: we investigate the performance evolution in section 5.4. Here, however, we are interested in determining the best strategy using as much data of the whole stream as possible. Therefore, we do not restrict the data used for calculating smoothing parameters to be chronologically before the measurement point.

#### 5.1.2. Cross validation approach

A common way to validate a model given a data set is to divide the available data into a subset used to build the model (the *training set*) and another subset used to measure the performance of this model (the *validation set*). To use the available data as much as possible, the whole data set is often broken up in many subsets, where each subset is used once to validate and the rest of the subsets are used to train. This cross validation approach (see [26], pp. 149) is also applied in the leave-one-out method that finds the optimal smoothing parameters for KDE/KCDE (Equation 5 and Equation 7).

To apply this cross validation approach to the prediction of a single event, we must subdivide the stream into parts. A logical subdivision is to treat the data for each time interval between two occurrences of this event as a subset, so that the number of subsets equals the number of event occurrences. It is not possible to further break up these subsets, because in that case the training set would contain event data about the exact event occurrence we want to predict. It is possible to use larger subsets by breaking up the data in a fixed number of subsets – so called *k-fold cross validation* – where  $k$  is smaller than the number of occurrences of the event of interest. This may be accomplished by joining the data of some of the intervals. Although the amount of cross validation is constant, the  $k$ -fold approach wastes at least a part of the data (i.e., not all data of all other intervals is used to predict events in a particular interval). Therefore we choose to use the minimum size subsets, even though it is more expensive to compute: for every  $z_j$ - $z_i$  pair, the worst case number of times the smoothing parameters must be recalculated equals the number of occurrences of  $z_i$ . For step 2 in the above calculation, it means that we use all data in the stream that is not in the  $t_{i,x-1}$  and  $t_{i,x}$  interval, to determine the best smoothing parameters. We are interested in the aggregated prediction results for the entire set of events of interest we are considering, so we repeat the above cross validation for all  $z_i$  and average the results.

#### 5.1.3. Measurement points

When using a regular distance scheme or a backward increasing distance scheme for the determination of the measurement points, the number of measurement points increases with a rise in the number of events of interest. To limit the necessary computing time, we therefore determine a fixed number of measurement points regardless of the number of events of interest. In this paper we use a statistical backward increasing distance scheme with an exponential distribution equal to  $\lambda e^{-\lambda x}$ , with  $\lambda = 1.5$ , and we draw 2000 measurement points per participant. This may mean, especially for a larger number of

events of interest, that some event intervals do not have any measurement points. The average time between the measurement point and the to-be-predicted event is less than the average distribution value of  $1/\lambda$  hours (40 minutes), because there is bounded time between consecutive occurrences of an event. Note that, as with the training sets and validation sets, the measurement points are specific for a single event of interest, i.e., they are not used to determine the prediction performance of all events of interest at that point in time.

#### 5.1.4. Strategy comparison algorithm

Now the evaluation method to determine the best prediction strategy is summarized as follows. We calculate performance results per event of interest, using interval cross validation. The measurement points at which the calculations take place are distributed such that the distance between a measurement point and the next occurrence of the event of interest is exponential with more measurement points close to the event of interest. We use a fixed number of measurement points per participant, and apply every predictor selection strategy at every measurement point and average their performance over all measurement points. Then these averaged results are used to compare the performance of a strategy, for every participant. Comparing the strategy performance for all participants provides an indication of the general performance of a strategy.

The comparison consists of two steps. First, for every measurement point, we compute a few basic properties for every available predictor at the time of that measurement point. These properties are the log likelihood, the remaining waiting time mean and the remaining waiting time variance, both for the relative and the periodic variant of the predictor. Second, we use those basic predictor properties to apply the predictor selection strategies and to calculate the performance of each strategy. We use a minimum log likelihood of -10.0.

The strategy ‘self conditional’ uses the previous occurrence of the event of interest to predict the next occurrence. As we saw, we have a relative and periodic predictor per preceding event, which means that for this strategy there are always two predictors. Therefore, for this strategy we report on the results for the relative and periodic predictors separately. A similar consideration is true for the ‘last conditional’ strategy. An overview of the considered strategies and their predictor selection during the second step of the strategy comparison is given in Table 1. This table also shows the strategy abbreviations we will use from now on.

Table 1: Predictor selection per strategy during the second step in the strategy comparison algorithm

<i>Strategy</i>	<i>Abbreviation</i>	<i>Predictor</i>
Self conditional (relative)	selfcondr	KDE based on previous occurrence of event itself
Self conditional (periodic)	selfcondp	KCDE based on previous occurrence of event itself
Last conditional (relative)	lastcondr	KDE based on most recent preceding event
Last conditional (periodic)	lastcondp	KCDE based on most recent preceding event
Lowest variance	lowestvar	Predictor (KDE or KCDE) with the lowest remaining waiting time variance
Event concurrence	evtconcur	Same as lowestvar, but only for preceding events that do not always concur with other preceding events
Best likelihood	bestll	Predictor (KDE or KCDE) with highest average log likelihood

To be able to filter out the preceding events that always concur with other preceding events, for strategy `evtconcur`, it is necessary to keep track of the mutual occurrence of every event pair. For our comparison based on cross validation, this means a recalculation of the event concurrence for every new interval, using all other intervals (the training set).

Similarly, with `bestll`, we must keep track of the previous performance of each predictor, by calculating the average likelihood for a predictor at the measurement points that fall in the same time range as the training set. For this, we use the likelihood values determined during the basic properties step in the intervals in the training set. Note that these values are calculated using the stream data in the other

intervals, including the interval for which we want to determine the average likelihood of the predictor. An effect of this approach is that the performance of a predictor is suppressed when the to-be-predicted event is an outlier. Therefore, this effect may have an influence on the performance of the `best11` strategy in cross-validation scenarios. An alternative approach would be to recalculate the likelihood at a measurement point in a training interval using only the data in other training intervals. Unfortunately, this requires additional  $(n-1)$  smoothing parameter calculations per interval, with  $n$  the number of event occurrences, lengthening the computing time with one to two orders of magnitude. The average likelihood must be recalculated – during preprocessing – for every new interval. We will see that the above effect is minimal when comparing the `best11` strategy performance evolution with the cross-validated performance.

### 5.1.5. Event occurrence count

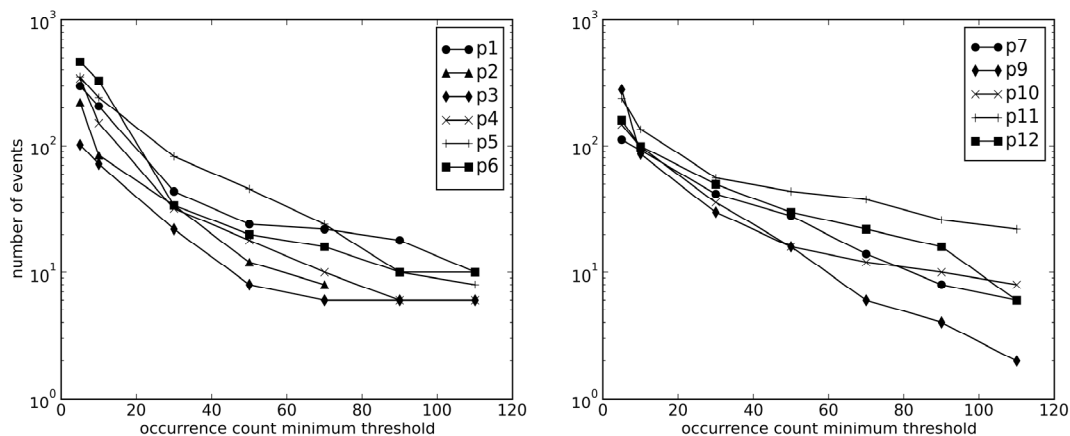


Figure 9: Number of events (log scale) with an occurrence count above different threshold values, for the participants in the CoSphere data set. Label p1 stands for participant 1, p2 for participant 2, etc.

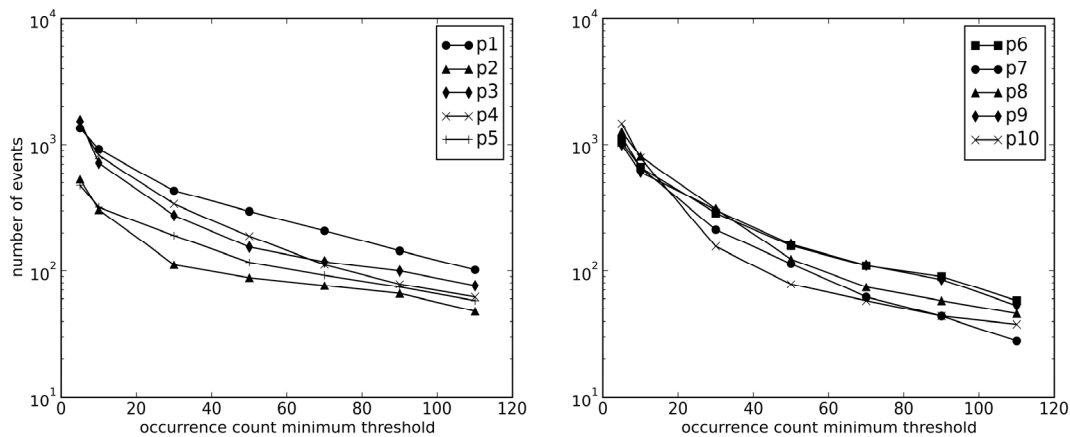


Figure 10: Number of events (log scale) with an occurrence count above different threshold values, for the participants in the Rice data set. Label p1 stands for participant 1, p2 for participant 2, etc.

Compared with the `SynthSet-II` data set, the `CoSphere` data set and especially the `Rice` data set have many more events and also more frequently occurring events. Figure 9 depicts the number of events that have an occurrence count above a given threshold value, for the participants in the `CoSphere` data set. It uses two subfigures to more clearly show the number of events for individual participant. For a threshold value of 110, there are only in the order of 10 events with a higher occurrence count for the participants. A threshold value of 5, however, provides in the order of several hundred qualifying events per participant. Figure 10 shows the same information for the `Rice` dataset. The number of event

occurrences for this data set is substantially larger, most likely because the Rice experiment sampled for in-range networks with a higher frequency. The occurrence count has substantial influence on the computation cost (see Section 6).

#### 5.1.6. Strategy comparison results

We have executed the strategy comparison for different combinations of the set of events of interest and the set of preceding events using the `CoSphere` and `Rice` data sets. The size of both event sets is determined by the occurrence count threshold applied for that set: only those events in the stream with an occurrence count above the threshold are part of the set. Obviously, the sets are equivalent in case both threshold values are the same. The `CoSphere` stream contains cellular, 802.11 and Bluetooth network events and the `Rice` stream cellular and 802.11 network events, so that the prediction of individual events of interest may be based on preceding events from any of these technologies. Apart from not containing Bluetooth events, the `Rice` data set differs from the `CoSphere` data set in the following respects: it has a higher sampling frequency, reporting on network visibility every minute, and it captures information about all in-range cellular base stations – not only the actively used base station as with the `CoSphere` data set.

The stream data contains interruptions caused by device resets during the collection of the data [19]. We have preprocessed the stream such that short interruptions less than 15 minutes are considered to not change the stream state: we assume no events happen during these short interruptions. If the interruption is longer than 15 minutes, we assume we do not know the events during that time and consequently discard all stream data during the interruption and the time before the interruption from the current event of interest interval.

Some of the events in the stream show a high frequency alternating occurrence, in combination with one or more other events. This typically takes place in cellular networks, where a stationary device sometimes frequently switches between base stations. To dampen this effect, we limit the on-off behavior by removing from the stream short term off states with a duration less than 15 minutes. By greatly reducing the number of occurrences of some of the events in this way, we also limit the necessary computing time. The numbers in Figure 9 and Figure 10 refer to the situation after removing short interruptions and short-term off states. Note that this scheme does not influence the occurrence of events associated with network entities that are visible for a short time once during a longer period (i.e., visible shorter than 15 minutes), for instance in case of traveling. This information is retained in the stream.

Applying the above strategy comparison algorithm on the stream, we observe the following results. When considering a range of occurrence threshold values applied to both the set of events of interest and the set of preceding events, the `best11` strategy shows the best overall results for most metrics, although for higher threshold values `lastcondr` is catching up and exceeds `best11` for a threshold value of 110. Especially with metric `mse`, the `lastcondr` strategy is quickly becoming the preferred strategy for higher threshold values. Also for higher threshold values, the strategies `lastcondp` and `lowestvar` start being best for some participants.

These results can be observed in Figure 11 and Figure 12. It shows the number of participants that have a strategy-metric pair as best predictor, for the strategies `lastcondr`, `lastcondp`, `lowestvar` and `best11`, with different values for the occurrence threshold for both sets. The other three strategies are left out, because they are not once marked as a best predictor selection strategy. For the `CoSphere` data set, the total number of participants for every metric-threshold pair is 11, except for the threshold values 90 and 110, in which case this number is 10. This is caused by the fact that participant 2 does not have events with an occurrence count above or equal to 90 (see also Figure 9). For the `Rice` data set, the total number of participants is 10.

When looking at the second best strategy for predictor selection, the outcome is much more varied. For the case of an occurrence threshold value of 5, for instance, which is dominated by `best11` as the best strategy, the second best strategy is hard to determine because it is more or less equally spread over `lastcondr`, `lastcondp`, `lowestvar` and `evtconcur`. For specific metrics, some are preferred: the `evtconcur` strategy, for example, is best for the maximum likelihood metric.

One notable outcome of this comparison is that `evtconcur` is not always better than `lowestvar`, as would be expected from the previous section. It is better for some metrics, but not all, and it is never a best strategy, while `lowestvar` is a best strategy occasionally (with higher threshold values). This may be caused by the effect of frequently occurring preceding events being pushed away by far less frequently occurring preceding events, while the less frequently occurring events do not have enough data to form a

stationary estimate. In that case, under cross validation, the estimate changes considerably from interval to interval, resulting in a less accurate prediction.

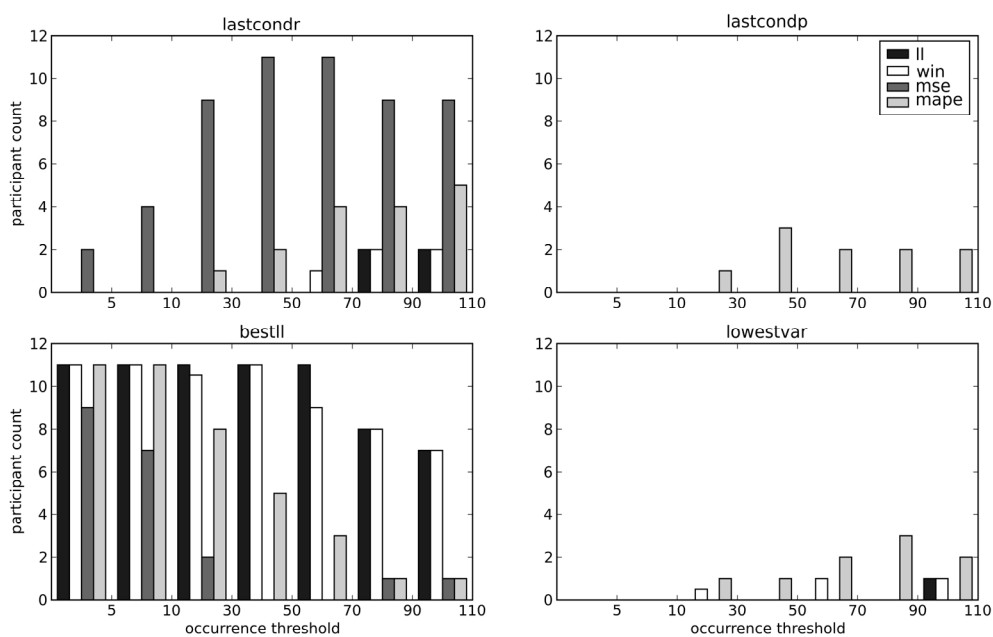


Figure 11: Comparison of the lastcondr, lastcondp, lowestvar and bestll strategies for all four metrics and for different values of the occurrence threshold for the CoSphere data set

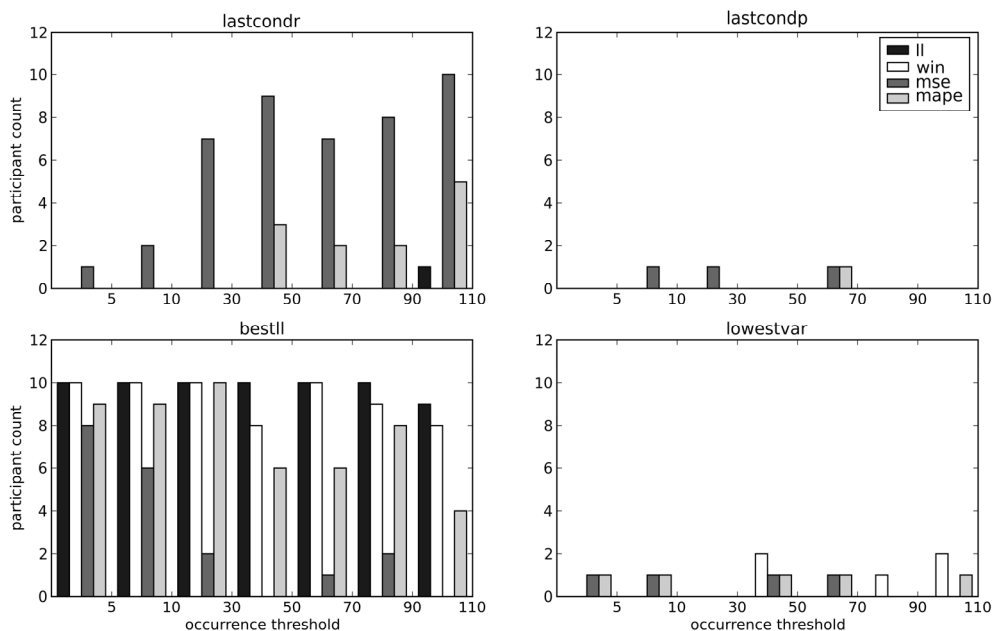


Figure 12: Comparison of the lastcondr, lastcondp, lowestvar and bestll strategies for all four metrics and for different values of the occurrence threshold for the Rice data set

Another notable outcome is the prediction performance of predictors based on relative estimates versus those based on periodic (daily) estimates. For the selfcond and lastcond strategies, overall the periodic variants do not provide better predictions than the relative variants and vice versa. The other

strategies sometimes choose a periodic predictor and at other time a relative predictor. The `lowestvar` and `evtconcur` strategies behave almost the same: they choose a relative predictor 50 percent of the times for an occurrence threshold of 5, dropping to 25 percent for an occurrence threshold of 110. The `bestll` strategy has a more stable relative predictor selection percentage of around 65 for all occurrence threshold values. This means that `bestll` considers a period predictor to have a better historical likelihood in 35 percent of the times, which is an indication that taking into account periodicity is improving the prediction performance. Note that we used the data of all available days in the `CoSphere` and `Rice` data sets, without making a distinction between work days and weekend days. Focusing on work days (or weekend days) may provide different results than we found for the complete data set.

#### 5.1.7. Comparison conclusions

The general conclusions from the cross validation strategy comparison are now the following. Overall, the best strategy for predictor selection is the `bestll` strategy: most convincingly for cases when we have a large set of events of interest being predicted by an equally large set of preceding events. For the metrics `mse` and `mape`, the `lastcondr` strategy is on par with `bestll`, and even surpasses it in case of highly frequently occurring events of interest and preceding events. Using a preceding occurrence of an event to predict the next occurrence of this event does not provide a good prediction, not for the relative nor for the period case. And finally, when comparing the periodic predictors with the relative predictors, none of these two predictor categories have a high prevalence over the other. These conclusions apply to both the `CoSphere` and the `Rice` data set.

### 5.2. Influence of different event set occurrence thresholds

We now investigate the influence of the sizes of the events of interest set and the preceding events set on the prediction performance. We want to answer the following question: given a set of events of interest, what is the influence of enlarging the set of preceding events (i.e., essentially having more information) on the prediction of the events of interest?

We do this by taking the `bestll` strategy and applying the normalized log likelihood as metric, to measure, in a cross validated manner, the prediction performance with different occurrence threshold values. As with the strategy comparison, we use the events of all three network technologies available in the original `CoSphere` stream and the events of the two network technologies in the `Rice` stream and use occurrence threshold values that are in [5, 10, 30, 50, 70, 90, 110].

When using more information here to predict an event of interest, in case of a growing data set caused by a decreasing occurrence threshold, we expect to see two opposing effects. The first effect is that more information allows for better prediction. The second effect is that predictors based on little historical data deteriorate the prediction performance. We describe both effects in more detail below.

#### 5.2.1. Effect of more information

The first effect is that more information allows for better prediction, because it helps to identify specific situations that would otherwise be unnoticed. For example, when a participant occasionally meets a person at the office at the end of a working day, indicated by the event of this person's Bluetooth device coming in range, it always takes longer before the participant arrives within range of his home network (perhaps because they join for an after-work activity). Now, if this Bluetooth event does not occur very often, it will quickly be filtered out by the occurrence threshold, and therefore not considered as a predictor.

#### 5.2.2. Effect of predictors with little history

The second effect is lack of stability of a predictor caused by an optimal smoothing parameter calculation with relatively few observations, which may mark a predictor as very good (high average likelihood for strategy `bestll`) simply because the few observations in the stream happen to be close to each other. In such a case, this predictor is chosen by `bestll` as the best predictor, while the next occurrence is likely to yield a low likelihood and not being predicted well at all. For instance, in the example above, all first few observations may indicate an arrival at home at 20:00 hours while the next observation shows an arrival time of 22:00 hours. Obviously, it is easy to set a minimum number of pairs required to have a valid predictor, but if set high, only the most frequently occurring events can serve as predictors. The cross validation performance algorithm applies a minimum of 3 pairs to calculate optimal

smoothing parameters, which allows for many predictors for all participants (for an occurrence threshold value of 5 the maximum number of predictors is close to 250 on average, as can be seen in Figure 9, and around 1100 for the `Rice` data set, as shown in Figure 10). Note that even when an event occurs 5 times in the stream, the number of pairs used to calculate the best smoothing parameters may be less than 4, because multiple pairs may be related to the interval for which we want to predict, and our cross validation approach only uses data from intervals other than the one we want to predict. If, in such a case, the number of pairs drops below 3, we do not consider that predictor.

### 5.2.3. Prediction results with variable event sets

The results of the prediction performance are depicted in the participant 3D grid plots in Figure 13 for participants 1 and 12 of the `CoSphere` data set, and in Figure 14 for participants 1 and 10 of the `Rice` data set. The plots of the other participants have comparable shapes.

The x-axis and y-axis of the plots indicate the occurrence threshold for the set of events of interest (`oc-evti`) and the occurrence threshold for the set of preceding events (`oc-evtc`), and have a value between 5 and 110. The z-axis marks the normalized log likelihood.

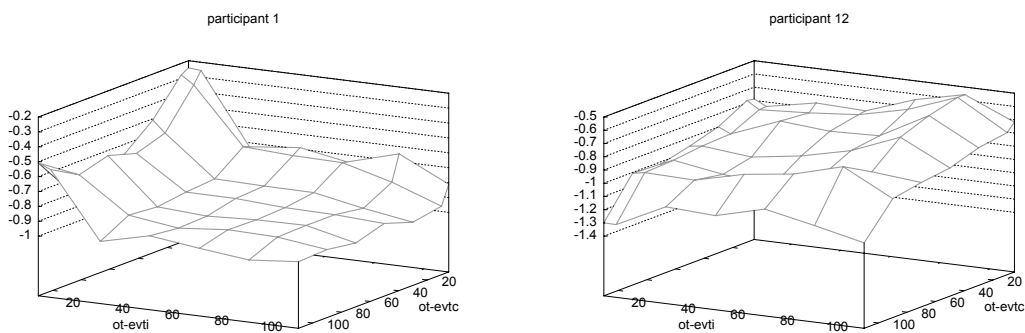


Figure 13: Normalized log likelihood of predictions with the `best11` strategy, for different combinations of the occurrence threshold values for the events of interest (`ot-evti`) and the preceding events (`ot-evtc`). Depicted are participant 1 and 12 from the `CoSphere` data set.

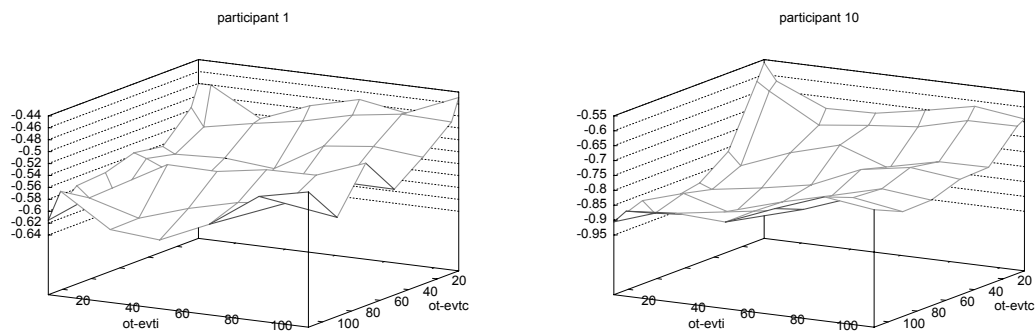


Figure 14: Normalized log likelihood of predictions with the `best11` strategy, for different combinations of the occurrence threshold values for the events of interest (`ot-evti`) and the preceding events (`ot-evtc`). Depicted are participant 1 and 10 from the `Rice` data set.

From the plots, we can get a qualitative indication of the prediction performance change for a fixed set of events of interest, while applying different sizes of the preceding event set. For instance, for `CoSphere` participant 1, the events of interest set with `oc-evti = 10` shows that a large preceding events set with `oc-evtc = 5` provides the best prediction performance. The performance decreases with higher `oc-evtc` values until 50, from which on there is an upward trend again.

When using the plots to get an indication of changing normalized prediction performance for a fixed preceding events set, we must realize that different event of interest sets have different measurement point distributions. Sets with only highly frequent events are much less likely to see measurement points long before the event of interest occurs, which may impact the prediction performance.

#### 5.2.4. Conclusions on variable occurrence thresholds

Considering the plots of all participants, we conclude the following. Under cross validation, there is a clear trend for the CoSphere data set that using a larger number of predictors, also those predictors that occur infrequently, results in a better prediction performance given a fixed set of events of interest. The effect of a possible lack of predictor stability does not seem to play an important role: in most cases, events that occur infrequently are also useful to take into account. For the Rice data set, however, this trend is less strong, but can still be observed for most participants.

#### 5.3. Influence of multi-network interface information

The CoSphere data set consists of cellular, 802.11 and Bluetooth events, but so far we selected events without differentiating on type of technology. We are interested in finding the influence of using different types of network events on the prediction performance. Knowing this influence helps to make decisions on which network interfaces to use to collect network events: if we want to predict Bluetooth events and the availability of 802.11 wireless LAN events do not positively contribute to a better prediction, then it is not wise to invest resources to gather these 802.11 events.

We again turn to cross validation to determine the prediction performance of a number of events of interest sets. In this case, we use a set of events of interest for every participant with a minimum occurrence threshold of 10, and filter this set based on the type(s) of network events we want to predict. We choose sets of preceding events in the same way, based on the network events we want to use as predictors. The rest of the prediction calculations are executed in the same way as the strategy comparison calculations.

##### 5.3.1. Cross-technology results

The results of these calculations on the participant data in the CoSphere data set are presented in Table 2. The events of interest in four different sets (evti sets) are predicted using the best11 strategy, where the prediction performance is indicated by the normalized log likelihood taken over all measurement points for a participant. Each type of network technology has its own character: ‘c’ indicates cellular events, ‘w’ are 802.11 wireless LAN events and ‘b’ are Bluetooth events. A set marked as ‘cwb’ therefore holds all three types of network events, while a set marked as ‘b’ holds only Bluetooth events.

Table 2: Aggregated prediction performance for event sets using different types of network technology events (CoSphere data set).

evti	cwb							c						
evtc	cwb	cw	cb	wb	c	w	b	cwb	cw	cb	wb	c	w	b
11	--	-0.08	-0.12	-0.24	-0.29	-0.41	-0.54	--	-0.06	-0.06	-0.35	-0.18	-0.63	-0.56

evti	w							b						
evtc	cwb	cw	cb	wb	c	w	b	cwb	cw	cb	wb	c	w	b
11	--	-0.05	-0.25	-0.11	-0.44	-0.18	-0.51	--	-0.15	-0.12	-0.15	-0.37	-0.54	-0.33

The seven sets of preceding events (evtc sets), used as predictor sets, constitute all possible combinations of network technology event types. The 11 row shows the aggregated results over all participants. An entry with dashes is the case that – on average – delivers the best prediction results for a particular evti set. The other results in that row for the same evti set indicate the average log likelihood difference taken over all participants between that entry and the ‘best result’ entry. As such, this weighted difference indicates how much worse that entry predicts compared to the best entry: the more negative this result, the worse the prediction. Note that individual participants may show a different preference order than the average preference order.

In all four cases, the average prediction results are best when using all types of network events. Therefore, when interested in the best performance, all network type events should be used to predict well in general. In most cases, however, leaving out Bluetooth events lowers the performance only marginally. And in individual cases, leaving out certain types of network events increases the prediction performance (such as not using Bluetooth when predicting cellular events only).

The prediction of events for individual network technologies benefits most from using preceding events of the same technology type. This is clearly the case for the cellular and 802.11 network technologies, where not having cellular respectively 802.11 preceding events reduces the performance significantly. This effect is less strong for Bluetooth, however: in general, using cellular only events is comparable with using Bluetooth only events as preceding events. For quite a number of individual participants, the cellular events predict Bluetooth events better than Bluetooth events themselves. We do not have a good explanation for this effect.

### 5.3.2. Conclusions for cross-technology prediction

Given these results, we conclude the following. In general, using more types of network events increases the prediction performance of sets combining events from arbitrary technology types. In individual situations however, this is not always the case. In general, events from the same technology contribute most to increasing the prediction performance. Again, this is not always true for individual cases. So, when deciding to collect or not collect events of a certain network type, there are a number of rules of thumb to apply upfront, although it is understood that this is not always best for individual cases. For a better indication of the influence of events of different network types, it is necessary to measure performance at the individual participant level.

### 5.4. Prediction performance evolution with increasing stream size

We now present the performance evolution results for the CoSphere data set, using the strategies we also compared previously. The evolution comparison differs from the cross validation comparison, in terms of the used algorithm. We use the same scheme for determining measurement points (i.e., backward increasing distance, exponentially distribution). The best smoothing parameters, however, are not calculated on the data of all other intervals except the current interval, but instead are determined using the data of preceding intervals only. When going over the measurement points for a participant, this means that the prediction calculation at measurement points later in time are based on more historical data than those earlier in time. It is interesting to see whether the strategy preference observed for the cross validation comparison holds for the case where we have a growing data set – the case that better reflects real scenarios.

The performance at a certain point in time is calculated over the preceding measurement points that fall into a sliding window. This is necessary, because the performance at individual measurement points may fluctuate considerably, making it hard to compare strategies visually in a plot. We set the window size to 1/3 of the total time a participant had his data collected, and average the results for the measurement points in that window (weighted performance). For the first part of the stream, this scheme averages over few measurement points, which makes that part less suitable for comparison. Note that participants have been in the trial with different duration, with an average duration of almost 34 days.

#### 5.4.1. Evolution results

We use the likelihood metric now to compare different strategies. The results from the cross validation comparison show that the `bestll`, `lastcondr`, `lowestvar`, and `lastcondp` strategies deliver the best predictors: we therefore focus on these strategies here. The prediction performance evolution for these strategies for participant 1 and 12 of the CoSphere data set is shown in Figure 15, with an occurrence threshold value of 10. We observe that, after initial fluctuation when the sliding window is still small, the weighted log likelihood is best for the `bestll` strategy for all participants.

In order to compare the evolution results with the cross validation results, we took the likelihood values for a fixed set of events of interest with an occurrence threshold of 50 and varied the preceding events set size. The trend observed for the evolution values is the same as the cross validation values: in general, the prediction performance increases with a lower threshold value. Also, individual trends are the same for the evolution and cross validation cases.

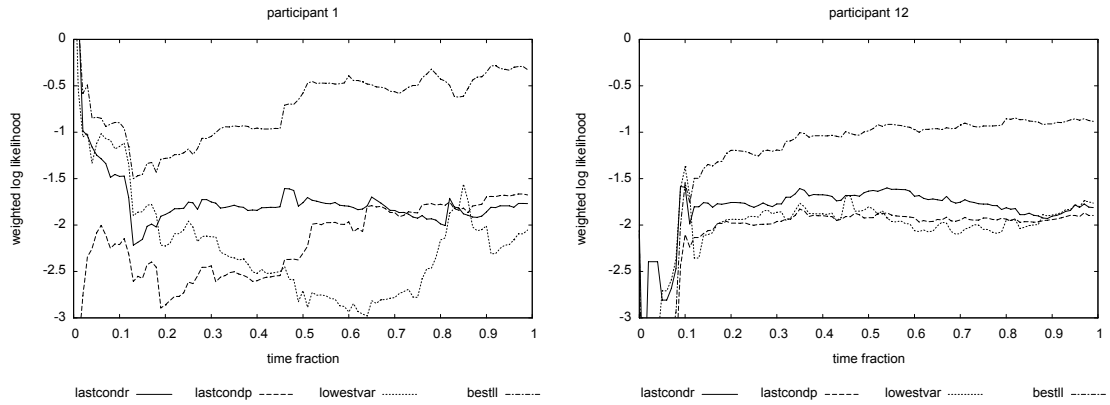


Figure 15: Evolution of strategy performance for participant 1 and 12 of the CoSphere data set.

#### 5.4.2. Prediction performance evolution conclusions

For all participants, the `bestll` strategy is quickly the one with the highest weighted log likelihood. This brings the evolution strategy comparison in line with the cross validation comparison. When doing the same calculation for different occurrence thresholds (not depicted here), the results are also in line with the cross validation results: for higher threshold values, the `lastcondr` strategy is often better. Also, we noticed that with higher threshold values, the likelihood for the different strategies is very close to each other.

## 6. Computational aspects

We now provide an analysis of the usage of computational resources of the model selection strategies introduced in Section 3, including the usage of the basic kernel density algorithms, and of the calculation of the prediction performance as discussed in Section 5. Additionally, we provide an overview of the computing costs of the smoothing parameter calculations and the execution of selection strategies for the CoSphere data set. The elementary calculation considered here is a single computation of the kernel function  $K$ .

### 6.1. Analysis of computational cost of kernel (conditional) density estimation and predictor selection

The straightforward calculation of the best smoothing parameters for KDE and KCDE has the following time cost. With  $n$  the number of observations in the data set and assuming  $m$  different smoothing parameter values, exhaustively determining the optimal value(s) requires  $O(mn^2)$  kernel calculations for KDE and  $O(m^2n^2)$  kernel calculations for KCDE. This polynomial time means that with an increase of the number of observations in a data set or with an increase of parameter granularity, the cost of computation is going up considerably. Note, that the number of observations in a data set is not the same as the total number of events that have occurred in the stream: a data set is extracted from the stream to hold all information relevant for a model, such as a set of durations for KDE or a set of {conditional event, event of interest} pairs for KCDE. So, even when the stream grows large, the size of some data sets may still be limited. In case a data set is large, however, previous research on optimizing kernel density calculations fortunately has resulted in more efficient approaches. Gray and Moore [10] show several orders of magnitude improvement for KDE and the approach in [12] provides substantial improvements for KCDE. Also, the process of finding the best smoothing parameter(s) may be substantially accelerated with limited error when using a *gradient descent* method instead of an exhaustive method. This method consists of choosing a starting value in the one or two dimensional parameter space and then making a next step to a nearby parameter value that has a lower associated cross-validated likelihood – until a local minimum is reached. For the on-line re-calculation of the smoothing parameter(s), we can select the best parameter value(s) of the previous calculation as a starting point for the gradient descent algorithm. In the case of converging parameter values, the old values are

close to the new values, so that the gradient descent method very quickly finds the new optimal values. Given these possibilities for optimizing kernel density calculations, it is reasonable to assume that also for larger data sets, the computational load of the KDE and KCDE parts in our approach will be within practical limits.

Now, depending on the choice of the model selection strategy, the number of different kinds of models (duration based or periodicity based), and the event of interest, it is necessary to repeat the best smoothing parameter calculation. Assuming a scenario where the event of interest is predicted by selecting, in the worst case, between all  $l$  other events as conditional events, with conditional events expressed as values on a chosen periodicity of the stream, a total of  $l$  KCDE best smoothing parameter calculations must be performed. In case we do not know the event of interest up front, and we want to predict – worst case – any of the  $l$  events for this scenario, a total of  $l^2$  calculations must be performed. This is showing that the computational cost may depend considerably on the number of defined events.

The re-calculation of the best smoothing parameter(s) is only relevant when new observations are added to the data set, i.e., when the stream grows, and the event of interest has reoccurred. In the time between two occurrences of the event of interest, the best smoothing parameters do not change and therefore do not need to be calculated every time we are interested in a prediction during this interval. Also, in case of highly converged densities with larger data sets, one or a few new observations of the event of interest will only marginally change the best smoothing parameter values. Therefore, the best smoothing parameter calculations are suitable for executing off line, on a periodic time scale.

The best predictor selection process has the following cost. For every preceding event available at the time a prediction is needed, the waiting time density must be scaled to obtain the residual waiting time, in accordance with Equation 2. This integration over the waiting time density is in its most straightforward form a succession of density calculations, using the pre-calculated best smoothing parameters. Every evaluation of the density function requires  $n$  kernel calculations for KDE and  $2n$  for KCDE (Equation 4 and Equation 6). When the density function is smooth, a high integration accuracy is reached with relatively few density calculations, so we choose the integration delta depending on the level of smoothing by setting it the  $h / 10$  for KDE and  $h_1 / 10$  for KCDE. When integrating to an infinite boundary, as is the case for Equation 2, we must find a suitable cutoff point after which no further points need to be calculated because their values are very close to zero. So, the number of density calculations depends on the smoothing parameter and the size of the smoothing parameter is usually smaller with larger  $n$ , which would result in a higher number of density calculations for a larger  $n$ . There is, however, an upper bound on the increase of density calculations due to a smaller smoothing parameter, because the smoothing parameter has a lower and upper bound. So, even though the number of density calculations depends on the smoothing parameter, the integral can be calculated in linear time.

When using the lowest variance strategy for best predictor selection, the variance of the residual waiting time density must be calculated. This involves determining the mean of the residual waiting time with a cost similar to integrating, followed by the calculation of the variance using this mean, also with a cost similar to integrating. The optimizations for the calculation of the integral – adapt the number of density calculation using the smoothing parameter and select a suitable cutoff point at the infinite boundaries – can also be applied to calculate the variance.

We conclude that calculating the optimal smoothing parameters are most costly in terms of time – compared to the other operations of the various strategies. Applying the gradient descent optimization algorithms can speed up this calculation considerably, although it still is of  $O(n^2)$  complexity. This is the reason that we can only use the method described in this paper on data streams that have relatively sparse event occurrences.

## 6.2. Prediction performance calculation aspects

A number of causes make the calculation of the cross validated prediction performance costly in terms of computing time and memory. As we showed above, the calculation of the smoothing parameters requires polynomial time with respect to the number of event pairs. Therefore, frequently occurring events of interest in combination with frequently occurring preceding events require substantial more computing time than less frequently occurring events.

Another cause of a substantial computing time is inherent to the operation of cross validation: for every interval of an event of interest, it is often necessary to recalculate the optimal smoothing parameters, especially for the most expensive, frequently occurring preceding events. Furthermore, the

calculation of the basic performance properties is expensive because it requires multiple integration actions per predictor at every measurement point. For the `Rice` data set, we therefore randomly select 100 events of interest from those events that have occurred at least 5 times in the stream, and use this limited initial set throughout the calculations in Section 5 (i.e., apply the threshold values on this limited set). In terms of memory usage, the most demanding parts are the caching of intermediary results such as the cross validated smoothing parameters and the basic performance properties.

To keep the calculation time as well as the memory usage within practical limits, we applied a number of optimizations. The strategy comparison is executed within a Python runtime environment [20], which is relatively slow compared to compiled code. By executing some of the core KDE and KCDE algorithms as native code, we realized a substantial speed increase. On the algorithmic side, we calculate smoothing parameters with a gradient descent optimization, which is much faster than the full algorithm. Additionally, by taking into account that the density estimates are constructed with Gaussian kernels, we were able to simplify the integration steps necessary for scaling the waiting time to the remaining waiting time and necessary for determining the expected remaining waiting time, resulting in faster calculation. Finally, we stored intermediary results in compressed form so that less disk space was required.

The above optimizations do not alter the selection of the optimal smoothing parameters, except for the gradient descent algorithm. We have executed a comparison of the gradient descent algorithm with one start point (‘regular gradient descent’), the gradient descent algorithm with multiple starting points (‘weighted gradient descent’) and the full algorithm. This comparison has used a random selection of over 4000 event pairs covering all 11 participants in the `CoSphere` data set. For KDE, this resulted in different smoothing parameters in 11.5% of the cases for the regular gradient descent algorithms compared to the full algorithm. For KCDE, this was 7.2%. For the KCDE weighted gradient descent algorithm this difference was only 0.1%. Additionally, we have executed the strategy comparison with a full KDE algorithm and a weighted gradient descent KCDE algorithm for the `CoSphere` data set and found no differences in the overall results. Therefore, we conclude that we can apply the regular gradient descent optimization for the smoothing parameter calculations in Section 5.

### 6.3. *CoSphere* data set calculations for smoothing parameters and predictor selection strategies

We now consider the computing costs for determining the smoothing parameters for the `CoSphere` data set. Additionally, we give an overview of the cost for applying the different predictor selection strategies on this data set.

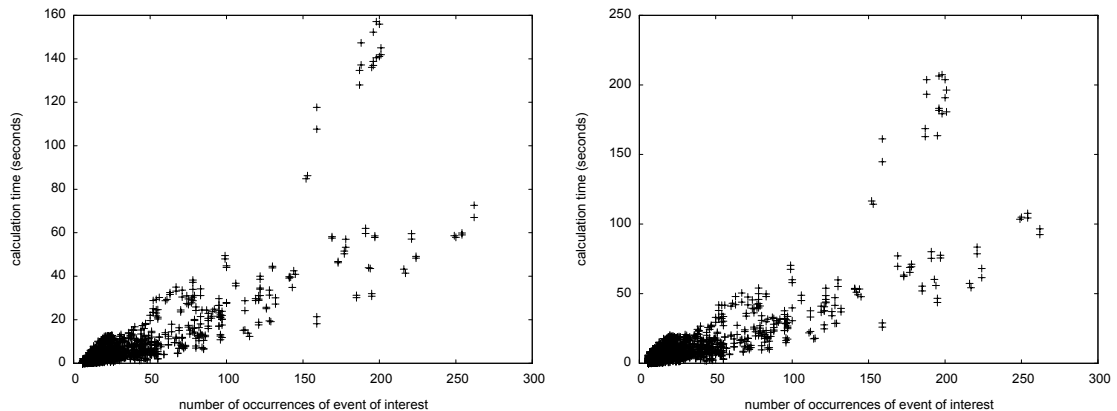


Figure 16: accumulated computing time of the smoothing parameter calculations for all predictors of an event of interest in the `CoSphere` data set, for KDE (left) using the full algorithm, and KCDE (right) using the weighted gradient descent algorithm.

In a typical real-world scenario, the prediction of events of interest takes place while the size of the data set keeps growing, resulting in increasing computing costs over time. We consider here a static situation, using all data in the data set, and calculate the smoothing parameters for all the predictors per events of interest. This provides an indication of the calculation time necessary to predict an event of interest using the full set of preceding events.

We executed the calculations on a laptop with Windows XP and a dual core Intel T5500 CPU with a speed of 1.66 GHz and with 2 GB RAM, in a single thread – using only one of the cores. We used the Python implementation of the KDE and KCDE algorithms as described above (including core KDE/KCDE calculations executed as native code). The KDE calculations were based on the full KDE algorithm and the KCDE calculations were based on the weighted gradient descent algorithm. The smoothing parameter range was [0.1, 10.0] with a step size of 0.1, and the event occurrence threshold was 5. The results are depicted in Figure 16, showing – per event of interest – the accumulated time for the calculation of the smoothing parameters of all predictors for KDE (left) and KCDE (right). The events of all the participants are depicted. As expected, the calculation time increases with a higher number of occurrences of the event of interest. This shows that even for the most frequently occurring events in this data set, predictor smoothing parameters can be determined within 5-6 minutes. Using the full KCDE algorithm, however, increases the KCDE computing time with a factor of approximately 125, and using the gradient descent algorithm reduces the computing time with a factor 10-25 (not depicted). Smoothing parameter memory consumption remains below 3 MByte per participant.

Using the calculated smoothing parameters over the complete CoSphere data set, we determined the cost of predictor selection strategies using a set of 2000 measurement points per participant (as in Section 5). Obviously, the strategies based on last occurrence of the event of interest or based on the last occurred preceding event do not require additional computation. The strategies `eventconcur`, `lowestvar`, and `bestll` do. We found that the accumulated computing time associated with these strategies for a single event of interest is up to 0.5 second for `bestll`, up to 8 seconds for `eventconcur`, and up to 60 seconds for `lowestvar`, so roughly one order of magnitude increase when going from `bestll`, to `eventconcur`, to `lowestvar`.

## 7. Discussion

It is important to note that the above results are based on using two real world data sets. Although collected at different geographical locations, they can not represent what is observed by a very large population. Therefore, we can not assume that these results hold for all possible network event streams collected on personal mobile devices.

We calculated the prediction performance based on event sets with a broad range of events, and have not focused on individual events. It is unlikely that single applications require visibility predictions for all possible events. Also, we used a set of measurement points with a distinct distribution for the time between the measurement points and the to-be-predicted events, and have not investigated the effect of changing this distribution (for instance, by varying  $\lambda$ ). Ultimately, the prediction performance is application specific, because the application determines at which points in time predictions are necessary and which (types of) events are important.

For the results on varying event sets and cross-technology influence, we have used the `bestll` strategy. These results are therefore valid for this strategy only: other prediction methods may obtain different outcomes.

It is clear that the computational costs of our approach will become very high in case of a long stream with many different events. This is not a hypothetical situation, because the gathering of data may take place on a continuous basis, following device owners for many months or even years. We have used a number of optimizations here, but we see many possibilities to reduce the computation time, a few of which we indicate here. A good starting point may be to investigate which historical data is still relevant: the habits of persons change over time so it makes sense to reduce the historical length of the stream. Another is to carefully choose which smoothing parameters to calculate at which moment. Some event pair sets are large and will not influence the smoothing parameters if the set gains a few extra pairs. Other sets have few observations, which require smoothing parameter recalculation for every new pair, but fortunately smoothing parameters based on these small sets are not expensive to calculate.

This paper has focused on network visibility events. Given the increasing number of sensors in mobile devices, such as GPS receivers and accelerometers, the integration of data from all these event sources – including network interfaces – into a single stream may provide additional benefits. Much in the same way as cross-technology information helps to predict events, cross-sensor information may do the same. This can benefit both sides: the network visibility events are predicted better when including other sensor

data, and the sensor data events are predicted better using network visibility information. Overall, we see a great future for the personal mobile device as a versatile sensing platform.

## 8. Related work

A few recent papers have a full or partial focus on the timing aspects in the prediction of network visibility, while a number of other papers are related in the sense that they predict network visibility from another point of view.

The work presented in [15] explicitly focuses on the transient behavior of access point association by using a semi-Markov model. The density estimation is based on a histogram approach which is known to be less accurate than kernel density estimation for many situations. Furthermore, the out-of-range prediction is based on the interval duration of previous access point associations, which is also the case in [16], and does not incorporate a conditional parameter as we do or vary the density estimate with the time of day. This may influence the density estimation accuracy. The approach described in [16] is partially based on parametric density estimation where residence time is fitted to a Weibull distribution. Both papers use the same data set (Dartmouth) collected at the infrastructure side.

The authors in [24] compare a number of prediction algorithms using real-user 802.11 infrastructure traces. The prediction concerns the next access point given a history with a fixed or dynamic length, and includes an n-order Markov model approach. Our approach differs in the sense that our primary interest is in the time aspects of visibility and not the logical order. Also, our dataset is produced on the device side, not in the infrastructure, and therefore gives a more complete view on what is visible from the mobile device perspective, although it is by far not as large as their 802.11 traces. The method presented in [14] focuses on predicting routes between ‘bases’ using GSM cell traces. Paths between routes are clustered together when they are similar. Time aspects such as the duration of routes and length of stay in bases are not incorporated. The connectivity forecasting approach in [17] also uses a Markov model, but focuses, contrary to our approach, only on short term predictions. For a survey of algorithms to forecast the next point of attachment to one or multiple network infrastructure, see [2].

Although strictly speaking the work in [9] does not cover wireless networks, it is concerned with recognizing ‘trajectory patterns’ and explicitly models the time it takes to go from one location to another. It does not, however, take into account the time spent at a location, let alone the probability density of the duration.

## 9. Conclusions

In this paper we have proposed a method for the prediction of events in the dynamic context of a person, as perceived with the sensors available on his personal device. This method supports predicting the next occurrence of events of interest happening on short as well as long term future times. It is based on modeling the sensed context as a stream of events and using the time correlation that exists between successive events. When multiple preceding events are available to act as predictors, we have defined a number of strategies to select the preceding event that is the best predictor.

We have studied the prediction performance of our method by focusing on network visibility events observed using the heterogeneous set of network interfaces available on high-end mobile phones. The input used for generating these prediction results consists of data from our own trial (CoSphere) and the data from another publicly available data set (Rice). Using a cross validation approach on large portions of the data set and an exponentially distributed (backward increasing) set of measurement points to calculate the performance, we show that in most cases the strategy selecting the best predictor based on the highest mean likelihood is the *best strategy*. Furthermore, we found that including *predictors of infrequently visible network entities* results in better predictions using the best strategy (i.e., more data results in better predictions). Also, we found that *cross-technology information in most cases improves the prediction performance* for the CoSphere data set. Or, in other words, to predict the visibility of entities of a single network technology, such as 802.11 access points, it helps to use the visibility of other types of entities such as cellular and Bluetooth entities.

Although we have focused in this paper on forecasting network events, our approach can be applied to a broader set of sensor events generated on mobile devices. For our future work, we intend to use the full

range of sensors to model and predict the surroundings of the user. Furthermore, we seek to investigate emerging applications utilizing the personal mobile device as a sensing platform.

## 10. Acknowledgement

We would like to thank Jan de Gooijer for providing feedback on our prediction approach. Furthermore, we would like to thank Henri ter Hofte and Raymond Otte for their support during the preparation and the execution of the CoSphere trial. The Dutch Freeband Communication Research Programme (Awareness project) supported this research under contract BSIK 03025.

## 11. References

- [1] L. Breiman, W. Meisel, and E. Purcell, Variable Kernel Estimates of Multivariate Densities, *Technometrics*, Vol. 19, No. 2, pp. 135-144, 1977
- [2] C. Cheng, R. Jain, and E. Van den Berg, Location Prediction Algorithms for Mobile Wireless Systems, *Handbook of Wireless Internet* (Ed. B. Furht, and M. Ilyas), CRC Press, 2003
- [3] Y.-S. Chow, S. Geman, and L.-D. Wu, Consistent Cross-Validated Density Estimation, *The Annals of Statistics*, Vol. 11, No. 1, pp. 25-38, 1983
- [4] CoSphere data set, <http://crowdad.cs.dartmouth.edu/novay/cosphere>
- [5] A. Cowling, and P. Hall, On Pseudodata Methods for Removing Boundary Effects in Kernel Density Estimation, *Journal of the Royal Statistical Society. Series B (Methodological)*, Vol. 58, No. 3, pp. 551-563, 1996
- [6] J. De Gooijer, and D. Zerom, On Conditional Density Estimation, *Statistica Neerlandica*, Vol. 57, pp. 159-176, 2003
- [7] V. Epanechnikov, Nonparametric Estimation of a Multidimensional Probability Density, *Theory of Probability and its Applications*, Vol. 14, Issue 1, pp. 153-158, 1969
- [8] V. Estivill-Castro, Why so many clustering algorithms – a position paper, *ACM SIGKDD Explorations Newsletter*, Vol. 4, Issue 1, pp. 65-75, June 2002
- [9] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli, Trajectory Pattern Mining, In *Proceedings of the Conference on Knowledge Discovery and Data Mining (KDD'07)*, San Jose, USA, August 2007
- [10] A. Gray, and A. Moore, Rapid evaluation of multiple density models, In *Proceedings of the 9th International Workshop on Artificial Intelligence and Statistics*, January 2003
- [11] W. Härdle, *Smoothing Techniques, with Implementation in S.*, Springer Series in Statistics, Springer Verlag, 1990
- [12] M. Holmes, A. Gray, and C. Isbell, Fast Nonparametric Conditional Density Estimation, In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'07)*, 2007
- [13] M. Kijima, *Markov Processes for Stochastic Modeling*, Chapman and Hall, 1997
- [14] K. Laasonen, Clustering and Prediction of Mobile User Routes from Cellular Data, In *Proceedings of the Conference on Principles of Data Mining and Knowledge Discovery (PKDD'05)*, October 2005
- [15] J.-K. Lee, and J. Hou, Modeling Steady-state and Transient Behaviors of User Mobility: Formulation, Analysis, and Application, In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'06)*, May 2006
- [16] D. Lelescu, U. Kozat, R. Jain, and M. Balakrishnan, Model T++: An Empirical Joint Space-Time Registration Model, In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'06)*, May 2006
- [17] A. Nicholson and B. Noble, BreadCrumbs: Forecasting Mobile Connectivity, In *Proceedings of the the International Conference on Mobile Computing and Networking (MobiCom'08)*, September 2008
- [18] A. Peddemors, H. Eertink, I. Niemegeers, and M. Bargh, Network Resource Awareness and Control in Mobile Applications, In *IEEE Internet Computing – Special Issue on Roaming*, March/April 2007
- [19] A. Peddemors, H. Eertink, and I. Niemegeers, Density Estimation for Out-of-Range Events on Personal Mobile Devices, In *Proceedings of the 1st ACM SIGMOBILE International Workshop on Mobility Models for Networking Research (MobilityModels'08)*, May 2008
- [20] Python Programming Language, <http://www.python.org/>
- [21] A. Rahmati, and L. Zhong, Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer, In *Proceedings of 5th International Conference on Mobile Systems, Applications, and Services (MobiSys'07)*, June 2007
- [22] D. Scott, *Multivariate Density Estimation: Theory, Practice and Visualization*, Wiley Series in Probability and Statistics, Wiley, 1992
- [23] B. Silverman, *Density Estimation for Statistics and Data Analysis*, Monographs on statistics and applied probability, Chapman and Hall, 1986
- [24] L. Song, D. Kotz, R. Jain and X. He, Evaluating location predictors with extensive Wi-Fi mobility data, In *Proceedings of INFOCOM*, April 2004
- [25] L. Wasserman, *All of Nonparametric Statistics*, Springer texts in statistics, Springer, 2006
- [26] F. Witten and E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques*, 2nd edition, Morgan Kaufman, 2005