Fuzzy Logic-Based Approximate Event Notification in Sparse MANETs

Anna Lekova
Institute of Control and System Research, Bulgarian Academy of Sciences
P.O.Box 79 Acad. G. Bonchev
1113 Sofia Bulgaria

Katrine Stemland Skjelsvik, Thomas Plagemann, Vera Goebel
Department of Informatics
University of Oslo
P.O.Box 1080, Blindern
NO-0316 Oslo, Norway

ABSTRACT

Mobile Ad-Hoc Networks (MANETs) are an important communication infrastructure to support emergency and rescue operations. To address the frequent disconnections and network partitions that might occur, we have developed a distributed event notification service (DENS) for sparse MANETs. In most event notification solutions, subscriptions are formed with crisp values or crisp value ranges. However, in emergency and rescue operations subscribers may not always have time to give crisp values or crisp value ranges. Moreover, subscriber's interests in queries have gradual nature and subjective measure that calls for computing by words. Therefore, we design and implement a simple fuzzy-concept based subscription language allowing more expressive subscriptions and more sophisticated event-filtering. It is built on two new ideas: using features as multi-attribute indexes of the subscription and predicate patterns for processing subscriptions with arbitrary Boolean operators. However, requiring more computational efforts, fuzzy logic introduces performance penalties in the whole network. The proposed services have been evaluated for run-time, space and scalability efficiency. The proposed design framework is extensible to the user- and application-semantics and configurable to the dynamics in data that publish/subscribe paradigm imposes at runtime.

Categories and Subject Descriptors

C.2.1. [**Computer Systems Organization**]: Computer-Communication Networks- d*istributed networks.* D.3.3 [**Programming Languages**]: Language Constructs and Features – *frameworks*, d*ata types and structures.* I.2.3 [**Computing Methodologies**]: Artificial Intelligence– u*ncertainty, ``fuzzy,'' and probabilistic reasoning.*

General Terms

Algorithms, Performance, Design, Languages.

Keywords

Sparse MANETs, middleware, approximate event notification.

1. INTRODUCTION

In the Ad-Hoc InfoWare project [10], we are developing middleware services for emergency and rescue applications, like command and control, dispatching of resources, and remote patient monitoring. Mobile Ad-Hoc Networks (MANETs) are typically used for emergency and rescue operations, because they are often performed in environments without any or only a very rudimentary amount of existing infrastructure. However, these networks might be sparse with low node density and high node mobility. Therefore, disconnections might happen frequently and the middleware services have to be designed in such a way that they gracefully handle network partitions and provide delay tolerant services to the applications. To meet this requirement, we have developed a delay tolerant Distributed Event Notification Service (DENS) [14]. One of the important design decisions for the DENS is to decouple subscribers and publishers. A subscriber is a node interested in being notified when an event occurs. An event is a change in data of interest, the node that is identifying this change in data is called a publisher. A subscriber is describing in a subscription “when” and “what” it wants to be notified about by means of selection predicates. A notification is a message that describes an event according to the current subscription interests. Decoupling subscribers and publishers means that subscribers do not need to be able to directly connect to publishers and they even do not need to know the name or address of the publishers and vice versa. A set of so-called DENS nodes forms an overlay and serves as mediators between publishers and subscribers. This overlay is responsible for delivering subscriptions and event notifications to their destinations, to replicate them, and to perform *store-carry-forward* operations in case of disconnections and network partitions. In order to save resources, the DENS delivers only events of interest, i.e., it performs source filtering. For example, a remote patient monitoring application might be interested in the event that the body temperature of any patient is falling below 34 °C or gets higher than 40 °C. This subscription is forwarded to the DENS, which in turn forwards it to all sources, i.e., health sensors on patients. The filtering of the event is done within the sensors. Only in case the filter criteria are fulfilled, a notification is sent from the sensor to the DENS, which in turn delivers it to the subscriber.

In addition to the dissemination process, the other main design decision for the DENS is the subscription specification and event filtering, i.e., the subscription language, that determines the usefulness of the DENS for the many different applications. Rich languages can be used to specify complex filters, but their processing in the publisher nodes and also in the DENS requires more resources than simple languages. If a simple and efficient language is sufficient for an application, it should be used to save the scarce resources. However, there might be applications that would benefit from more complex subscription languages.

In most event notification solutions, subscriptions are formed with crisp values or crisp value ranges. However, in emergency and rescue operations subscriber's interests often have gradual nature and subjective measure or there may not always be time to give

crisp values or crisp value ranges in queries. Subscribers do not want to worry about violating the interval-ranges. Moreover, there might be applications that would benefit from more complex subscription filters, e.g. to capture uncertain data during rescue operations. For instance, agents in the RoboCup rescue scenario gather and circulate information for observed fires, smoke or even "the danger" in the vicinity. Transitional values as "semi-burning" would be helpful and need to be modeled by qualitative terms in predicates. Agent online decisions might be aided by an intelligent approach for computing with words by exploiting fuzzy logic, introduced by Zadeh **Error! Reference source not found.**. Therefore, we design and implement a simple fuzzy-concept based subscription language, data structures and filtering algorithms allowing more expressive subscriptions and more sophisticated event-filtering. It is built on two new ideas: using features as multi-attribute indexes of the subscription and predicate patterns for processing subscriptions with arbitrary Boolean operators.

However, requiring more computational efforts and space, fuzzy logic might introduce performance penalties in the whole network when not carefully designed. There are three areas in which space and computational efficiency is especially important: First, the representation of fuzzy logic based subscriptions might consume too much space on the nodes, i.e., subscribers, publishers and DENS nodes, and in the network in form of subscription and notification messages. Second, source filtering is performed on sensor nodes and might be too complex for these nodes. Finally, to forward notifications from publishers to subscribers, the DENS has to match subscriptions and notifications. The efficiency and scalability of this forwarding process in the DENS overlay depends on the complexity of the matching. It is the contribution of this paper to describe the design, implementation and evaluation of a subscription language that is able to model uncertainties and solutions for efficient filtering and matching based on fuzzy inference.

The remainder of the paper is structured as follows: In Section 2, we briefly describe the DENS architecture. In Section 3, we present the requirements for a fuzzy-concept based service and related works. In Section 4, concepts, ideas and fuzzy solutions about event notification in uncertainty, as well as a simple fuzzy concept-based subscription language, are proposed. In Section 5 the implementation, emulation and evaluation of the proposed fuzzy-based data model and algorithms in Sun Java Wireless Toolkit and NEMAN, are illustrated. Finally, conclusions and future directions are given.

2. DENS

In this section, we give a brief introduction to the DENS middleware and the importance of event filtering to generate notifications, as well as the matching to identify the subscriber(s) who are interesting in these notifications.

The DENS service runs on some of the nodes in the network forming an overlay. Subscribers and publishers only have to send a subscription or notification to the DENS overlay, which in turn delivers it to its destination. Subscriptions are replicated among the DENS nodes to increase service availability. Figure 1 shows the DENS architecture. Based on it, three different protocols are developed for DENS nodes to manage and replicate subscription information, to disseminate notifications, and to deliver them to their destinations. The functionalities of event-filtering and routing are decoupled which enables independent development of protocols at routing layer and subscription language at the application layer. We support decoupling of subscribers and publishers and at the same time want to support more than one subscription language. This is achieved by using language specific plug-ins at the DENS and at the publisher node: A look-up plug-in used for finding publishers, a filtering plug-in for filtering events at the source node, and a matching plug-in for matching notifications sent to the DENS overlay and stored subscriptions. A more detailed description can be found in [15].

When a subscriber wants to send a subscription it uses the SUB/DENS protocol to send it. The DENS node or nodes receiving the subscription stores it, and finds the correct publisher using the Knowledge Manager (KM) [11]. The KM manages the information sharing and therefore addresses two important requirements for publish/subscribe systems for emergency and rescue operations: 1) that subscribers do not need to know the publisher and 2) that each organization should be able to use its own vocabulary. Each node keeps a view of local resources available for sharing, together with metadata about resources available on other nodes. Metadata extracts are exchanged with neighboring nodes as they come into range. The KM keeps a cross-vocabulary thesaurus for all a priori known vocabularies used by the participating organizations, offering services for synonym look-ups and locating related metadata resources. The thesaurus maps of concept terms between the vocabularies. One of the main component in the KM is the Data Dictionary Manager which manages the metadata-handling The Data Dictionary Manager maintains a Local Data Dictionary (LDD) and a Semantic Linked Distributed Data Dictionary. Metadata descriptions of local resources are enhanced with concepts from vocabularies, and registered for sharing in the LDD.

The PUB/DENS protocol is used for communication between DENS and publishers, i.e., subscriptions are sent from DENS to the publisher and notifications are sent from publisher to DENS. Source-filtering is used to reduce the number of events sent from the publisher. The filtering is done by a monitoring agent, called *watchdog* (WD), running on the publisher node. It generates notification control messages that describe an event according to the current subscription interests. The WD Manager is responsible for updating subscriptions on the publisher and starting the WD Execution Environment (WD_EE) with the specified subscription language and the correct filtering function. The WD Manager implements the interface between the DENS and the WDs. When the DENS sends a subscription using SUB/DENS protocol, the WD Manager registers it and checks to see whether the same subscription has been received before, and updates the subscribers currently interested in events at this node. The programming model of WD Manager has an interface to DENS to obtain information from the KM to find the protocol and content type of data which should be monitored for the received subscription. The WD_EE instantiates the corresponding class for monitoring data acquisition applications, processes data according to the content type (e.g. if the protocol is "file" the content handler is "text/csv") and generates notifications when an event occurs that fulfills some conditions specified in some of the active subscriptions. Afterward the WD Manager sends a notification to the DENS using the PUB/DENS protocol. Finally, a publisher needs to understand the subscription to filter the events of interest. The WD is able to be extended by plug-ins provided by the language developer defining the subscription language and filterng function, e.g. database and SQL querying, Data Stream Management Systems and continues

querying, XML and XML querying, XPath and Bloom filters [18], fuzzy sets and fuzzy inference, etc. The format of notifications is also language specific.

When receiving a notification, the DENS needs to identify the subscriber(s) and to deliver this notification. This is done by matching the incoming notification with stored on DENS subscriptions. Subscription language IDs are used by the DENS and the publisher to identify which particular plug-in has to be invoked for matching constraints in predicates of current subscription to actual values in the notification and evaluating how close they are.

The DENS/DENS protocol is used for communication among the DENS nodes, including replication of subscriptions and undelivered notifications, consistency management etc. The storage component is used for storing undelivered notifications since DENS has to perform *store-carry-forward* operations in case of disconnections and network partitions.

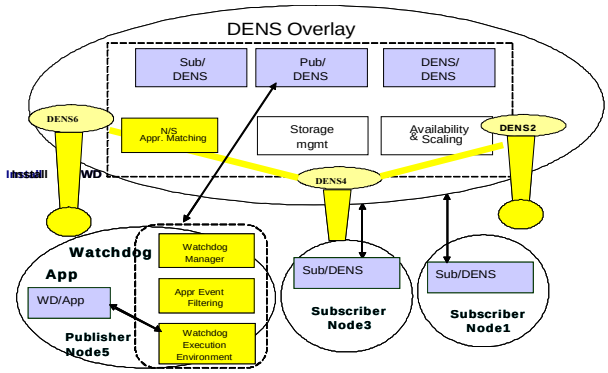


Figure 1. DENS Architecture

3. THE NEED OF FUZZY-CONCEPT BASED SERVICES

In this section, we analyze the need of fuzzy-concept based services for emergency and rescue applications, as well as DENS requirements for fuzzy model implementation. The state-of-the-art in expressiveness of content-based subscription languages and event filtering algorithms are revised, as well as the data-driven fuzzy modeling problem.

3.1 Requirements from Rescue Applications

Publish/subscribe systems and their languages are typically designed for the situation that many subscribers are interested in the same event. However, in rescue operations there are events that are only relevant to a very few subscribers, e.g., health status information is only relevant for those that are assigned to monitor the particular patient. In order to save resources, the DENS delivers only events of interest by performing source filtering. If an event of interest has happened on a publisher, an approximate notification is generated.

The proposed fuzzy-concept design is motivated by the need to handle and aggregate uncertainties in subscriptions and perform event filtering using natural language instead of numerical values. Since, the properties described often have gradual nature and subjective measure, for attributes such as “temperature” or “road access”, selecting terms such as “hot”, “clear”, “totally blocked”, etc. would be necessary. Modifying the linguistic terms, such as “warm” to “very warm” will allow flexible ways to specify the user interests, too. Certainly, we could use interval-ranged values to express uncertainties in a crisp system, however lacking of transitional boundaries does not take into account the values slightly out of the interval. Thus, subscribers need to worry about not violating the interval-ranges. At times, they might have been wrong in expressing their interests in exact terms. In other words, the tolerance level for how close or how loose information items in the published event adheres to the subscriptions’ interests is a design solution with respect to the error-tolerance in rescue operations.

As a natural conclusion, fuzzy logic is a suitable approach for representing the current subscribers’ interests about incoming events and handling an approximate fit to the data in notifications. Fuzzy logic is a superset of classical logic with the introduction of “degree of membership”. Uncertainties are presented as fuzzy sets (A_i), which are often expressed by words and interpreted by their membership functions $\mu_{A.}$ A *fuzzifier* is used at the input of the system to convert crisp to fuzzy data, while a *defuzzifier* - from fuzzy to crisp data. The fuzzifier converts each element of crisp data *x* from a certain domain (*D*) into fuzzy set using the concept of *degree of membership* of x to *A* - $\mu_A(x)$, which is a number in [0÷1]. Fuzzified input data trigger one or several rules in the fuzzy model to calculate the result. IF-THEN rules map the input values to the output space in terms of implication relation between fuzzy sets in “IF” and “THEN” parts. *Fuzzy reasoning* is numerically processing of the information in the fuzzy rules.

3.2 Requirements from DENS

To address DENS concerns about space and computational efficiency, designing the most efficient publish/subscribe algorithms and data structures is crucial. Subscription and notification messages need to be packed in as much as possible before sending to the DENS overlay, since the representation of fuzzy logic based subscriptions might consume too much space on the nodes and in the network in form of subscription and notification messages; and in form of replications. Replications forward much network traffic causing overhead and power dissipation.

The flow of messages from senders to receivers is driven by the content of the messages rather than by explicit addresses. This suggests using of data-driven model to perform computations in order dictated by data dependencies in subscriptions. Intelligent fuzzy system configurable to the dynamics in data that publish/subscribe paradigm imposes needs to be designed. The design of fuzzy inference system (FIS) consists of the following steps:

- determine the available inputs and output variables;
- determine the domains of variables and their sets of fuzzy terms;
- define the membership functions of the fuzzy terms;
- determine a fuzzy rule base;
- determine a fuzzy model and inference implication.

The main idea in classical fuzzy approaches from Zadeh and Takagi-Sugeno[16] is to calculate the result by evaluating the degree of matches from the measurement that triggered one or several rules in the fuzzy model. IF-THEN rules map the input values to the output space through composition and aggregation of the fuzzy sets in the “IF” part. Each rule is evaluated by an impli-

cation relation between the multi-dimensional fuzzy set in the "IF" part and output fuzzy set in "THEN" part that gives the relevance of the rule.

It is evident that a priori information about how the events should be processed according to the fuzzy rule base could not be placed on the local device since the available inputs, fuzzy terms and fuzzy membership functions depend on the data in the active subscriptions. Hence, an intelligent fuzzy model configuring and adapting its performance to the data in subscriptions, has to be designed. However, the majority of conventional fuzzy toolboxes fall short of design framework that allows customization and self-configuration of the fuzzy model at runtime. Conventional interactive fuzzy tools as MATLAB could not be used although they provide exporting of FIS as a stand-alone application. First, FIS has a fixed number of inputs and outputs, as well as a fixed set of fuzzy rules. Second, the size of the FIS is about 25-30KB since the toolbox supplies a fuzzy inference engine that can execute FIS as an external application. This memory footprint is not practical and useful if resource-constrained devices.

3.3 Related Work

A detailed survey and classification of subscription languages is made in [6] and [4]. A content-based publish/subscribe language is the most expressive language using flexible scheme for specifying filters. The filters utilize Boolean-valued functions over subscription predicates, e.g. ("temperature_value >38C"). As in the publish/subscribe system Elvin [13] we use source-filtering to reduce the number of notifications sent from the publishers. In Elvin these "reverse subscriptions" are called quench expressions and sent to the publishers to stop them from sending notifications to which no subscribers are interested.

The filtering process is influenced by the complexity of the subscription model. Filtering of events based on conjunction of crisp predicates is not a costly task. Filtering a subscription interested in arbitrary Boolean operators among predicates at a specific location and expressed in fuzzy terms, is obviously more costly. Almost all filtering algorithms support only conjunctions of predicates in subscriptions. They do not take into account the various combinations of predicates. They determine fulfilled predicates and count the number of matching predicates to be equal to the total number of predicates in the subscription.

The common practice if predicates are connected via arbitrary Boolean operators is to transform subscriptions into canonical expressions as disjunctive normal forms (DNFs) [6]. However, in sparse MANETs it is very questionable whether the resources are enough for transformation into canonical expressions. DNFs are exponential in size compared to their original Boolean expressions. In [6] an approach for arbitrary Boolean subscriptions without transforming to DNFs is presented.

A few existing publish/subscribe systems can model uncertainties for the information in subscriptions and notifications, and can perform approximate event filtering. Approximate-content based subscription language that allows uncertainties to be involved and indicates how to process these data, are proposed in [8, 9]. In [8] the theoretic basis of A-ToPSS is described. Its publish/subscribe model is based on possibility theory and fuzzy set theory to process uncertainties to the information in either subscriptions or publications. The filtering algorithm performs uncertainty inherent, exploiting one-dimensional index structure, proceeds in two stages. First, predicates are matched and second, matching subscriptions are identified. A fuzzy tuple space implementation extends LighTS [9]. The matching semantic combines the membership degrees using a fuzzy aggregating operator from the Fuzzy_Tuple library. Partial scores return true only if the membership value of the crisp value found in the field being compared is higher than a given threshold, associated to the membership function in a tuple.

The approach that these two fuzzy systems use is so called *flat modeling approach*. It works in problem domains where there is little or no dependency between input variables. However, decision-making during rescue operations often require complex aggregating function of two or more predicates that introduces relationships among data. For instance, all predicates might be location and/or temporally restricted. Flat modeling will yield a decomposition error. Decomposition into fuzzy Cartesian product space taking into account various combination of predicates which directly addresses this problem through breakdown of the input fuzzy sets [2]. We recall that if X_1 and X_2 are two crisp sets, their Cartesian crisp product $X_1 \times X_2$ is a set consisting of all pairs (x_1, x_2) where $x_1 \in X_1$ and $x_2 \in X_2$ [17]. The notion of the crisp Cartesian product can be extended to the fuzzy Cartesian product (FCP). The Cartesian product of fuzzy sets X_1 and X_2 is defined as $R_{X1\times X2} = \{ \mu_{X1\times X2}(x_1, x_2)/(x_1, x_2)\{ x_1 \in X_1; x_2 \in X_2\}$. A fuzzy relation (x_1,x_2) is a two-dimensional fuzzy set over the Cartesian product of crisp sets X_1 and X_2. It specifies more than presence or absence of an association between the elements of the crisp sets. A two-dimensional membership function for correlated attributes in subscription is defined as:

$$\mu_{X1\times X2}(x_1, x_2) = T\text{-}(co)norm(\mu_{X1}(x_1), \mu_{X2}(x_2))$$

Different approaches have been proposed for data-driven fuzzy modeling. A survey of such systems, as well as some improvements of interpretability of generated fuzzy models is given in [5]. The main goal is to develop an automatic rule generation mechanism without any assumption about the structure of the data, such as the number of attributes, number of fuzzy sets for each attributes, etc. In the current work the interpretability is improved by a rule-generation mechanism which reduces the number of produced rules using a general fuzzifier and fuzzy rules with variable fuzzy regions. The approach is similar to those in [1, 7].

4. MODELING AND PROCESSING OF UNCERTAINTY

In this section we describe our design solutions for a simple fuzzy-based subscription language and its data structures and filtering/matching algorithms, conforming to DENS requirements for minimal use the resources of mobile devices.

4.1. IDEAS, CONCEPTS AND MODELS

Fuzzy logic is a suitable approach for modeling and solving tasks for classification in uncertainty. We exploit fuzzy sets for modeling imprecise information in subscriptions. Membership degrees are first computed, which evaluate the values of the notification to the constraints in the subscription predicates, which are in a form of membership functions. Then, aggregating operators compute the overall score of matching for each subscription to a notification by a degree of certainty. The type of aggregation is user- or application- related. An XML subscription language is provided that enables one to write queries over events. Using XML schema subscribers do not have to learn a completely novel language and an extension mechanism of XML schema is used to make a sub-

scription language extensible to user- or application- needs. Different averaging functions might have been defined, such as more or less sensitivity to some attributes or fulfillment by their weighted or balance average. The semantics of equally named aggregating functions might not be the same. For instance, AND could be a minimum, product, averaging one. If the aggregating functions are complex or highly non-linear, a mechanism for their approximation to reduce the complexity in calculations is provided.

We created and implemented an extensible design framework configurable to the dynamics in data that publish/subscribe paradigm imposes to FIS at runtime. It provides the programming of data-driven fuzzy knowledge-base system which core consists of a set of interfaces and classes having abstract methods for collecting and structuring of accessible subscriptions into fuzzy rules. Fuzzy model configures itself by setting up interfaces and classes that model FIS (such as Mamdani or Takagi-Sugeno), aggregating functions among predicates, internal parameters of FIS (such as the number of input parameters), etc. The framework uses an extensible subscription language for defining new operators about the type of predicates or aggregators. In case of approximate predicates, the framework allows customization to different fuzzy terms at runtime. Extensions about new operators and associated plug-ins are processed by dynamic parsers which default implementation is characterized by the ability to extend the definition of rules how to process a format of new operators in subscriptions and how to map the uncertainties in predicates to the parameters for configuring the fuzzy model.

Our proposal for a data-driven fuzzy model is built on two new ideas: using features as multi-attribute index for the subscription and predicate patterns for indexing arbitrary Boolean operators in the subscription. Features and patterns are assigned to subscriptions and notifications as arrays of bits. They are used for configuring and adapting the fuzzy rules with variable fuzzy regions of the fuzzy model on the publisher or DENS node. These regions are customized still on the subscriber node from the dynamic parser that binary encodes and maps membership functions of original fuzzy terms in the subscription to the parameters of the universal fuzzifier used in the fuzzy model. Universal fuzzifier applies a membership function to the numerical values of notification attributes according to the fuzzy constraints in subscription predicates. Thus, space efficiency of the publish/subscribe system is improved, since the encoded subscription messages do not carry the whole mathematical description of original fuzzy membership functions along their lifetime. A single, unified data structure for binary encoding of different types of predicate constraints, such as crisp, crisp range-interval, string, date, time, approximate ones, is designed. Fuzzy terms reduce the number of subscription predicates to be evaluated since two predicates are needed to represent an interval-range value against one, if fuzzy term is used as a constraint. Another design solution is indexing, sorting and grouping of subscriptions according to their features and covering relations. If features of active subscriptions on the node are sorted, the filtering algorithm stops earlier rather than evaluate all subscriptions. If active subscriptions are aggregated according to their covering, many subscriptions can be evaluated using single matching. If we e.g., consider two subscriptions SA_1 and SA_2, SA_1 covers SA_2 if all events which are valid for SA_2 are also valid for SA_1 [4].

We call *approximate source filtering* the human-like fuzzy reasoning in linguistic terms how to filter the numerical values in incoming events using aggregated from active subscriptions imprecise information in the form of natural language. Contents of all subscription interests about events happened on a particular source node are transformed into fuzzy rules. Fuzzy inference will use this set of fuzzy rules for filtering of events afterwards. Subscriptions are processed according to their features. A feature is an interpreted value of an array of bits consisting of flags about the state of the subscription attributes. Features drive the filtering in two steps. In the first step on the PUB node, the *potential* notifications according to the features of active subscriptions are tailored. Covering of subscriptions could be exploited for reducing the number of sent notifications: if two notifications NA_1 and NA_2 match SA_1 and SA_2, where SA_1 covers SA_2, then NA_1 covers NA_2. Approximate evaluation of notification values to constraints in predicates is performed in the second step. We call *approximate matching* the reasoning in linguistic terms for how to match imprecise data in subscriptions to numerical values in the notification. The available subscriptions on DENS, summarized in form of fuzzy rule base, are matched to incoming notifications. In the first step on DENS node, the features of subscription vector (SV) and notification vector (NV) are matched via bitwise operators. In case of fitting in step two, we apply membership functions of all predicates in current subscription to actual values of attributes in the notification for evaluating how close they are. The overall score aggregates all degrees of membership and the subscription is relevant if the score is greater than the subscription's threshold.

4.2. Subscription Language and Data Model

An approximate subscription contains a set of approximate predicates combined by Boolean operators. The form of the approximate predicate is *attribute-relational_operator-membership_function* triple:

$$s(x_1,\cdots,x_m) = R(\{x_1 \; op \; \mu_{A1}(x_1)\},\ldots,\{x_m \; op \; \mu_{Am}(x_m)\}),$$

where x is the attribute name, 'op' is the operator, and μ is a membership function for a fuzzy set A. The relational_operator could be: *is*, *is not*, qualitative *is* as "more or less" or a temporally restricted *is* as "last few seconds". A subscription relation R defines a Boolean function connecting all predicates in *s*. Set operations such as union or intersection applied to fuzzy sets defined in different domains result in a multi-dimensional fuzzy set in the Cartesian product space of those domains. A notification consists of a set of attributes which have a name and a value. Mathematically, for a given notification vector X (x_1, ... ,x_m) in this space a subscription relation R constitutes a function defined over the fuzzy Cartesian product of the domains of X and yields a transitional truth value of *s* for this vector.

The subscription language specifies the queries over content of events using special tags (operators for defining the different type of predicates and aggregating functions) such as <crisp-predicate> - an operator on how to model constraints as crisp predicates, <aggregating-function> - an operator how to handle the logical expressions between predicates, <predicate-pattern> - how to handle arbitrary Boolean operators among predicates. The extension mechanism associates new interface methods to new defined tags that will be invoked at runtime.

Tags for representing <approximate-predicate> operator consist of information about fuzzy variables, domains, fuzzy sets and fuzzifiers. They are used to generate new linguistic variables. The

choice of the parameters of the fuzzifier customizes the semantics of the approximate predicate according to the user perception. In [19] Zadeh introduces so-called linguistic hedges or fuzzy quantifiers. They are mathematical operators that modify either the shape of the linguistic variable or its membership function reflecting a variation on its semantics. Thus hedges adjust the corresponding threshold of subscriptions' degree of fulfillment automatically.

<Approximate Predicate>
<Attribute type="fuzzy variable"> *Patient_Temperature*
<Fuzzy Terms>*low, ,high, very_high* **</Fuzzy Terms>**
<Domain_min> *30*°C **</Domain_min>**
<Domain_max>50°C**</Domain_max>**
<Fuzzy Set> <term>*low* **</term>**
<Fuzzifier> *trapezoidal (width, left, right)*
</Fuzzifier> </Fuzzy Set>
<Fuzzy Set> <term>*high* **</term>**
<Fuzzifier> *triangular (width, center)*
</Fuzzifier></Fuzzy Set>
<Fuzzy Set> <term>*very_high***</term>**
<Hedge>Very **</Hedge><term>***high* **</term>**
… </Fuzzy Set>
</Attribute>
<Aggregating-function>harmonic_mean**</Aggregating-function>**

An example describing a subscription in XML format, is shown on Fig.2. Since, XML documents require expensive string-based parsing that will introduce performance penalties on DENS, subscription is decoded in binary format on the sub node by a dynamic XML parser at runtime. A subscription, as a vector with variable-length of bytes is attached to SUB/DENS protocol packets. For 877 bytes of subscription in XML format and 20KB for XML parser corresponds 17 bytes of subscription in binary code and 4KB source code for approximate filtering.

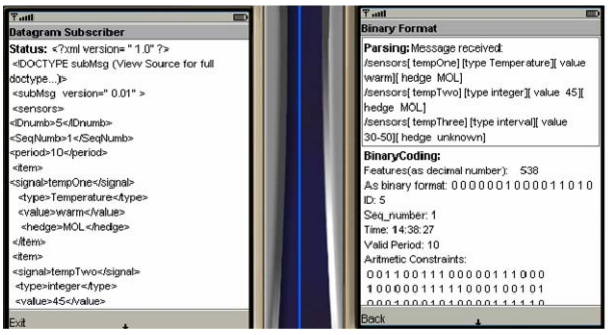


Figure 2. Expressing uncertainties in XML subscription

The rules how to parse the tags in the <Approximate Predicate> is accomplished by implementing new interface methods extending the rules of dynamic parsers by plug-in a new functionality These methods customize how to map the uncertainties in predicates to the internal parameters of the fuzzy model. After parsing the <Aggregating-function> operator, a variable is assigned to the internal data structure that indicates the aggregator used, such as minimum, product, averaging fuzzy aggregators, etc. There is an XML operator for defining complex aggregators with matching semantics based on a function of two or more predicates. It provides tags for describing the local linear models if the function could be approximated in fuzzy Cartesian product space.

Attributes Look-up:

Nodes that are publishers announce their attributes to the KM (see Section 2). The KM offers services for synonym look-ups and locates related metadata resources keeping a cross-vocabulary thesaurus for all a priori known vocabularies used by the participating organizations. We assume that the described events of interest in a new subscription are metadata entries in the LDD and attributes are concept terms from a known vocabulary. Subscribers have two options to be informed whether the information they are looking for is monitored: either by querying the KM or by browsing the concept terms of local sources. For instance, a subscriber can be prompted that two publisher nodes in a current partition have "entries for temperature measurements" with a concept term "temp" from medical vocabulary. An attribute specific look-up function queries the KM for finding the meta-information about which concept terms user could use as attributes in subscription predicates. Synonym look-up services aim the subscriber in questioning.

The following assumptions have been made: 1) The supported events in the whole system are categorized as metadata entries and indexed a priori according to their names; 2) The supported attributes of each event are linked to concept terms and indexed a priori according to their names; 3) The domains of all attributes are defined a-priori; 4) A basic set of fuzzy terms for describing concept terms is predefined.

Concept terms as indexes, in fact are the position of an attribute in features that point the presence or absence of interest in current subscription. For instance, the index for an attribute "wind_direction" is 1, i.e. the first bit in the binary array is active. Based on these indexes the dynamic parser assigns subscription features (SF) to each subscription. The corresponding bit or bits in SF is "1" when there is a constraint value about this attribute and its cardinality determines how many bits are charged. For instance, if we have a SV interested in attribute 4, 5 and 6 the SF is then 56 and the cardinality is 3. Thus, the subscription message does not carry the names of attributes since they are indexed in features according to the names of their concept terms. The features state which event (LDD metadata entry) should be monitored and which attributes (concept term) of that event the subscription is interested in. One subscription may be interested in only the temperature values, another in both temperature and wind speed. Notification Feature (NF) is an array of bits and a bit(s) is charged if this event should be monitored**.** Attributes of this event are indexed according to the subscription feature SF. The corresponding bit or bits in NF is "1" when there is a constraint value about this attribute and its cardinality determines how many bits are charged. The WD in the current implementation generates a notification message as a vector of bytes with only these values, for which there are an interest in the SF. If the event is $(a_1^{(1)},\ldots,a_k^{(1)}),(a_1^{(2)},\ldots,a_k^{(2)}),\ldots$,where a_k^T are attributes of the time-ordered events, *potential* notifications according to subscriptions $SV_1,\ldots,SV_i$ are:

$$(x_1^{(1)},\ldots,x_{card(SF_1)}^{(1)},NF_1^{(1)}),(x_1^{(1)},\ldots,x_{card(SFn)}^{(1)},NF_n^{(1)}),\ldots$$

where $NF_n^{(1)} = SF_n$ and $card(SFn)$ is the number of attributes that participate in the current subscription. The assigned at the end NF are same as SF. The corresponding NF is 56 and the NV is 11 bytes.

The lengths of SV (NV) are tailored according to the SF. SF and NF are *k+6* bytes, where *k* is the cardinality of SF. First 6 bytes in the SV or NV store info about ID number, sequential number, valid period, timestamp and aggregatorID. The offset for each attribute pointing which bytes contain its fuzzy constraints in SV are defined after masking the bits in SF from the beginning of SV (Fig.4.a.). The values of constraints are normalized in the range [0÷1] according to their domains and are mapped to the parameters of the universal fuzzifier (Eq.1.) used in the fuzzy model to calculate degrees of membership. Constraints values are disseminated as a sequence of three fuzzy parameters (v_i;V_i,γ), where v_i and V_i are the right and left borders of the area where the membership grade is 1 (fig.3).
Therefore, the varying length of *SV (NV)* and its bytes for fuzzy constraints is described as follows:

$$SV = 6 + card(SF_i) * 2 + card(SF_i)$$
$$NV = 6 + card(NF_i)$$

Notation γ is a sensitive parameter describing the generalization region of the corresponding fuzzy set. γ determines the spreads of membership function as a tangent of $\angle CAF$ which usually is between 30 and 89 degrees and γ is in the range [0.57÷57]. The larger γ the less fuzziness. In order to achieve a single, unified data structure and faster event filtering, we decode crisp values, crisp value ranges and string predicates like approximate ones. When subscription consists of string and crisp predicates, v_i=V_i, and γ parameter is large enough to decreases the fuzziness of the constraint value in the predicate. γ eliminates the spreads of fuzzy set since the angle is close to 90 degrees and γ is large.

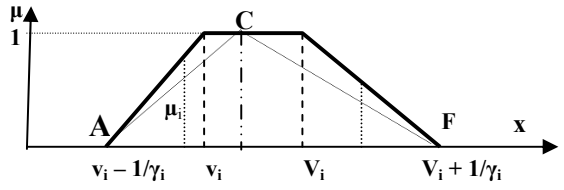


Figure3. One-Dimensional membership function

$$m_{A_i}(x_i) = 1 \text{ if } v_i < x_i < V_i; or\,[1 - max(0, min(1, \gamma(v_i - x_i)))] \text{ if } x_i < v_i;$$
$$or\,[1 - max(0, min(1, \gamma(x_i - V_i)))] \text{ if } x_i > V_i \qquad (1)$$

More expressiveness requires more composite data model for representing subscriptions. This influences the complexity of filtering/matching algorithms, as well as how much these algorithms can be optimized. The following data structures are proposed: without indexing predicates, with indexing predicates and data structure for decomposing the predicates into Cartesian product space. For all structures each bit in features indexes the offset in SV, where a membership function describes the user's constraint about this attribute.

The data structure on Fig. 4.a. stores the constraints for each predicate according to its fuzzy set definitions in SV. NV encodes actual values of attributes in the notification. A subscription list stores the overall score (Λ) after aggregating all membership degrees applying fuzzy aggregator based on its ID. A predicate hashtable depicted on Fig.4.b. indexes predicates according to their names and store the fuzzy constraints for each predicate. It allows the degree of match for each predicate to be evaluated only once using an index. Subscription evaluation is based on the association list linking the subscriptions that contain that predicate. A subscription list keeps the overall score of each subscription. The overall data structure with decomposition of attributes into Fuzzy Cartesian Product is depicted on Fig. 4.c. If $x_i^{(t)}, i = 1,...,m$ are the number of attributes participating in the current feature, the two-dimensional membership functions that this data structure encodes in break-down subscription vector (BDSV) are: $(\mu_{x_1^{(1)} \times x_2^{(1)}}, ..., \mu_{x_i^{(1)} \times x_j^{(1)}}, SV_n^{(1)})$, where $\mu_{x_i^{(t)} \times x_j^{(t)}}, i, j = 1,...,m; j > i$. The number of all two-dimensional μ is $proj = \frac{m * (m - 1)}{2}$, *where m is the cardinality of* $SAF_n^{(1)}$. Subscription list stores the aggregated by fuzzy mean formula [16] degrees of match from all fuzzy relations.

A tag <predicate-pattern> has been designed to provide user to write more flexible subscriptions for processing arbitrary Boolean operators between predicates. Using features, implemented operators for predicate patterns (PP), such as *d2c1*, and the data structure on fig.4.d, the XML parser maps the information in these tags into a vector of variable length of bytes. The seventh byte encodes the predicate pattern as a consequence of flags. The first and third bits code the operator between operands as conjunctions (c) or disjunctions (d), the second and fourth bits - the type of the operand: predicate or fuzzy relational pair (FRP), the last two bits point how many bits the pattern uses. Next the vector consists of several bytes defining how many FRP participates and which one- or two- dimensional membership function (μ) to be processed. The last *k* bytes before features carry universal fuzzifier parameters, where *mbf[i] is a sequence of $\{v_i, V_i, \gamma_i\}$. For instance, in case of subscription $(p1 \vee p2 \vee p3) \wedge p4$ the number of last *k* bytes is 21. 3 two-dimensional μ correlating two one-dimensional μ, each μ is encoded by 3 bytes $\{v_i, V_i, \gamma_i\}$ plus one-dimensional μ*3 bytes: (3 FRP*2)*3 +1*3=21.

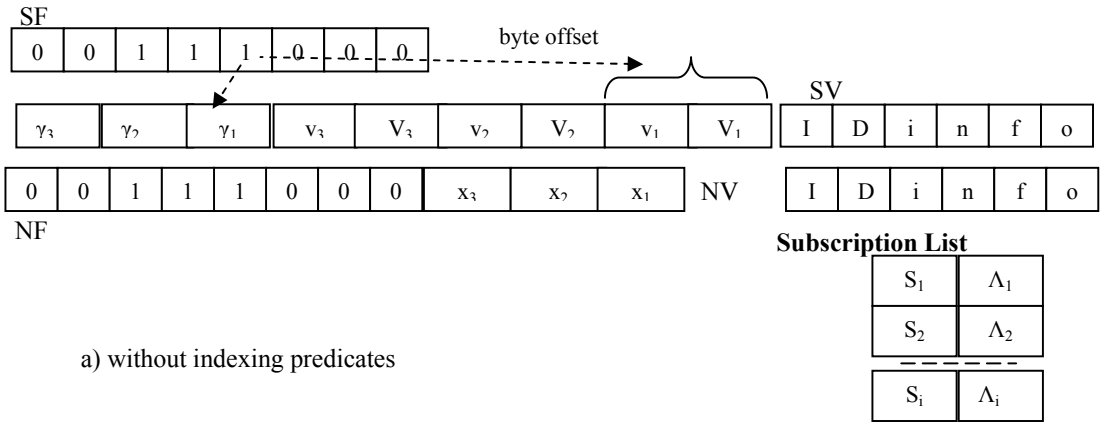
4.3. Processing of Uncertainty

We propose three types of filtering/matching algorithms, one using predicate indexing, one without indexing and one with decomposition of the input variables into fuzzy Cartesian product. In the non-indexing algorithm, predicates of each subscription are tested independently by applying their membership functions to values of attributes in the potential (or actual) notification. Then an aggregating operator computes the overall degree of matching for each subscription. Using predicate indexing the number of subscriptions that must be examined is reduced. There are two stages in the indexing algorithm. In the first stage, predicates are evaluated and, in the second, satisfied subscriptions are identified and evaluated by their overall scores. In the break-down algorithm for function approximation, first the membership degrees are computed and then all two-dimensional degrees by decomposing of the predicate constraints into fuzzy Cartesian product. If local linear models describing the system's behavior in all FRP would be found, the aggregating function is approximated in *THEN* part of the fuzzy rules and the rules are aggregated and defuzzified by using the fuzzy-mean formula [16].

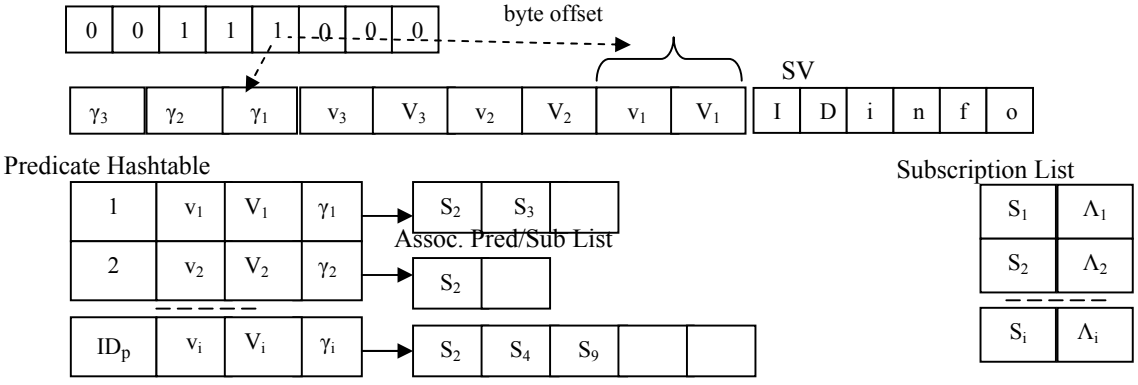
The TS is of first-order if the *THEN* parts consist of a linear function and of zero-order - if consist of a scalar. In the break-down

algorithm for arbitrary Boolean subscriptions the decomposition of the input variables is performed only in those fuzzy relations which attributes participate in features and have number bigger than 1 after the connective operator in the pattern. For instance, if d2c1 and features 00000111 – a two-dimensional μ in FRP $x_0 \times x_1$ and one-dimensional μ about x_3 will be evaluated. Individual contribution of fuzzy relations to the overall degree of matching is performed according to their connective operators, i.e. all FRP with *AND* connective are aggregated and defuzzificated by using the fuzzy-mean formula and all FRP with OR connective – by maximum mean. The overall score aggregates them according to the last in pattern connective operator.

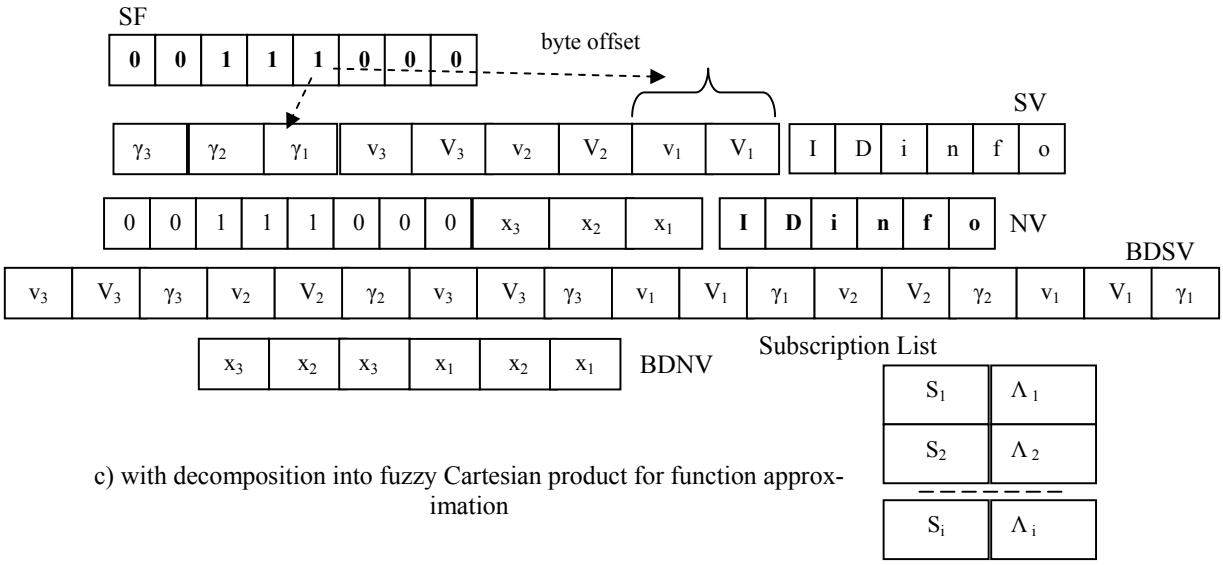
Non-indexing approach is designed for filtering on the publisher node since *if* at least one subscription is satisfied *then* the notification is sent and the filtering continues only with subscriptions that have different features than the sent one. We optimize the matching process on the DENS nodes by first checking the subscription and notification features for potential hits. Only in case of relevant features, matching of notification values to constraints in predicates is performed in the second step. A flat non-indexing algorithm with features (FNIF) proposes optimizations using features as multi-attribute indexes. If features are sorted faster filtering/matching is performed in a first step. A flat non-indexing algorithm with sorted features (FNIFS) has been designed. A flat non-indexing algorithm with features that are assigned after covering the subscriptions first is FNIFC. FNIF, FNIFS, FNIFC and a flat algorithm with indexing of predicates (FIP) have been designed for matching on the DENS node. Three breakdown filtering algorithms decomposing the input variables into Cartesian product space have been designed, as well They allow complex arbitrary Boolean relationships among predicates to be processed (BFCPAB) and function approximation using of first-order (BFCPA1) and of zero-order TS system (BFCPA2).



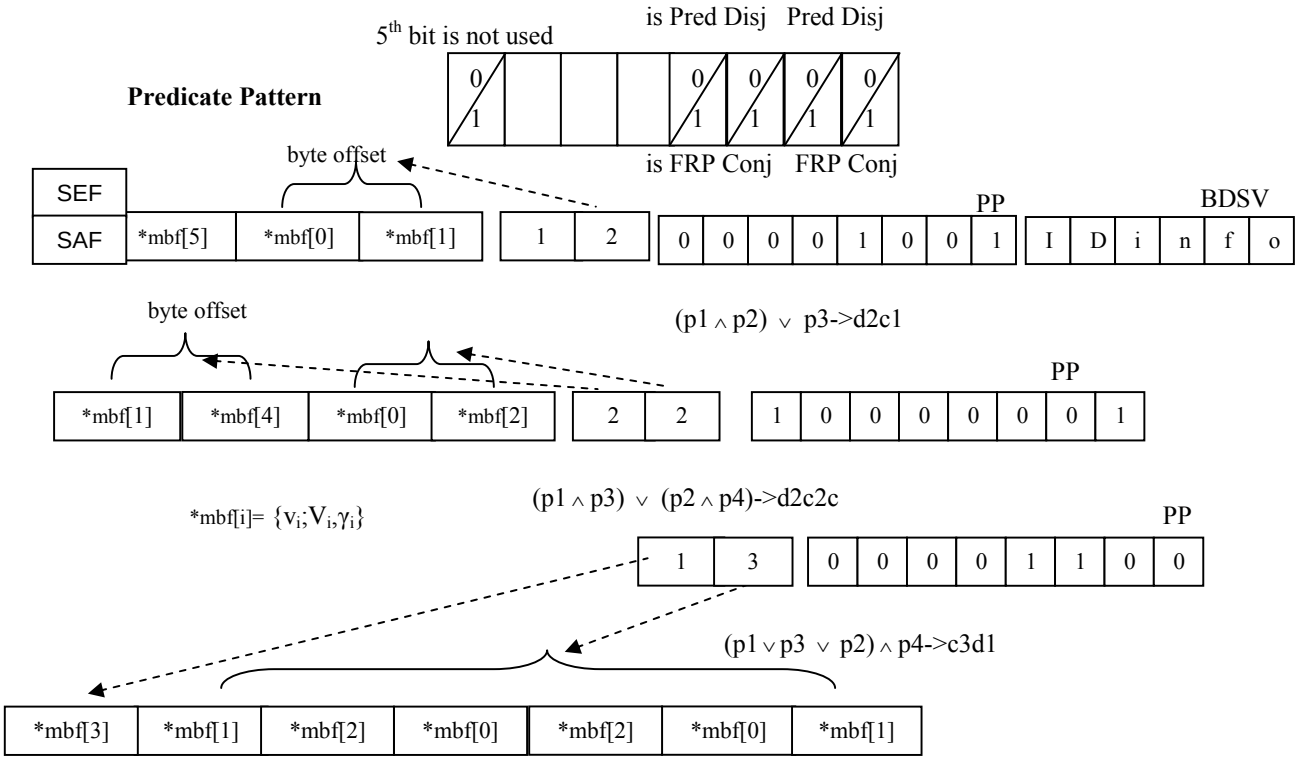
a) without indexing predicates



b) with indexing predicates



c) with decomposition into fuzzy Cartesian product for function approx-imation



d) with decomposition into fuzzy Cartesian product for arbitrary Boolean subscriptions

Figure 4. Data Structures

4.4. Optimizations

The main goal of optimization is reducing the event filtering time. On the one hand, reducing the number of subscriptions that have to be evaluated and on the other hand, reducing the complexity in calculations that more expressive subscriptions introduce. Some CLDC (Connected Limited Device Configuration) devices have no math library that includes floating point calculations. Using CLDC math library for SQRT, sin, tangent, etc., increases the matching time significantly. We applied the above two break-down algorithms for approximating those functions and replaced the floating-point math by approximate fix-point math.

Reducing the processing time by indexing of predicates or sorting of subscriptions according to their features results in fast event filtering. The pseudo code of the optimized matching algorithms using features is presented in fig.5.a. The sorting algorithm exploits some properties for starting evaluation if and only if the feature of notification is bigger than the first subscription's feature and less than the last subscription's one. Enumeration of available subscriptions starts if and only if the feature of the notification is less than the current one. A pseudo code for FNIFS is presented in Fig.5.b. The covering algorithm (fig.5.c.) takes advantage of similarity in the system, which is significant when approximate terms are used. However, dynamism in the topology of sparse MANETs imposes algorithms for insertion, covering, intersection and sorting of subscriptions to be fast and efficient to make the loading time of subscription as low as possible.

```
Input: SV, N
Variables: s=number of subcriptions,
i=number of predicates
SatS:set of satisfied subscriptions
Body:
1. for each subscription s_s in SV
        locate its feature SFs_s
        if (XOR-ing SFs_s with NF) ==0;
        then
For each attribute in features locate corresponding (v_i;V_i,γ)
  compute μ_i
        else continue next s_s;
2. for all attributes switch(aggregator)
3. defuzzification of μ_i to yield FDM
4. if  FDM_s>threshold  then SatSub U SV[s]

        return SatSub
```

Figure 5.a. Pseudo-code of matching algorithm FNIF

```
1 Step –Sorting SV
InpI  Input: SV
      Variables: i=number of subcriptions,
      Feat[s_s]  - array storing active features
      SFeat[s_s]  - array storing sorted features
      SSV-vector storing sorted subscriptions
      Body:
      1. for each subscription s_i in SV
            locate its feature SFs and store in Feat[s]
      2. SFeat[s_i] =Hoare's Quick Sort algorithm (Feat[s_i], 0, Feat[s_i].length)
      3. for each feature f in SFeat[s_i]
            for each subscription s_i in SV
            if(SFeat[f]= = SF[s_i]
                  SV.removeElement(s_i);
                  SoSV.addElement(s_i);
      4. Return SSV

      2 Step:
      Input: SSV-vector storing sorted subscriptions, N
      Variables: i=number of subcriptions,
      SatS: storing set of satisfied subscriptions
      Body:
      1. for each subscription s_i in SSV
            if(NF>60) locate last element from SFs_s
            locate feature SF s_i
            if NF>=SF s_i exit
            if (XOR-ing SFs_s with NF) ])= =0)
            then: for each attribute locate corresponding (v_i;V_i,γ)
```

Figure 5.b. Pseudo-code of matching algorithm FNIFS

```
1 Step –Covering SV
Input: SV, input subscription s
      Variables: i=number of subcriptions,
      AssocSub – vector (or hashtable) storing subscription's associations
      UnsubSub – vector (or hashtable) storing associated subscriptions
      Body:
      1. for each subscription s_i in SV
            boolean add_this_sub=false
            xor feature SF with SFi
      2. if (SF ^ SF_i) !=0:
            if(card>=card_i):
                  if(SF& SF_i +(SF - SF_i))= =SF:
               masking bits in (SF ^ SF_i ) gives the attrs in s_i that s_i does
               not have does not have
              skip evaluation of these attrs
            in case s_i covers s: AssocSub.put(sID, s_iID, SF_i)
            if(card<card_i):

                  if(SF& SF_i +( SF_i -SF))= = SF_i:

               masking bits in (SF ^ SF_i ) gives the attrs in s_i that s does not
               have;  skip evaluation of these attrs
             in case s covers s_i: AssocSub.put(s_iID, sID, SF)

      3. if (SF ^ SF_i) = =0:
            for each attribute constraints in s_s [card]
            if((s[vi]<= s_i [vi])&&(s[Vi]>= s_i [Vi])&& (s[γ]>= s_i [γ]))
            s_i covers s: AssocSub.put(sID, s_i ID)
            add_this_sub=true; SV.removeElement(s_i)
            UnsubSub.addElement(s_i)
            else if((s_i [vi]<= s [vi])&&( s_i [Vi]>= s [Vi])&&
                  (s_i [γ]>= s [γ]))
                  {  s covers s_i: AssocSub.put(s_iID, sID)
            UnsubSub.addElement(s)
            } else add_this_sub=true
      4. if(add_this_sub) SV.addElement(s);
      5. Return SV
      2 Step: same as in FNIF
```

Figure 5.c.Pseudo-code of matching algorithm FNIFC

5. IMPLEMENTATION AND EVALUATION

In this section the implementation, emulation and evaluation of the proposed fuzzy-concept based data model and algorithms in Sun Java Wireless Toolkit 2.3 (WTK23) and NEMAN [11], are illustrated.

We have designed a light–weight portable MIDP Java WD application supporting devices running Java2 Micro Edition. WD_EE implements a protocol handler interface for monitoring files on a local mobile file system (JSR75 API). We have implemented and evaluated the performance of a filtering algorithm with non-indexing of predicates in subscriptions, with and without sorted features and covered subscriptions. We evaluated the algorithm by indexing of predicates in the subscription on the DENS nodes, as well. An implementation of the arbitrary Boolean subscriptions

with patterns: p1 ∧ (p2 ∨ p3); (p1 ∧ p2) ∨ p3; (p1 ∧p2) ∨ (p3 ∧p4) and (p1 ∨ p2 ∨ p3) ∧ p4, have been tested.

We used NEMAN [11] as an emulation platform and the test setup is described in fig.6. NEMAN provides a virtual wireless network that can handle hundreds of nodes on a single machine. On one machine NEMAN is running and emulating a MANET with 50 nodes. In addition to the test machine we have used a wireless weather station "METEO_PRO WS 2305" with sensors collecting information such as temperature, wind speed, etc., and these sensors send values to a base station which logs the values. The collected values were then used as data for a publisher node. We used a simplified implementation of the KM, i.e., a simplified LDD for providing the necessary metadata. The WD is implemented on a laptop, and the WD Manager is linked to a node in the MANET which then acts as a publisher. WD has been emulated in a WTK23. It uses the PUB/DENS protocol to communicate with the DENS overlay. The nodes in the network picked out to be DENS nodes (grey nodes) uses the matching functions when receiving a notification.

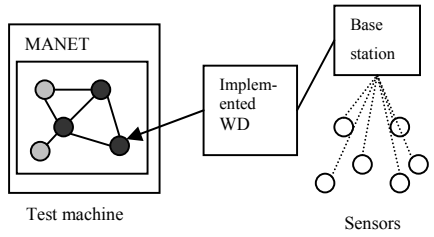


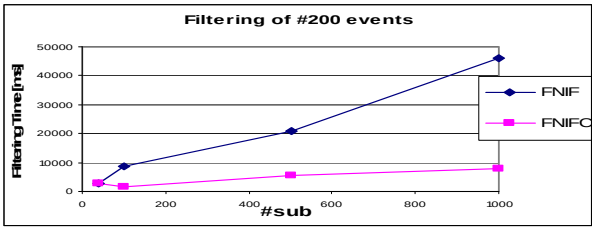
Figure 6: Test Setup

Since, the idea of DENS design is to decouple the subscription and notification delivery from subscription specification and event filtering, the performance costs have been measured locally on the publisher or DENS nodes and globally in NEMAN. Global cost evaluates the global network, mobility and application performance. In the present study we focused mainly on the local cost. Local cost evaluates the behavior and performance on the publisher and DENS nodes according to the application workload, the subscription specification and the degree of expressiveness of subscriptions.

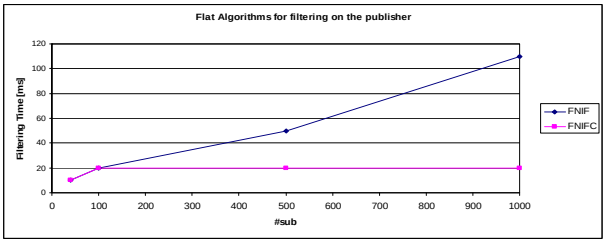
Our first experiment was to implement and compare the matching times for the crisp and fuzzy algorithms on DENS overlay in NEMAN. To put up a test scenario we had up to 1000 randomly generated subscriptions. The publisher starts publishing notifications over a 1 second period for 200 measured events. We varied the workload up to 1000 subscriptions with 3 and 4 attributes. The time to do the matching was ~100 ms for the fuzzy flat algorithm, ~600 ms for the "breakdown" one and negligible for the attribute-valued crisp algorithm. These results showed that fuzzy event notification could be well supported and is relatively comparable to crisp one.

Fig.7.a. shows the time for filtering of 200 events when the number of inserted subscriptions on the publisher is 40, 100, 200, 500 or 1000 every one second. Thus, the times for subscription insertion, deletion and their aggregation into fuzzy rules base have been tested. These times can be neglected if not covering of subscriptions is performed. Fig.7.b. shows the filtering times of one event according to different number of subscriptions using flat algorithms. FNIF increases linearly with the number of registered subscriptions. If subscriptions' covering is performed (FNIFC), the filtering time is flat and low – about 20ms since subscription equality in the system is increasing: 60% for 100 subscriptions, 92% for 500 and 96% for 1000 subscriptions. However, optimized algorithms require subscription loading time that needs to be taken into account (Fig.7.c.). Times consuming for decomposing of predicates using "break down filtering algorithm" are shown in Fig.7.d. Minimum, maximum, product, geometric- (SQRT), harmonic- and arithmetic-mean aggregating functions have been implemented. Different times have been observed when product, approximated SQRT; Java Math SQRT and arithmetic mean-aggregator (fig.9.e.) have been used. Since Java Math square root is a very costly operator, a precision of the SQRT aggregator has been traded-off against the time. Approximated SQRT aggregator is proposed to be used which matching accuracy is distributed within an error-interval of 1% around the true value. Expressiveness of the data model influences the scalability of the implementation on mobile devices to process up to 10000 subscriptions (Fig.7.f.). Memory is scarce on MIDP devices. The Memory Footprint of WD implementation is 49KB (MIDP Java Descriptor File +JARfile). A memory monitor in WTK23 examines the amount of memory currently allocated to each process over time. The default color mobile phone has 500KB RAM and the maximum amount of resident memory allocated for some processes is: for loading the objects of WD_M -118KB; for launching the MIDP application – 19KB; for starting WD_EE for content handler "text/csv" – 44KB. Allocated memory for filtering and covering for different number of subscriptions is shown in Fig.7.g.

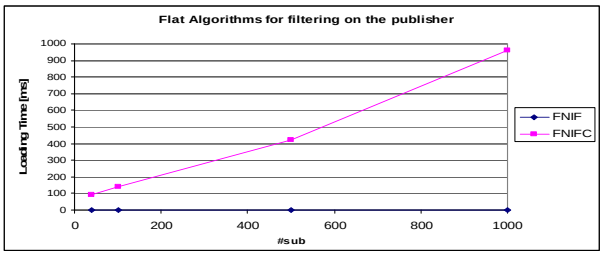
When a notification was sent to the DENS node in JWT23, the best performance shows algorithm exploiting sorted features and covering subscriptions - 6.5-7ms (Fig.7.h.). If predicates of subscriptions are hashed and indexed (FIP), the matching time is low too (6-6.5ms), however the loading time is large (Fig.7.i.). The Memory Footprint of MIDP implementation on the DENS node is 22KB. We compared times for the crisp and approximate matching algorithms when a notification was sent to the DENS overlay in NEMAN. We used workload of 100 subscriptions with 3 attributes. The time to do the matching was ~10 ms for the implemented FNIF, ~60 ms for "break down matching algorithm" and negligible for the simple attribute-valued crisp one.



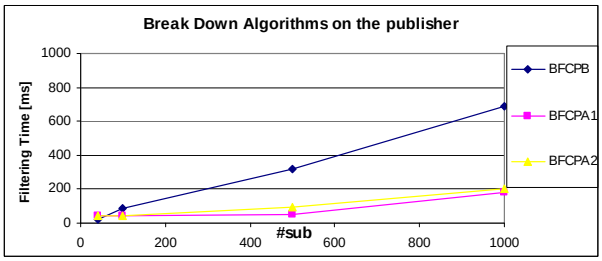
a) filtering time for 200 events



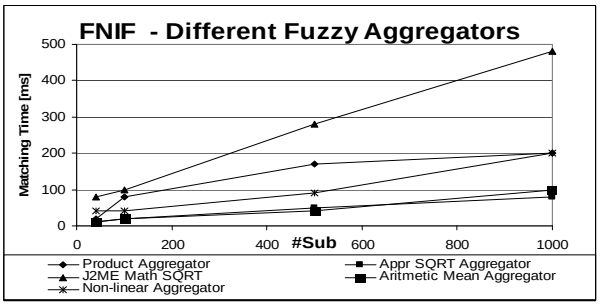
b) Filtering time on publisher



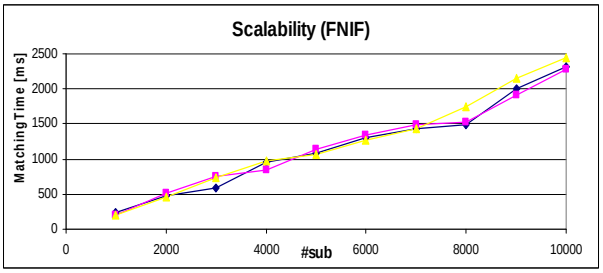
c) Loading time on publisher



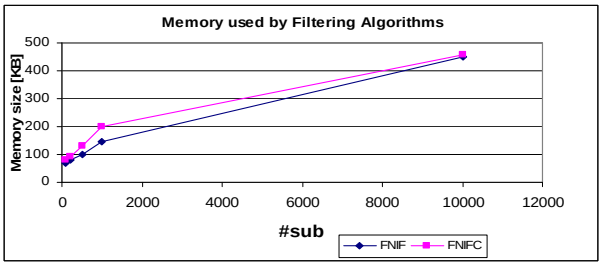
d) Filtering time on publisher



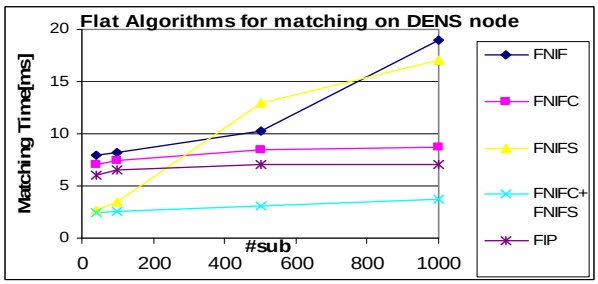
e) Different fuzzy aggregators



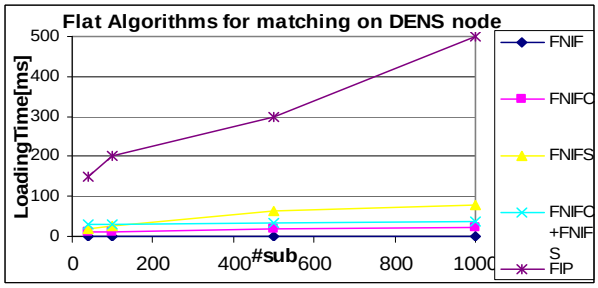
f) Scalability



g) Resident memory allocated for filtering processes



h) Matching time on DENS node



i) Loading time on DENS node

Figure 7: Emulations in Sun Java Wireless Toolkit 2.3

6. CONCLUSIONS

Sparse MANETs require efficient event notification services for information sharing that could be performed on mobile device without restricting the expressiveness of a subscription language and losing the portability of services. This paper presented a design of subscription language, data structures and algorithms that can be used to perform approximate event notification on the DENS overlay in the Ad-Hoc InfoWare project for emergency and rescue applications. Fuzzy logic has been successfully applied for modeling and processing of subscriptions in uncertainty. It not only making distributed event notification services more advanced, it performs error-tolerance in information providing and information consuming. We have established that crisp and approximate event filtering are comparable in time, space and scalability efficiency. Simulations results show that publish/subscribe can be well supported by an approximate filtering function, and that the load is tolerable for mobile devices with limited computing capability.

Since, the proposed design framework is extensible and configurable, the design of additional data structures and methods for time synchronization and composition of subscriber interests about events on DENS, will be easily set up. As for a future work, we would like to research how temporal relationships among predicates in subscriptions could be represented by fuzzy temporary restrictors, to study and model event correlations using the subscription features and to improve the attribute look-up services to facilitate subscribers in combining events.

7. ACKNOWLEDGMENTS

This research has been performed in the Ad-Hoc InfoWare project, funded by the Norwegian Research Council in the IKT-2010 Program, Project No. 152929/431.

8. REFERENCES

[1] Abe Sh, M. Lan, *Fuzzy rules extraction directly from numerical data for function approximation*, IEEE Trans. System Man Cybernet 25, 1995, 119-129.

[2] Babuska, H.B. Verbruggen, *Constructing fuzzy models by product space clustering.* In H. Hellendoorn.

[3] Carzaniga A., L. Wolf. *Content-based networking: A new communication infrastructure*. In NSF Workshop on an Infrastructure for Mobile and Wireless Systems, Scottsdale, Arizona, Oct. 2001

[4] Carzaniga, D. Rosenblum and A. Wolf. *Achieving Scalability and Expressiveness in an Internet-Scale Event Notification Service*, In 9th ACM Symposium on Principles of Distributed Computing (PODC), 2000, pp.219-227

[5] Chen M., D. Linkens, *Rule-base self-generation and simplification for data driven fuzzy models*. Fuzzy Sets and Systems 142:2, 2004, 243-265

[6] Jung D., A. Hinze, *A Meta-service for Event Notification.* Lecture Notes in Computer Science, Volume 3290 / 2004 , 283-300 29

[7] Lekova. A., at all, *Redundant fuzzy rules exclusion by genetic algorithms*, Fuzzy Sets and Systems 100, 1998, 235-243

[8] Liu, H.-A. Jacobsen. *Modelling Uncertainties in Publish/Subscribe*. International Conference on Data Engineering (ICDE), Boston, MA, June 2004, 510-522.

[9] Picco G., D. Balzarotti and P. Costa, *LighTS: a lightweight, customizable tuple space supporting context-aware applications.* ACM Symposium on Applied Computing, 2005, 413-419.

[10] Plagemann, T., Goebel, V., Griwodz, C., and Halvorsen, P. *Towards Middleware Services for Ad-Hoc Network Applications*. In the 9th IEEE Workshop on Future Trends of Distributed Computing Systems, Puerto Rico, May 2003, 249-257

[11] Pužar M and T. Plagemann, *NEMAN: A Network Emulator for Mobile Ad-Hoc Networks.* 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005.

[12] Sanderson N., V. Goebel and E. Munthe-Kaas, *Knowledge Management in Mobile Ad-Hoc Networks for Rescue Scenarios.* Workshop on Semantic Web Technology for Mobile and Ubiquitous Applications, ISWC 2004, 11, 2004.

[13] Segall, D. Arnold. *Elvin has left the building: A publish/subscribe notification service with quenching.* In Proceedings of AUUG97, Brisbane, Australia, September 1997.

[14] Skjelsvik K., V. Goebel, T. Plagemann,. *Distributed Event Notification Service for Mobile Ad Hoc Networks*, IEEE Distributed Systems Online, vol.5(8), 2004.

[15] Skjelsvik K., at all, *Supporting Multiple Subscription Languages by a Single Event Notification Overlay in Sparse MANETs.* MobiDE'06,June 25, 2006

[16] Takagi T., M. Sugeno. *Fuzzy identification of systems and its applications to modeling and control.* IEEE Trans. Syst., Man, Cybern., vol. 15, 1985, 116–132.

[17] Yager R., D.P.Filev, *Essential of Fuzzy Modeling and Control.* Wiley, New York, 1994.

[18] Yoneki E., J. Bacon. *An Adaptive Approach to Content-Based Subscription in Mobile Ad Hoc Networks.* Proceedings of the 1st International Workshop on Mobile Peer-to-Peer Computing, 2004, 92-97

[19] Zadeh L., *A fuzzy-set-theoretic interpretation of linguistic hedges*. Journal of Cybernetic, 2, 1972