Opportunistic Mobile Sensor Data Collection with SCAR

Bence Pásztor, Mirco Musolesi and Cecilia Mascolo
Department of Computer Science, University College London
Gower Street, London, WC1E 6BT, United Kingdom
{b.pasztor|m.musolesi|c.mascolo}@cs.ucl.ac.uk

Abstract

Sensors are now embedded in all sorts of devices (such as phones and PDAs) and attached to many moving things such as robots, vehicles and animals. The collection of data from these mobile sensors presents challenges related to the variability of the topology of the sensor network and the need to limit communication (for energy or bandwidth saving). Fortunately, the data collected, despite considerable, is often delay tolerant and its delivery to the sinks is, in most cases, not time critical.

We have devised SCAR, a context aware opportunistic routing protocol which allows efficient routing of sensor data to sinks, through selection of best paths by prediction over movement patterns and current battery level of nodes. In this paper we present the implementation of the protocol in Contiki and validate the approach through the use of the COOJA simulator with mobility traces provided by the ZebraNet Project. We compare the performance with respect to random choice based dissemination.

1 Introduction

Sensor devices are now starting to be embedded in virtually all sorts of items, from vehicles and furniture to humans and animals. This generates networks of wirelessly connected devices with topologies which could be very dynamic. These devices are used to monitor a very large set of environmental indicators such as temperature, humidity or chemical pollution; innovative applications include body measurements and the analysis of mobility and interactions among individuals or wildlife species. The amounts of data generated are usually quite large, however, fortunately, the data is, in most cases, also *delay tolerant*, in the sense that it can wait in the network for quite a while before being collected.

The scenario we envisage in this paper is one where the mobile sensor nodes (e.g., animals, vehicles or humans) route data through each others in order to reach sink nodes, which can be either mobile or fixed. The fixed nodes are intended as nodes connected to a backbone network and, therefore, able to forward the data to the appropriate place when this is reached. The challenges offered by this scenario are many and include the quantity of data to be shipped to the sinks, the potentially scarce communication power (i.e., energy and bandwidth) of the nodes, the possible communication and sensor hardware faults, the mobility and the limited buffer size of the nodes.

Different techniques could be employed for mobile sensor data gathering. A basic strategy would be to only allow data delivery when sensors are in direct proximity of the sinks. This technique has very little communication overhead, given that messages are only sent directly from the sensor node generating messages to the sink. However, depending on how frequently sensor nodes meet the sinks, the delivery of the data might be very poor. This is particularly true if the sinks are very few and spread out. More refined techniques include epidemically inspired approaches [24], which would randomly spread the data over the sensor network, so that eventually a sink could be reached. These approaches have very good delivery ratio if buffers are sufficiently large, however the overhead in terms of communication and, therefore, energy, is quite high. A major research effort in the direction of opportunistic sensor routing is the ZebraNet Project [12, 16] at Princeton University: researchers used collars with sensors to collect environmental information and to study the movement of animals. Data are replicated using a modified epidemic protocol based on priorities considering the originator of the information. The data created locally are the last to be deleted. In [23], Small and Haas describe another interesting application of epidemic routing protocols to a problem of cost-effective data collection, using whales as message carriers. Fixed buoys are used to collect data that are copied epidemically, stored and spread among the whales. In [25] an approach which is based on a probabilistic delivery approach for data messages is presented. The paper also discusses how the replication of the data over the sensor network can be constrained using a fault tolerance value

1-4244-1455-5/07/$25.00

associated to each data message. However, this approach still has quite a high overhead in terms of message spreading, due to the coarse grained delivery probability technique used for the choice of the nodes on which to replicate and the amount of replication involved by the approach. In sensor networks where energy and, therefore, communication overhead is an issue, the spreading of the message needs to be carefully controlled and traded off for the delivery ratio. This is even more true if the nodes have limited memory so that the buffer size is small and very few messages can be stored.

In this paper we present the design, implementation and evaluation of SCAR (Sensor Context-Aware Routing), a routing approach which uses prediction techniques over context of the sensor node (such as previously encountered neighbours, battery level, etc.) to foresee which of the sensor neighbours are the best carriers for the data messages. This approach is suitable for environments where mobility patterns can be predicted to a certain extent (such as humans and animals). We further adopt different classes of messages in order to achieve an intelligent buffer management.

Multiple carriers are chosen among the neighbours of the data source sensor, based on their history in terms of encounters, mobility and resources, however the number of replicated data around in the network is still considerably smaller than in any epidemic based approaches, in particular than in [25], where the effects of replication may lead to an epidemic-like spreading of the message.

Our prediction framework for choosing a carrier is based on Kalman Filter based forecasting model and has been exploited in [20], where we describe our Context-aware Adaptive Routing (CAR) protocol for mobile ad hoc networks. SCAR has maintained the prediction based approach used in CAR but all the aspects related to the communication and the replication had to be redesigned. In particular, SCAR has to suit the high data traffic of sensor networks. This is achieved by limiting the horizon in which deterministic information is kept to the neighbours of a sensor, and, given the fault rate of a sensor network, we have introduced an intelligent buffer management algorithm and multiple carriers for the message. An initial design of SCAR has been presented in the short paper [18].

In this paper we provide a detailed description of its design, its implementation based on the Contiki operating system [7] and a simulation evaluation with real movement traces in the COOJA simulator [22] to analyse its performance in a large scale scenario. The structure of this paper is as follows: in Section 2 we present our approach, whereas in Section 3 we discuss the details of our implementation of SCAR. In Section 4 we describe the results of our evaluation using the COOJA simulator. In Section 5 we compare our approach with the state of the art. Section 6 concludes the paper, outlining our current research directions.

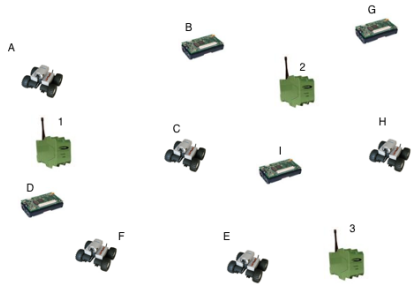

Figure 1. Sensor network composed of sensors (indicated with letters) and sinks (indicated with numbers). Sensors and sinks can be mobile or fixed.

2 SCAR

In this section we discuss the details of SCAR. Our approach can be summarised as follows: the mobile sensor nodes (e.g., attached to animals) try to send their data to sink nodes, scattered over the field (e.g., a forest); each sensor node will try to deliver its data in bundles to a number of neighbouring sensor nodes which seem to be the *best carriers* to reach a sink. The system is depicted in Figure 1.

The decision process by which nodes select the best carriers is based on the prediction of the future evolution of the system in terms of colocation, mobility and battery level. More specifically, our solution relies on the analysis of the history of the movement patterns of the nodes and their colocation with the sinks and on the evaluation of the current available resources of the sensors. Each node evaluates its relative mobility, calculating its change rate of connectivity with other sensors, colocation with sinks, and battery level. The forecasted values of the attributes describing the context are then combined to define a delivery probability $P(s_i)$ of delivering bundles to sinks for each sensor s_i.

While moving, the sensors will transfer their data to other sensors only if these have a higher probability to deliver the data to sinks (i.e., they are better carriers). The calculation of the delivery probability is $local$ and it does not involve any distributed computation. Nodes only periodically exchange information about their current delivery probability and their available buffer space with the neighbours.

We assume that each device of the system is actively involved in the storing-and-forwarding process: usually sensor networks are owned and deployed by a single organisation. For the sake of this work we also assume that nodes have synchronised clocks: this is a reasonable assumptions

as digital clocks are getting cheaper and more precise, however we are working on relaxing this assumption, especially in cases where the data resolution needed is not so small.

We will now go into the details of the protocol.

2.1 Multi-carrier Selection

Each sensor that is the source of some data tries to place bundles on a number of neighbouring nodes which have the best chance to deliver them to a sink node. Each node maintains an ordered list of the neighbouring nodes (including itself) decreasingly ordered according to their delivery probabilities. Each node then replicates the bundle to the first R nodes ($R-1$ nodes if the node itself is in the first R positions of the list). The value of R is application specific and can be considered as a priority level associated to the data retrieved by the sensor.

The replica sent to the node with the highest delivery probability is labelled as *master copy*. The other replicas are labelled as *backup copies*. These can be overwritten if buffers are full, whereas master copies are deleted only when sensors exchange the data with the sinks. In general, this distinction is used for an intelligent management of the buffer, that we will describe in Section 2.3. A unique identifier is also associated to each bundle. Replicas of the same bundle have the same identifier.

Each node keeps monitoring for neighbours with better probability of delivery than its own. If these exist, the data bundles are transferred from one buffer to the other. This, however, implies that the bundles are only *replicated* on a number of nodes in the first hop, while they are *forwarded* (i.e., deleted from one node and copied on another), later if the carriers, while roaming, find either a sink or a better carrier.

As this is a sensor network, the high level of faults in the nodes implies that some replication on the data needs to be allowed. However, if the amount of data generated by the sensors is considerable, the approach of replication adopted by epidemic-like protocols incurs in heavy overheads. SCAR replicates less but tries to control the replication in an intelligent way by predicting the future evolution of the system.

As it will be explained in the following section, the delivery probability of the nodes also keeps into account the energy level of the nodes, so to avoid that some best carriers become strong attractors and run into low battery problems more quickly than others. In other words, we will show that as the battery level decreases, the probability of being selected decreases.

2.2 Choice of Best Carriers

In order to select the best carrier(s) for the data bundles, we use a mechanisms based on the estimation of the future behaviour of each sensor node which depends on the history of its colocation with sinks, its changing rate of connectivity (i.e., its mobility), and its power level[1]. The prediction algorithm is completely local: each node collects the context information, analyses the data and predicts the future evolution of the context indicators.

2.2.1 Forecasting techniques for probabilistic routing

Each node predicts, using time series forecasting techniques, the evolution of its *context* described by a set of attributes. In particular, we consider three indicators describing its colocation with the sinks, its change degree of connectivity and its battery level.

More specifically, a utility function is associated to each context indicator. Our aim is to maximise each attribute, in other words, to choose the node that presents the best tradeoff between the attributes representing the relevant aspects of the system for the optimisation of the bundle delivery process. Analytically, considering k attributes with associated utility functions $U_1(s_i), ..., U_k(s_i)$, the problem can be reformulated as a multiple criteria decision problem [14] with k goals:

$$Maximise\{U(s_i)\} = f(U_1(s_i), ..., U_k(s_i)) \quad (1)$$

The combined goal function, using the the so-called *Weights method*, can be defined as

$$Maximise\{\sum_{j=1}^{n} w_j U_j(s_i)\} \quad (2)$$

where $w_1, w_2, ...w_k$ are *significance weights* reflecting the relative importance of each goal. In our case, the solution is very simple, since it consists in the evaluation of the function $f(U_1, ..., U_k)$ using the values predicted for each node and in the selection of the node(s) i with the maximum of such values.

The overall utility function $U(s_i)$ gives a measure of the probability that a node s_i is able to deliver bundles to any sink. The delivery probability of each sensor will be equal to its composed utility function. More formally, the delivery probability of a sensor s_i is defined as

$$P(s_i) = U(s_i) \quad (3)$$

This utility function is therefore computed by considering its relative mobility (calculated by evaluating its change

[1]Obviously the choice of these parameter is application specific: for wildlife monitoring applications (our main target), battery power is a vital parameter, while this is not the case for vehicular applications. If SCAR was to applied to vehicular applications this parameter should not be used.

degree of connectivity history), its colocation with sinks, and its survivability (calculated by considering its battery level history)[2]. We associate a utility function to each of these indicators, respectively $U_{cdc}(s_i)$, $U_{coloc}(s_i)$ and $U_{battery}(s_i)$), and we compose these utility functions using a weighted sum as follows:

$$U(s_i) = w_{cdc}\widehat{U}_{cdc}(s_i) + w_{coloc}\widehat{U}_{coloc}(s_i) + w_{bat}\widehat{U}_{bat}(s_i) \quad (4)$$

where

- $\widehat{U}_{cdc}(s_i)$ measures the change degree of connectivity of the node i that we define as the number of connections and disconnections that a node has experienced over the last period of $[t-1, t]$ seconds normalised by considering the nodes that have been in reach in this period. This parameter measures relative mobility and, consequently, the probability that a node will meet different nodes in a given period of time, that is the aspect that we are interested in. In fact, being in reach of a large number of different nodes increases the probability of meeting sensors with higher delivery probability or sinks. On the other hand, it may be possible to have a node that moves around but always together with the same nodes; in this case, the node is always colocated with the same devices. Even if its physical mobility is high, its topological mobility (i.e., considering its abstract connectivity graph) is equal to 0.

  More precisely, let $N_{i_{t-1}}$ the set of the neighbours of the node h at time t, the input value to the predictor at time t for $\widehat{U}_{cdc}(s_i)$ is equal to:

$$U_{cdc}(s_i) = \frac{|N_{i_{t-1}} \cup N_{i_t}| - |N_{i_{t-1}} \cap N_{i_t}|}{|N_{i_{t-1}} \cup N_{i_t}|} \quad (5)$$

  where N_{i_t} is the number of nodes in reach of the sensor s_i at time t. Intuitively, this corresponds to the number of nodes that have transitioned from the in reach to out of reach status or vice versa in the time interval $[t-1, t]$, normalised by dividing it for the total number of nodes met in the same time interval.

- $\widehat{U}_{coloc}(s_i)$ summarises the history of colocation of the sensor s_i with a sink. Therefore, the value of $\widehat{U}_{coloc}(s_i)$ is high if a node has been close to a sink. The input of this filter is equal to $\frac{1}{d}$ with d number of hops from the closest sink. Doing this, we obtain a decreasing gradient of this value as the distance from a sink increases. If a path does not exist between the sink and the host, the input is set to 0. This information is extracted from the routing table (a special out-of-range utility value is assigned to sinks).

- $\widehat{U}_{bat}(s_i)$ gives an estimation of the future battery level of the node. The value 1 corresponds to a full battery, whereas 0 corresponds to an empty one[3].

The relative importance of these utility functions is defined by using the weights w_{cdc}, w_{coloc} and w_{bat}. Weights are used to assign different importance to the different dimensions of the sensor context. For example, if the battery level is a critical dimension (that is often the case in wireless sensor networks, except for devices embedded in cars, planes or trains), a high value should be assigned to w_{bat}.

It is important to note that these utility functions represent an estimation of the future trend of these indicators calculated by exploiting time series analysis and forecasting techniques and not the current values of these utility functions. We use the symbol $\widehat{\ }$ to indicate the fact that these are predicted values and not current ones. The forecasted values are calculated by exploiting Kalman filter prediction techniques [13] that were originally developed in automatic control systems theory. These are essentially a method of discrete signal processing that provides optimal estimates of the current state of a dynamic system described by a *state vector*. The state is updated using periodic observations of the system, if available, using a set of *prediction recursive equations*.

One of the main advantages of the Kalman filter is that it does not require the storage of the entire past history of the system, making it suitable for a sensor network setting in which computational and memory resources are very limited. Moreover, this technique is also very lightweight from a computational point of view, since the forecasting model only requires the update of the values representing the state using a system composed of linear equations (without any integration or differentiation required). We present a brief summary of the mathematical aspects of the application of state space models theory and Kalman filter time series analysis to our problem in the appendix of this paper. The interested reader can find more details in [20].

2.3 Buffer Management

2.3.1 Bundle Priorities

As discussed, a replica of a bundle can be a master or backup copy. When two nodes exchange their delivery probability, they also send the number of available slots in their buffer. We assume that the size of the buffer slots is

[2]Even if we take into consideration only these three context indicators, our framework allows for the integration of other utility functions describing other aspects of the system that may be important to improve the performance of the storing-and-forwarding strategy.

[3]We assume mobility is not a source of energy consumption, i.e., that nodes move using other sources of energy (like in the case of animals carrying the sensors in wildlife monitoring applications).

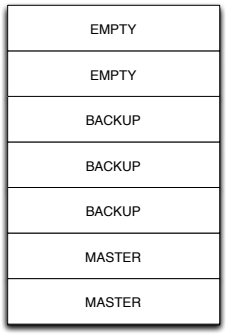

Figure 2. Buffer Management: the figure shows a buffer with a size equal to 7, with 2 master copies in it. In this case, the node will advertise 5 available slots.

fixed[4]. A slot is considered available, if it does not contain a bundle or if its content can be overwritten (i.e., the slot contains a backup copy). For example, in Figure 2, a buffer composed of 7 slots is represented. The buffer contains three backup copies and two slots are empty. The sensor will then advertise 5 available slots.

Bundles are copied in the buffer of the other sensors firstly using the available empty slots and then overwriting the slots containing backup copies. Finally, we would like to discuss an interesting limit case. It may happen that a buffer is full and contains only master copies. In this case, the sensor will not accept any bundle from the other nodes[5]. However, if the node has been selected to carry so many master copies, as its probability of being in reach of a sink is very high, it is likely that the sensor will quickly reach a sink and will transfer all the bundles and then free all the slots in its buffer.

2.3.2 Bundle Deletion Mechanisms

When a sensor meets a sink, the latter sends a hash table containing the identifiers of the bundles to the former. The sensor deletes all the bundles that have already been delivered from its buffer and then sends all the bundles that have not been delivered yet to the sink.

When two mobile sinks get in reach, they exchange these hash tables. Then, each sink updates its hash table adding the identifiers of the bundles delivered to the other sink and not already present in it. A timestamp is associated to each entry of the tables and the older ones are periodically removed.

[4]For simplicity, we also assume here that all the bundles have the same size. However, this mechanism can be easily extended in order to considered bundles of variable size, such as bundles that require two buffer slots and so on.

[5]When the number of slots is equal to 0, sensors will not advertise their deliver probability, in order to avoid a waste of energy, since this action will be completely useless.

2.4 Exchange of Context Information

Neighbours exchange their own delivery probabilities. Each node maintains an ordered *delivery probability list*. Each entry of this table has the structure *(sensorId, deliveryProb, availableSlots)*; where *sensorId* is the sensor identifier, *deliveryProb* is its current delivery probability and the last field is the number of available slots defined as discussed in Section 2.3.

Periodically, each sensor sends its delivery probability to their neighbours together with the number of the available slots.

2.5 Replication Process

As said before, each sensor keeps monitoring if neighbours with better probability of delivery than its own are in reach. This is done by examining the context information received by the other nodes. If there is a node in proximity with a higher delivery probability, the bundles are transferred to that node.

A bundle is copied from a sensor s_A to a sensor s_B if and only if the probability of s_A is a lot larger than the probability of s_B:

$$P(s_B) \gg P(s_A) \tag{6}$$

This is evaluated by setting an *exchange threshold* ζ. Therefore, the replication process between s_A and s_B happens if and only if

$$P(s_B) - P(s_A) > \zeta \tag{7}$$

This prevents replication actions that are not characterised by a good trade-off between delivery probability and energy consumption. Moreover, it avoids possible bundle thrashing, that may cause considerable waste of energy.

Finally, if the buffers of the other nodes are full, not all the bundles can be transferred. In this case, priority is given to the master copies. If there is not enough space for all the master copies, these are selected for replication randomly[6]. The same happens for the backup copies.

2.6 Emergency Replication

An additional mechanism is introduced in order to cope with situations where nodes carrying master copies exhaust their battery. When the battery level is low (i.e., under a certain threshold), the master copies of the bundles are copied

[6]Alternatively, a priority may be associated to each bundle and used for this selection process. The number of initial replicas can also be used as priority. In this paper, we assume that all the data sources have the same importance (i.e., priority).

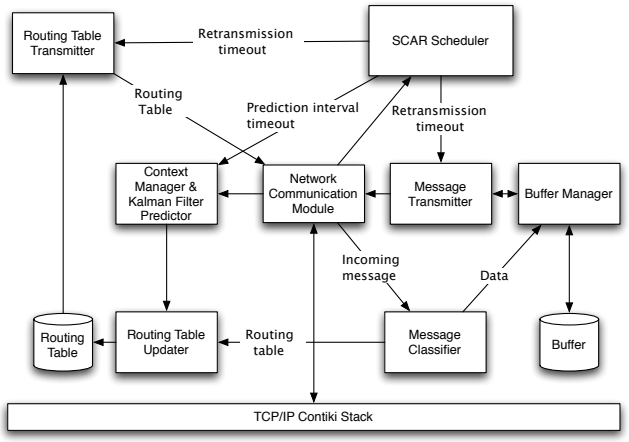

Figure 3. Architecture of the SCAR implementation on Contiki.

to the nodes in reach that have a sufficient number of free slots without considering the current values of their delivery probabilities.

In general, the fact that the battery level is taken into consideration in the calculation of the delivery probability should be sufficient to avoid these situations. However, it may happen that nodes with low battery level store master copies because of a particular combination of weights that gives a low relative importance to battery level and/or high values of the colocation and change degree of connectivity attributes. For this reason, this mechanism is introduced to increase the fault tolerance of the system and it will be used only in "emergency" situations.

2.7 Predictability of the Sensor Network Scenario

Our system relies on predictions about the future values of context attributes. However, in some conditions predictions are not reliable, e.g., because the prediction model used in the forecasting framework does not provide sufficient accuracy.

To assess the quality of context predictions it is possible to use the technique presented in [21], based on the analysis of the forecasting error [3]. The analysis of the predictability of the time series can be performed periodically in order to save resources. However, it is worth noting that this technique is lightweight from a computational point of view.

When the predictability component determines that predictions are unreliable, we will use alternative protocols to carry the data, for example epidemic-style approaches.

3 Implementation Details

In this section we describe the implementation of SCAR on top of the Tmote Sky nodes [19], equipped with a MSP430 microprocessor, a cc2420 Chipcon radio chipset [4] and a number of sensors (i.e., temperature, light and humidity).

The nodes are running the Contiki Operating System [7]. Contiki is an open source, highly portable, multi-tasking operating system for memory-constrained networked embedded systems. Contiki consists of an event-driven kernel on top of which application programs are dynamically loaded and unloaded at runtime. Contiki processes use *protothreads* [8] that provide a thread-like programming abstraction on top of the event-driven kernel.

SCAR is implemented as a Contiki process loaded with the kernel. The routing protocol itself consists of a protothread called *SCAR process*, waiting for different events: incoming data or timer-notifications. A timer is used to time the periodic transmission of routing beacons containing the current delivery probability and buffer size of the node. Another timer is used to send data messages periodically. Each message is uniquely identified by the host name and a message number generated using a counter that is incremented by one for each message sent. Whenever these timers expire, or there is an incoming packet, an event is fired. Each event is caught in the main loop of the process, and dealt with (when the routing timer expires, a routing message is sent). The SCAR process is also waiting for incoming data, which is processed and stored depending on the message type. If it is a routing message, then the information is extracted and the corresponding routing table entry is updated. If it is a data message, then an ACK is sent and the message is stored in the buffer. Messages are not deleted from the buffer unless they are acknowledged or the buffer becomes full as described above. The architecture of the SCAR implementation on Contiki is shown in Figure 3.

Each node has a routing table with entries holding information (i.e., buffer size, availability, delivery probability) about other, neighbouring nodes. When a message is to be transmitted, this routing table is considered and the message is forwarded to the node with the best characteristics. If no such neighbour is available, then the message is stored in the buffer, and transmitted later. Both the routing table and the messages are stored as C structures. The routing table is updated when a routing message is received, and also just before the current node sends a routing message. In this way, the node can check whether it has gone out of reach of any of its previous neighbours, or gained a new neighbouring node, and updates its local Kalman predictor.

Kalman filter prediction technique may appear heavyweight from a computational point of view. Instead, since they require only the storage of the current predicted val-

Program 1 Excerpt of the code used for the calculation of the predicted value using Kalman filter forecasting.

```
void getPredictedValue(struct kalmanStruct* kalman, float currentValue, float
initialValue) {
  float residual;
  kalman->currentInputValue=currentValue;
  kalman->pastPredictedValue=kalman->currentPredictedValue;

  if (kalman->counter==1) {
    kalman->currentValue=currentValue;
    kalman->currentPredictedValue=initialValue;
    kalman->currentOmegat=initialOmegat;
    kalman->counter++;
  } else {
    kalman->currentValue=currentValue;
    kalman->currentPredictedValue=kalman->currentPredictedValue+kalman->currentOmegat*
    (kalman->currentValue-kalman->currentPredictedValue)/(kalman->currentOmegat+kalman->rt);
    kalman->currentOmegat=kalman->currentOmegat+kalman->qt-kalman->currentOmegat*kalman->currentOmegat/
    (kalman->currentOmegat+kalman->rt);
    kalman->counter++;
  }
}
```

ues and the state of the filter, they are also suitable for very resource-constrained devices like motes. However, since the motes have a simple, 16-bit processor without floating-point support, floating-point arithmetic had to be added at the software level.

An excerpt of the code used to update the state of one Kalman filter predictor is presented in the box Program 1. It essentially consists of the update of two recursive equations for the calculation of the current state and current predicted value given the observed value of the attribute taken into consideration.

The memory footprint of the implementation of SCAR is 16 KB. The number of lines of code is about 700. The memory required by the operating system is 28KB in the configuration used in our implementation. Additional space is required to store the routing table and the message buffer, but the entire program can be loaded onto a Tmote Sky, still leaving some space for other applications. The size of a routing table entry is 10 bytes. A SCAR message has the following structure:

```
struct message{
int msgType;
uip_ipaddr_t nextHop;
int index;
int msgId;
        uip_ipaddr_t msgSource;
int isBackup;
float delProb;
struct data msg[7];
};
```

The message type (`msgType`) describes the type of the message (routing, data or ack message). `nextHop` indicates the next recipient for the message while `index` is the hopcount for the message. `msgId` is the identifier of the message, which is a unique identifier per message source (`msgSource`). `isBackup` indicates if the message is a master copy or a replica, while `delProb` indicates the delivery probability: this is used for routing messages. The message content is in `data`.

Integers are 2 bytes, floats are 4 bytes while `uip_ipaddrs` are 8 bytes. The full header of the message is 26 bytes.

4 Evaluation

In this section we report on the evaluation of SCAR through the use of the COOJA simulator [22] and realistic connectivity traces generated with the Zebranet trace generator [16].

COOJA [22] is a wireless network simulator for the Contiki OS. It is written in Java and uses the Java-Native Interface (JNI) to translate programs written in Contiki into Java code. COOJA provides plug-ins to monitor the nodes, from their power level to the state of their LEDs with the possibility of easily plugging extensions. The standard version of COOJA does not provide support for mobility. We have added this functionality by means of an additional `Positioner` interface. Using this, COOJA can read coordinates from a file and move the nodes accordingly. With an additional `RadioInterface`, we enabled COOJA to read connectivity traces, i.e., traces containing information about the topology of the network in terms of connections and disconnections duration among each pair of the nodes of the system.

4.1 Simulation Settings

The simulation settings of COOJA are indicated in Table 1, unless otherwise specified. We have tried to mirror the TMote Sky radio capabilities.

The traces we have used are the ones generated by the Zebranet trace generator written by Yong Wang to generate traces for a different number of nodes with the same distribution of node speed and movement patterns. The script

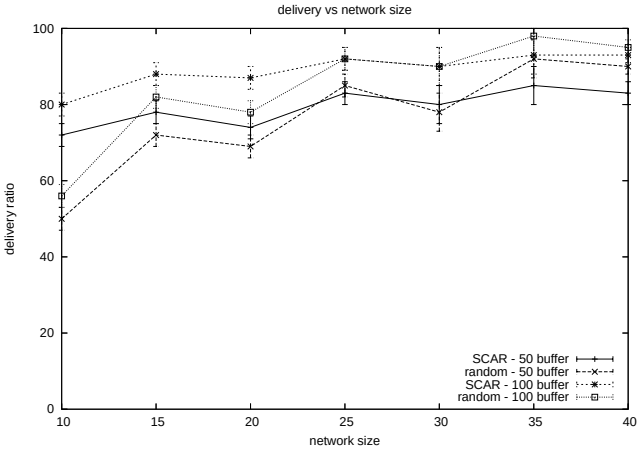

Figure 4. Delivery ratio vs node density.

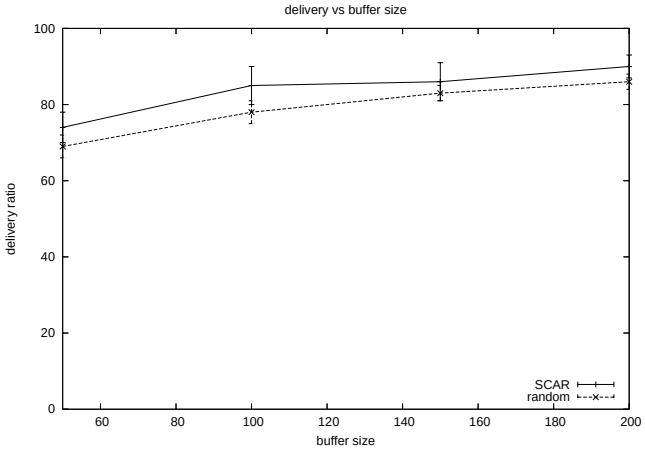

Figure 6. Delivery ratio vs buffer size.

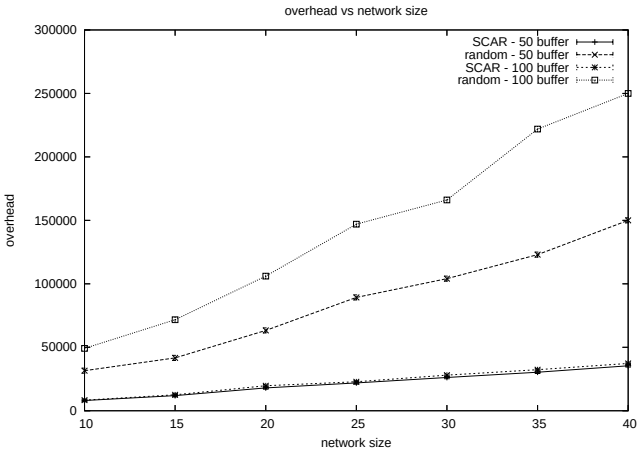

Figure 5. Overhead vs node density.

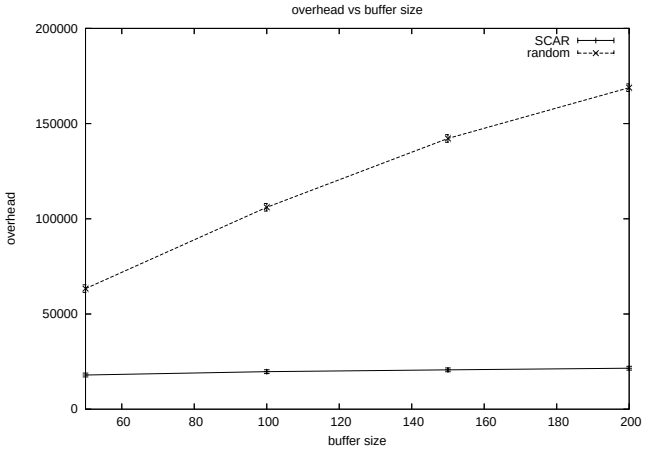

Figure 7. Overhead vs buffer size.

generates random traces based on real data. The positions of the nodes were updated every 8 minutes, routing messages and data messages were sent every 2 and 5 minutes, respectively. The nodes stopped sending after the 100th sent message, and acted as intermediate nodes until the simulation ended. Each simulation lasted for 10 simulated hours; the messages were sent in the first 8 hours approximately (i.e., with a transmission interval equal to 5 minutes).

We compared SCAR with a random choice based protocol which replicates to k neighbours as SCAR does but does not rely on any predictive mechanism to choose the best carriers for the data. This is fair comparison as it allows us to evaluate the accuracy of the prediction based choice. The implementation of this protocol is very similar to SCAR except for the decision-making part, which is random. The only exception is the case when a sink is in reach; then the node will transmit its messages to the sink directly also in the case of the random choice.

4.2 Simulation Results

We now report the results of the simulation. The metrics we have concentrated on are message delivery ratio and message overhead (which is important for estimating sensor energy consumption). In studying the performance we have varied the number of nodes in the scenario, increasing

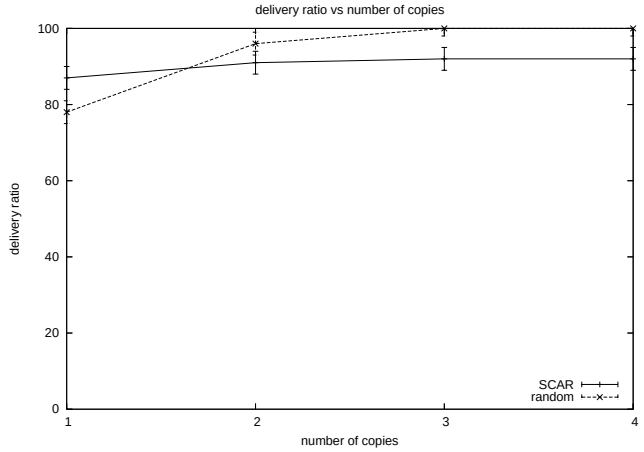

Figure 8. Delivery ratio vs number of copies.

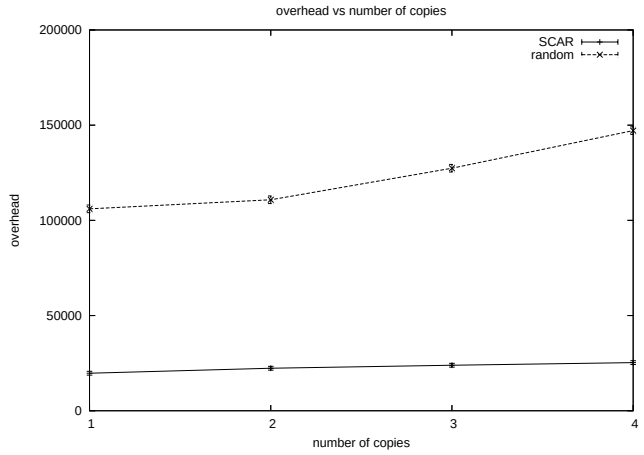

Figure 9. Overhead vs number of copies.

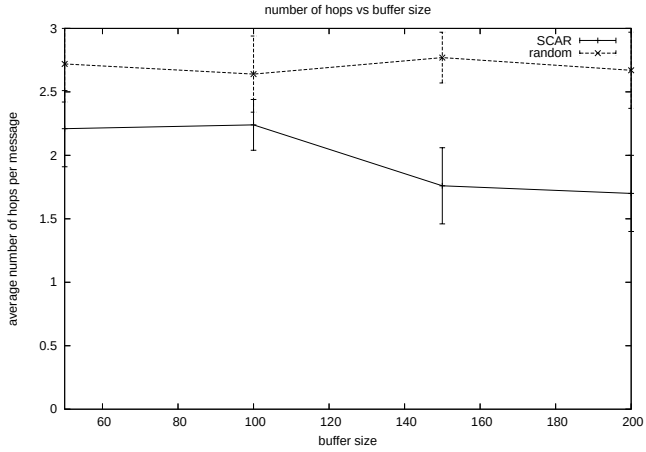

Figure 10. Hops vs buffer size.

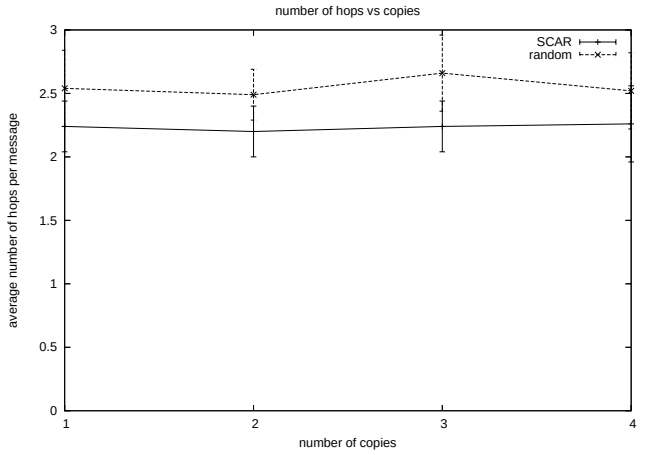

Figure 11. Hops vs number of copies.

density, the buffer size and the number of replicas made by each sensor.

Figure 4 shows the delivery ratio against the number of nodes in the network for different buffer sizes for the two protocols. The associated overhead in terms of number of messages sent is reported in Figure 5. As the buffer size increases, the delivery ratio increases, given the lower probability of losing messages due to memory limitations. In a less dense network the gap in terms of performance between SCAR and random is more evident: in a more connected graph, the random selection protocol has a higher probability of being in contact of the recipient of the message by means of the random forwarding. We observe that our main application scenarios (wildlife monitoring) are usually characterised by very sparse intermittently connected networks. The performance of the two protocols are similar for denser networks, but the overhead associated to the random selection one is much higher. The random protocol is reaching a high overhead due to random walk-like forwarding of the copies. Figures 6 and 7 show the influence of the buffer size in scenario composed of 20 sensors for a larger range of values. With a large buffer, the random protocol has similar performance in this scenario, but the associated overhead is nearly an order of magnitude higher.

The improvement of the delivery ratio using an increasing number of copies can be observed in Figure 8. In these

Area Size	2 km x 2 km
Radio Range	200 m
Transmission Rate	250 kbps
Number of Sensors	10 to 40 (step: 5)
Number of Sinks	10% of Sensors
Buffer Size	50-100-150-200
Number of Pack. Sent (per host)	100
Routing Retrans. Interval	120 s
Message Retrans. Interval	300 s
Number of Replicas	1-2-3-4
Simulation Duration	10 h

Table 1. COOJA settings.

simulations, we consider a 20 hosts scenario; the buffer was set to 100. There is a high price to pay in terms of overhead as shown in Figure 9. The random protocol outperforms SCAR, but its overhead is extremely high in comparison with our protocol from an energy consumption point of view. The saturation of the delivery ratio of SCAR is due to the limited duration of the simulation. SCAR is able to reach 100% delivery ratio also in this sparse scenario with a longer simulation time. We also observe that increasing the number of copies beyond two does not increase the delivery ratio. This is due to the fact that the number of better carriers in terms of delivery probability is usually equal to or lower than two.

To further assess the efficiency of SCAR compared to random routing, we have plotted the average number of hops a message travelled from source to destination against different buffer sizes and varying number of copies. For these experiments, the network size was set to 20. In Figure 10 the number of traversed hops is more or less constant for the random protocol, while it decreases as the buffer size increases in SCAR. This can be explained by the intelligent forwarding of SCAR. As we increase the buffer, a good carrier can take more messages to the sink, thus SCAR uses less nodes to deliver a message on average.

Figure 11 shows the average number of hops considering different number of copies. The buffer size was set to 100. We only considered the first delivered copy in our simulations. The graph shows a constant number of hops for both protocols as the number of copies increases. This can be explained by observing that, in most cases, the master copy is delivered to the sink successfully. Again, the graph shows a higher average number of hops for the random routing protocol than SCAR: this is also due to effectiveness of the prediction based routing mechanisms.

To summarise, we have shown that SCAR performs reasonably well in scenarios characterised by sparse topologies. A higher buffer size and number of copies ensure better performance but with an increased overhead and resource consumption. With respect to the random selection protocol, the simulations demonstrate the effectiveness of the prediction mechanism for the intelligent forwarding of the replicas. This mechanism ensures very high delivery ratio with a considerably smaller overhead.

5 Related Work

There have been a number of attempts of dealing with delay tolerant networks [9] overcoming the limitation of synchronous forwarding. In the area of mobile ad hoc networking, for instance, epidemic routing protocols [24] form the basis for much of the work in this field.

In [23], Small and Haas describe an interesting application of epidemic routing protocols to a problem of cost-effective data collection, using whales as message carriers. In [15], Lindgren et al. propose a probabilistic routing approach to enable asynchronous communication among intermittently connected clouds of nodes. Their approach is based on the fact that the exploited communication model is typically transitive and, for this reason, the probability of message delivery must be calculated accordingly. Zhao et al. in [26] discuss the so-called Message Ferrying approach for message delivery in mobile ad hoc networks. The authors propose a pro-active solution based on the exploitation of highly mobile nodes called ferries. These nodes move according to pre-defined routes, carrying messages between disconnected portions of the network.

In terms of sensor networks a lot of effort has been devolved into data forwarding in static sensor networks [6, 10, 17]. Some attempts have also been done in the direction of more dynamic sensor networks where mobile sinks are available such as [5, 11]. In ZebraNet [12, 16] mobile sensors are deployed for tracking zebras in a hostile and wide environment. This is one of the closest work to ours together with [25].

However, with respect to these works, our data transmission overhead is lower (we do not have epidemic-like dissemination) and, thanks to the prediction techniques used to calculate the probabilities, the delivery of data is still reasonably high. In other words, we believe that our solution provide a better trade-off between the delivery ratio and the energy consumption (i.e., improved sensor survivability).

More recently, the MetroSense project [2] is investigating the possibility of deploying a large scale sensing infrastructure, where delay-tolerant data collection play a key role. SCAR can be seen as a possible protocol to support asynchronous data retrieval in this kind of settings.

6 Conclusions

In this paper we have described SCAR, a protocol for data forwarding on mobile sensor networks towards a num-

ber of fixed or mobile sinks. We have illustrated the details of the SCAR implementation over Tmote Sky nodes, and of the evaluation through the COOJA simulator with realistic mobility traces.

We are investigating real deployment issues in the context of our wildlife monitoring project with tagged animals as sensor nodes.

Acknowledgments We thank EPSRC through projects CREAM and WILDSENSING and the European Union through project RUNES.

References

[1] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting*. Springer, 1996.

[2] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson. People-centric urban sensing. In *Proceedings of 2nd ACM/IEEE International Wireless Internet Conference (WICON 2006)*, August 2-5, 2006, Boston, USA.

[3] C. Chatfield. *The Analysis of Time Series An Introduction*. Chapman and Hall, 2004.

[4] Chipcon. 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Datasheet, 2007. www.chipcon.com/.

[5] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy, and G. P. Picco. TINYLIME: Bridging Mobile and Sensor Networks through Middleware. In *Proceedings of PerCom 2005*, pages 61–72, Kauai Island (Hawaii, USA), Mar. 2005. IEEE Computer Society.

[6] A. Demers, J. Gehrke, R. Rajaraman, N. Trigoni, and Y. Yao. Energy-Efficient Data Management for Sensor Networks. In *Proceedings of the IEEE Upstate NY Workshop on Sensor Networks 2003*, 2003.

[7] A. Dunkels, B. Grnvall, and T. Voigt. Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*, Tampa, Florida, USA, Nov. 2004.

[8] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying event-driven programming of memory-constrained embedded systems. In *Proceedings of SenSys'06*, page 14, Boulder, Colorado, USA, 2006.

[9] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of SIGCOMM'03*, August 2003.

[10] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of ACM/IEEE MOBICOM'00*, pages 56–67, 2000.

[11] D. Jea, A. A. Somasundara, and M. B. Srivastava. Multiple Controlled Mobile Elements (Data Mules) for Data Collection in Sensor Networks. In *Proceedings of DCOSS'05*, pages 244–257, 2005.

[12] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet. *SIGOPS Operating Systems Reviews*, 36(5):96–107, 2002.

[13] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME Journal of Basic Engineering*, March 1960.

[14] R. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preference and Value Tradeoffs*. Wiley, 1976.

[15] A. Lindgren, A. Doria, and O. Schelen. Probabilistic routing in intermittently connected networks. *Mobile Computing and Communications Review*, 7(3), July 03.

[16] T. Liu, C. M. Sadler, P. Zhang, and M. Martonosi. Implementing software on resource-constrained mobile sensors: experiences with Impala and ZebraNet. In *Proceedings of MobiSys'04*, pages 256–269, New York, NY, USA, 2004. ACM Press.

[17] S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TAG: a Tiny Aggregation Tree for ad-hoc sensor networks. In *Proceedings of OSDI'02*, 2002.

[18] C. Mascolo and M. Musolesi. SCAR: Context-aware Adaptive Routing in Delay Tolerant Mobile Sensor Networks. In *Proceedings of the Delay Tolerant Networks Symposium. ACM International Wireless Communications and Mobile Computing Conference (IWCMC) 2006*. ACM Press, June 2006.

[19] Moteiv. Tmote Sky – Ultra low power IEEE 802.15.4 compliant wireless sensor module datasheet, 2007. www.moteiv.com/products/docs/tmote-sky-datasheet.pdf.

[20] M. Musolesi, S. Hailes, and C. Mascolo. Adaptive routing for intermittently connected mobile ad hoc networks. In *Proceedings of IEEE WoWMoM 2005. Taormina, Italy.* IEEE press, June 2005.

[21] M. Musolesi and C. Mascolo. Evaluating context information predictability for autonomic communication. In *Proceedings of ACC'06*, Niagara Falls, NY, June 2006. IEEE Computer Society Press.

[22] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt. Cross-level sensor network simulation with cooja. In *Proceedings of Proceedings of SenseApp 2006*, Tampa, Florida, USA, 2006.

[23] T. Small and Z. J. Haas. The shared wireless infostation model- a new ad hoc networking paradigm (or where is a whale, there is a way). In *Proceedings of MobiHoc'03*, June 2003.

[24] A. Vahdat and D. Becker. Epidemic routing for partially connected ad hoc networks. Technical Report CS-2000-06, Department of Computer Science, Duke University, 2000.

[25] Y. Wang and H. Wu. DFT-MSN: The Delay/Fault-Tolerant Mobile Sensor Network for Pervasive Information Gathering. In *Proceedings of INFOCOM'06*, Barcelona, Spain, April 2006.

[26] W. Zhao, M. Ammar, and E. Zegura. A message ferrying approach for data delivery in sparse mobile ad hoc networks. In *Proceedings of MobiHoc'04*, May 2004.

A SCAR Forecasting Model

In this appendix we present the forecasting model used for the prediction of context information in SCAR.

A state space model for a time series $\mathbf{Y}_t$ consists of two equations. The first one called the *observation equation* is the following

$$\mathbf{Y}_t = G_t\mathbf{X}_t + \mathbf{W}_t \quad t = 1, 2, ...$$

with $\mathbf{W}_t$ defined as[7]

$$\mathbf{W}_t = WN(0, R_t)$$

This equation defines the w-dimensional observation $\{\mathbf{Y}_t\}$ as a linear function of a v-dimensional state variables $\{\mathbf{X}_t\}$ and a noise term. The second one is the *state equation* defined as follows

$$\mathbf{X}_{t+1} = F_t\mathbf{X}_t + \mathbf{V}_t \quad t = 1, 2, ...$$

with $\mathbf{V}_t$ defined as

$$\mathbf{V}_t = WN(0, Q_t)$$

This equation determines the state $\mathbf{X}_{t+1}$ at time $t + 1$ in terms of the previous state $\mathbf{X}_t$ and a noise term. Let w as the dimension of $\mathbf{Y}_t$ and v as the dimension of $\mathbf{X}_t$, $\{G_t\}$ is a sequence of $w \times v$ matrices and $\{F_t\}$ is a sequence of $v \times v$ matrices. We assume that $\{\mathbf{V}_t\}$ is uncorrelated with $\{\mathbf{W}_t\}$, even if a more general form of the state space model allows for correlation between these two variables. Analytically, we can rewrite this condition as follows

$$E(\mathbf{W}_s\mathbf{V}_t^T) = 0 \quad \forall\, s, t$$

We also assume that the initial state $\mathbf{X}_1$ is uncorrelated with all of the noise terms $\{\mathbf{V}_t\}$ and $\{\mathbf{W}_t\}$.

With the notation of $P_t(\mathbf{X})$ we refer to the best linear predictor (in the sense of minimum mean-square error) of $\mathbf{X}$ in terms of $\mathbf{Y}$ at the time t. $P_t(\mathbf{X})$ is defined as follows

$$P_t(\mathbf{X}) \equiv \begin{bmatrix} P_t(X_1) & ... & P_t(X_v) \end{bmatrix}^T$$

where

$$P_t(X_i) \equiv P(X_i|\mathbf{Y}_0, \mathbf{Y}_1, ..., \mathbf{Y}_t)$$

$P(X_i|\mathbf{Y}_0, \mathbf{Y}_1, ..., \mathbf{Y}_t)$ indicates the best predictor of X_i given $\mathbf{Y}_0, ..., \mathbf{Y}_t$. We can also observe that $P_t(\mathbf{X})$ has the following form

$$P_t(\mathbf{X}) = A_0\mathbf{Y}_0 + ... + A_t\mathbf{Y}_t$$

since it is a linear function of $\mathbf{Y}_0, ..., \mathbf{Y}_t$. It is possible to prove [1] for the state space model discussed in the previous section that the one-step predictor

$$\widehat{\mathbf{X}}_t \equiv P_{t-1}(\mathbf{X}_t)$$

and their error covariance matrices

$$\Omega_t = E[(\mathbf{X}_t - \widehat{\mathbf{X}}_t)(\mathbf{X}_t - \widehat{\mathbf{X}}_t)^T]$$

are determined by these initial conditions

$$\widehat{\mathbf{X}}_1 = P(\mathbf{X}_1|\mathbf{Y}_0)$$

$$\Omega_1 = E[(\mathbf{X}_1 - \widehat{\mathbf{X}}_1)(\mathbf{X}_1 - \widehat{\mathbf{X}}_1)^T]$$

and these recursive equations

$$\widehat{\mathbf{X}}_{t+1} = F_t\widehat{\mathbf{X}}_t + \Theta_t\Delta_t^{-1}(\mathbf{Y}_t - G_t\widehat{\mathbf{X}}_t)$$

$$\Omega_{t+1} = F_t\Omega_tF_t^T + Q_t - \Theta_t\Delta_t^{-1}\Theta_t^T$$

where

$$\Delta_t = G_t\Omega_tG_t^T + R_t$$

$$\Theta_t = F_t\Omega_tG_t^T$$

As estimation model, we use a basic state space model composed of the following two scalar equations

$$Y_t = X_t + W_t \quad t = 1, 2, ...$$

with

$$W_t = WN(0, Q_t)$$

and

$$X_{t+1} = X_t + V_t \quad t = 1, 2, ...$$

with

$$V_t = WN(0, R_t)$$

With respect to the Kalman filter prediction, we can consider a mono-dimensional system with

$$G_t = [1]$$

$$F_t = [1]$$

Therefore, we can derive the recursive equations of the Kalman filter for the prediction of the values of this series. Given the previous observed value Y_t and the predicted value at time t, $\widehat{X}_t$, the recursive equation for the determination of the predicted value at time $t + 1$ is

$$\widehat{X}_{t+1} = \widehat{X}_t + \frac{\Omega_t}{\Omega_t + R_t}(Y_t - \widehat{X}_t)$$

with

$$\Omega_{t+1} = \Omega_t + Q_t - \frac{\Theta_t^2}{\Omega_t + R_t}$$

Since in this case

$$\Omega_t = \Theta_t$$

we can also write

$$\Omega_{t+1} = \Omega_t + Q_t - \frac{\Omega_t^2}{\Omega_t + R_t}$$

[7] WN stands for White Noise, a term that derives from telecommunication engineering. A white noise is a sequence of uncorrelated random variables X_t, each with the same mean and variance σ^2. Therefore, white noise is also an example of stationary time series. More specifically, the notation $WN(0, \{R_t\})$ indicates white noise with zero mean and variance R_t.