ABSTRACT
The subscription language is an important design decision for distributed event notification services (DENS). In order to minimize resource consumption and enable applications to use rich and complex subscription languages only when they are really needed, we have developed a DENS that separates the concerns of delivering subscriptions and notifications from the subscription specification and event filtering, i.e., the subscription language. To resolve the conflict between subscription language independence in DENS and a strict decoupling of publishers and subscribers through the DENS, we request that for each new subscription language three language specific plug-ins are provided. In this paper, we present the technical details of this solution and describe our proof-of-concept implementation that supports a simple attribute-value based subscription language and a fuzzy concept-based language.

Categories and Subject Descriptors
C.2.1 [Computer-Communication Networks]: Network Architecture and Design – Store and forward networks, wireless communication

General Terms: Languages

Keywords: Subscription language, publish/subscribe, middleware

1. INTRODUCTION
In the Ad-Hoc InfoWare project [8], we are developing middleware services for emergency and rescue applications, like command and control, dispatching of resources, and remote patient monitoring. Since emergency and rescue operations are typically performed in environments without any or only a very rudimentary amount of existing infrastructure, mobile ad-hoc networks are used for communication. However, these networks might be sparse, e.g., low node density and high node mobility. Thus, 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) [13]. The DENS is decoupling subscribers and publishers. 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 is fulfilled, a notification is sent from the sensor to the DENS, which in turn delivers it to the subscriber. In this context, the subscription language is an important design decision with respect to 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. Another important reason to support multiple subscription languages is the fact that in emergency and rescue operations, devices from different organizations form a shared network and dissemination platform at the scene. Application developers and end-users from the different organizations would obviously benefit from using their domain specific vocabulary to specify the events they are
interested in. In [6], a meta-subscription language is suggested that enables different applications to use different subscription languages. All subscriptions are translated into the meta-subscription language such that the publish/subscribe system has only to handle one language. This approach is not meaningful in our application domain since such a meta-language is always the most complex language, which contradicts to our need to minimize resource consumption.

Therefore, we decided to design and implement the DENS in such a way that we separate the concerns of subscription and notification delivery from subscription specification and event filtering, i.e., the subscription language. The result of this work and the contribution of this paper is our solution to achieve subscription language independence for the DENS. In other words, the DENS can be extended with an arbitrary subscription language and once a new subscription language is plugged into the DENS it is supported by the DENS side by side with all other subscription languages that have been plugged in earlier. This multiple subscription language support and extensibility of the DENS is in conflict with the principle of decoupling subscribers from publishers. In order to identify the publisher to which a subscription should be forwarded and vice versa, the subscription respectively notification have to be analyzed by the DENS. We resolve this contradiction by requesting the designer of a new subscription language to provide three subscription language specific functions that must be plugged into the DENS: look-up() to identify the destination for a subscription, filter() to filter the event at the publisher, and match() to identify the destinations for notifications. Only these functions inspect the content of subscriptions and notifications. For all other tasks in the DENS, subscriptions and notifications are just opaque packets. A subscription language ID (SL_ID) is attached to all subscriptions and notifications and allows to invoke the language specific functions.

The rest of this paper is organized as follows: In Section 2, we discuss different subscription languages and related work. Due to space limitations, we can only give a very brief introduction into the DENS and the Knowledge Manager in Section 3 before we describe the main contribution of this paper, i.e., how the design of the service supports different subscription languages. In Section 4, we describe a proof-of-concept implementation in an emulator using two subscription languages. We conclude and discuss future work in Section 5.

2. Subscription Languages

In this section, we describe different subscription languages, analyze the requirements on subscription languages for rescue and emergency applications, and revise the state-of-the-art in publish/subscribe systems with respect to these requirements.

2.1 Categorization and Expressiveness

The three main publish/subscribe data models are channel-based, subject-based, and content-based. These models have different expressive power. In a channel-based system, subscribers subscribe to events from a specific source. In subject-based systems like TIB [15], subscribers subscribe to predefined subjects and each event has a subject. In content-based systems like SIENA [1], subscribers specify the content of the events they are interested in by using a filter. A Boolean filter function, like “temperature > 35”, returns true if an event matches the subscription, false otherwise. The content-based models are thus more expressive. The filter may consist of several predicates, i.e., attribute, operator, value tuples, for different event parameters, and operators. The more operators the language supports, such as conjunction, disjunction, negation, etc. the more complex and richer language and the more costly is the execution of the filter function. A subscription could also be extended to take into account context information such as geographic location and time. An even richer subscription language introduces uncertainties, i.e., in the filtering function there are not only operators such as “=” or “>” which result in true or false, but also “~” stating interest in an event which is more or less exact expressed in the subscription. The matching process will of course be influenced by the complexity of the subscription model. Filtering events just based on the subject is not a costly task. Filtering a subscription interested in composite events, in a specific order, taking place at a specific geographical location, at moving nodes, expressed in fuzzy terms, is obviously more costly. Examples of different subscription/filtering languages are: simple attribute operator value tuples, XQuery or XPath for XML, fuzzy predicates, and continuous queries for Data Stream Management Systems.

2.2 Requirements from Rescue Applications

In a rescue operation, there are numerous events that are potentially of interest for either the applications and end-users or the middleware. This includes readings from health status sensors that are useful for medical applications and updates in routing tables that indicate that new nodes have joined the network. Some of these events and their sources might be known a priori, however, many of them not, because the fact that certain events are of interest and who generates them depends on the particular rescue operation. 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. Furthermore, it is important to only notify about events if they are really important, because information overflow is dangerous for rescue personnel, and resources, like network bandwidth and energy, are scarce and must be used carefully. With respect to the example of health status monitoring this could mean that not each single reading from the body temperature sensor is important, but only if the temperature is falling below 34 °C or getting higher than 40 °C. Therefore, it is important to support subscription languages to perform content-based filtering at the source. The event sources might be known by the subscriber a priori, at the time the subscription is made, or it might not be known at all. For example, a fireman wants to subscribe to a temperature sensor close to the fire. Consequently, it should be possible within the subscription to specify the event source by the name or ID of the source node, wild cards, like all sources or any source of a certain type, or by describing certain properties of the source node, like its location.

In a rescue operation, individuals from different organizations cooperate by sharing information. With respect to the current state-of-the-art it is reasonable to assume that each organization is using its own standardized vocabulary and data model. Therefore, it is also desirable to support subscriptions that are targeting different vocabularies. This support must also include the translation or mapping between different vocabularies since
information in rescue operations is shared also across different organizations. For example, the coordinators of firemen and policemen might be interested in the temperature in a certain room. They may use different terms for temperature even if the sensor supports only the firemen vocabulary.

Finally, it should be noted that there is no particular content-based language that is better suited than all others. With respect to the need of handling resources efficiently, simple languages are better than complex ones. However, some events might not be expressible with simple languages and therefore complex languages might also help to save resources by providing more sophisticated filtering. Separating choice of subscription language from how notifications are disseminated allows to always select the best language and the best dissemination protocols for the particular purpose.

In summary, a publish/subscribe system for emergency and rescue applications should fulfill the following requirements:

- content-based filtering at source nodes,
- flexible definition of source nodes in subscriptions,
- multiple vocabularies, and
- multiple languages.

In the following sections, we revise the state-of-the-art with respect to these requirements.

### 2.3 Related Work

Most publish/subscribe systems are tailored for a specific network and a specific subscription language. Often, the registration of potential publishers and their event subjects are done a priori or announced by advertisements at run-time. Of course, we must also decide a priori what kind of information on a node should be sharable, however this is not restricted to a well-defined topic hierarchy tailored for one application. Also, most of the related work in publish/subscribe systems for MANETs does not consider network partitions. The systems which do, are mainly not relying on brokers, and notifications are disseminated throughout the network using probabilistic broadcasting, gossiping or epidemic routing. As described in the previous section, many events will not be of interest to all nodes. Thus, replicating all events at every node and using content-based filtering at the nodes will in many cases not be efficient.

The STEAM system [7] is tailored for applications where subscribers only are interested in events happening in its proximity and thus network partitions are not considered. Subject and proximity filters are implemented on the producer or publisher side and contents filters on the consumer or subscriber side. Producers announce the type of event they may produce together with the geographical area this event carries information value. The middleware joins a proximity group of interest, i.e., for which it has either a subscription or an announcement, once the host machine is within the associated geographical scope. The work is based on the assumption that the closer subscribers are located to a publisher, the more likely they are to be interested in its events. This assumption may not be valid in a rescue operation where rescue personnel in charge may be interested in events happening in the whole area. The publish/subscribe service presented in [5] builds multicast trees for each publisher. The multicast tree is built in a distributed manner using only local information. But, as the authors mention, this does not fit a highly mobile environment, the algorithm is more suitable for mobility scenarios where a node moves around occasionally, and settles for a period before moving again. A topic based publish/subscribe without mediators, is described in [3]. The topics are arranged in a hierarchy. The system is completely decentralized, and assumes no underlying routing protocol. Subscriptions are sent to neighbors using beacons, and events received are sent to neighbors based on this subscription knowledge. This system assumes that the topic hierarchy is known by all nodes. In [4], subscriptions are propagated only to the mediators close to the subscriber. The subscription language is content-based and the event is routed along the link a matching subscription was received from. Thus content-based routing. A content-based publish/subscribe system for MANETs which integrates an extended ODMRP (On-Demand Multicast Routing Protocol) to disseminate notifications is proposed in [16].

Some related work is looking into how to make a publish/subscribe service which is configurable and can support different networks and subscription languages. GREEN [12] is a configurable and re-configurable publish-subscribe middleware which can be used for different network types and not only fixed infrastructure based networks or only MANETs. Also different subscription languages can be used by GREEN. The middleware has two layers; an interaction framework which provides what they call interaction types, e.g. content-based or topic-based and an overlay framework which can be configured for the network. They have implemented an event broker overlay for MANET using subject and proximity filtering at the publisher side and content filter at the subscriber side similar to STEAM. We are not looking into all kinds of network types from WAN to MANETs, but rather looking into different MANET scenarios so GREEN is a more general solution. However, our DENS also supports the process of locating publishers without the need of the publisher sending out announcements in a specific language or only using subject filters at the publisher nodes as in STEAM. YANCEES [11] is a versatile publish/subscribe infrastructure designed to be programmable, configurable and extensible at run time. It uses a combination of plug-ins, extensible languages and open implementation allowing the replacement of the event dispatcher with different event routing strategies. Different event dispatchers can be used simultaneously. YANCEES provides a protocol form of interaction besides the common publish/subscribe extensions of existing events, subscription and notification models. If a new protocol, publish, or subscription commands are specified, plug-ins are implemented and specific dynamic XML parsers bind these new protocols or commands to their plug-ins. Parsers are responsible for reading publish and subscribe commands and converting them to the installed native event dispatcher formats at runtime, as well. In the current implementation, XML schema provides the extensibility mechanism used to define the subscription, notification and protocol languages. Another approach for supporting different subscription languages is described in [6]. They suggest a solution for avoiding that subscribers have to send different subscription stating the same interests to a number of services using different filter languages with different expressiveness that will cause repeated subscriptions in the system. As mentioned in the Introduction they address these problems designing a Meta Event Notification Service that support heterogeneity of services and providers. Each filter defined at a meta-service is transformed into a filter expressed in the language of each event notification source. Such meta-schema for exchanging event filter definitions and event
information, as well as recognition of duplicates at the Meta level, will introduce overhead in MANETs.

The related work does not fulfil all our requirements as they do not support network partitions, decoupling of subscribers and publishers, multiple vocabularies and multiple languages at runtime.

3. Architecture
In this section, we briefly introduce the two components of the Ad-Hoc InfoWare middleware that are important for multiple subscription language support, i.e., DENS and Knowledge Manager, before we describe the multi subscription language support in detail.

3.1 DENS Architecture and Protocols
Information sharing is mission critical for emergency and rescue scenarios. Therefore, those services that support information sharing must be designed and implemented in such a way that the services achieve high availability and graceful degradation in case of disconnections and network partitions. We address this requirement in the DENS by using several cooperating DENS nodes that act as mediators between subscribers and publishers. These nodes then form 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. If there is no possibility to reach the destination due to network partition or the destination device is turned off, the DENS overlay stores the subscription or notification and tries later, i.e., performs store-carry-forward operation. Furthermore, subscriptions and undelivered notifications are replicated among the DENS nodes to increase service availability and enable all DENS nodes to gracefully degrade their service if they are partitioned from the network. This means that the service is available even though some of the DENS nodes are temporarily turned off or out of communication range, and hence not a single point of failure. In each partition, the DENS nodes can directly service all subscribers and publishers in their partition. Since the networks in emergency and rescue operations could be very dynamic, network partitions and mergings might happen frequently. This typically leads to inconsistencies between DENS nodes from different partitions with respect to the stored subscriptions and undelivered notifications. It should be noted that there is an inherent trade-off between reliability, i.e., delivery ratio, and availability on the one hand and overhead in consistency management on the other hand.

The more DENS nodes exist, the higher is the reliability and availability, and the higher are the costs of consistency management. In general, more powerful nodes are selected to run the DENS service, but the DENS service is configurable such that it could also run in a minimal configuration on weaker nodes. There are no assumptions about subscriber and publisher nodes, except that publisher nodes must be able to process the filters that specify the events of interest.

Figure 1 shows the architecture of DENS nodes. There are three classes of delivery protocols:

- **SUB-DENS** is used for communication between subscriber and DENS, i.e., subscriptions are sent from the subscriber to the DENS and notifications are sent from the DENS to the subscriber.
- **PUB-DENS** 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.
- **DENS-DENS** is used for communication among the DENS nodes, including replication of subscriptions and undelivered notifications, consistency management etc. In order to identify the destination of subscriptions, subscription language specific look up functions are used. The destination of notifications is identified by calling a language specific matching function (see Section 3.3).

Each DENS node needs in minimum to support these protocols. When an application or a middleware service wants to subscribe to an event, it sends a subscription request to the DENS overlay. A timeout is set, and if it does not receive an acknowledgement before the timer expires, the subscriber will resend the subscription. Notifications filtered at the publisher nodes are sent to the DENS overlay with a unique ID. Then the notifications are matched with the subscriptions to find which subscribers are interested in it. Subscription information is replicated via the DENS-DENS protocol. This means that a publisher may send a notification to any DENS node, for instance the closest. The nodes in the network collect information about DENS nodes in its proximity by listening to beacons sent regularly by the DENS nodes. These beacons are broadcasted locally. The beacons are also used for initiating synchronization of information between DENS nodes. When a DENS node receives a DENS beacon from another DENS node, they may exchange information about notifications, known subscribers to the notifications, and a flag telling whether this notification has been successfully delivered or not, in addition to subscription information. The protocols are independent of the underlying routing protocol. Different scenarios with respect to node density and the degree of mobility, and also the application scenario, few subscribers, many subscribers, long-lasting subscriptions or not, etc., requires different approaches to achieve a high delivery ratio and low resource consumption. In mobility scenarios with many network partitions, the DENS nodes need to more actively exchange information to spread the notifications to the subscribers in the different partitions. For this reason we have different variants of these protocols in different situations.
The availability and scaling component is used to ensure that there is a sufficient number of DENS nodes in each network partition. The storage component is used for storing subscriptions and undelivered notifications. Since network partitioning may occur or subscriber nodes may be temporarily turned off to save power, all notifications may not be delivered at once. Consistency issues are managed by the State Management Component. A more thorough description of the DENS architecture can be found in [14].

### 3.2 Knowledge Manager

Two important requirements for publish/subscribe systems for emergency and rescue operations are that subscribers do not need to know the publisher and each organization should be able to use its own vocabulary. These two requirements are addressed by the Knowledge Manager (KM) [10]. The main objective of this component is to manage sharing and integration of available information in the network. 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. A resource can be any information item that is to be shared, like for instance data items, sensor profiles, vocabularies and ontologies.

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 contains simple one to one mappings of concept terms between the vocabularies. The Data Dictionary Manager is the main metadata handling component within the KM, responsible for sharing and update of metadata. It maintains a Local Data Dictionary and a Semantic Linked Distributed Data Dictionary (SDDD). Metadata descriptions of local resources are enhanced with concepts from vocabularies, and registered for sharing in the Local Data Dictionary. The concept terms are extracted to the SDDD, which is responsible for linking the concepts to where the full metadata descriptions can be found and distributing this information, thus creating a view of shared information in the network.

To explain how this is achieved, we give an example of a sensor node registering to share information about measured body temperature of a patient. The nodeID of this sensor is ‘nod1’. At startup, some initial information is registered to its data dictionaries, e.g., data structures and vocabularies. The node starts logging temperature measurements in a file named ‘patient.dta’. The file is registered with metadata description as a resource for sharing in the Local Data Dictionary, and relevant concept terms and metadata resource ID (metaResID) is extracted to the SDDD. We use two levels of links to differentiate between pointers to local items (link level 0) and to metadata resources on another node (link level 1). Availability of the metadata resource is shown as 0 or 1. Below, we show example contents of the Local Data Dictionary and the SDDD before registering with any neighbors (see Table 1 and Table 2).

<table>
<thead>
<tr>
<th>Table 1: Local data Dictionary Node 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>metaRes ID</td>
</tr>
<tr>
<td>mID2</td>
</tr>
</tbody>
</table>

When ‘nod1’ encounters another node (‘nod2’), they will exchange SDDD contents. Below, we show SDDD example contents for ‘nod2’ after merge. If ‘nod2’ later gets a request about temperature-related data, it can make a lookup in its SDDD and find that ‘nod1’ contains such information. The node ‘nod1’ can then be requested for the actual temperature measurements. This is illustrated in Table 3. Periodical clean-ups are performed to remove outdated links and information in the dictionaries.

<table>
<thead>
<tr>
<th>Table 3: Semantic Linked Data Dictionary Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept Term</td>
</tr>
<tr>
<td>temp</td>
</tr>
</tbody>
</table>

### 3.3 Multiple Language Support

The developer of a new subscription language and the KM enables the DENS to resolve the conflict of supporting multiple subscription languages and decoupling subscribers and publishers at the same time. The KM is used for locating publishers. When receiving a subscription, the DENS needs to look-up the publisher. This request to (1) parse the subscription to extract the attributes that are used to form the filter and (2) call the look-up function that retrieves the sources for these attributes, i.e., the publishers, from the KM. When receiving a notification, the DENS needs to identify the subscriber(s). This is done by matching the incoming notification with stored subscriptions. Since this matching is language specific it must be provided by the language developer as a plug-in. Finally, a publisher needs to understand the subscription to filter the events of interest. The language specific filtering has also to be provided by the language developer as a plug-in. Subscription language IDs (SL-IDs) are used by the DENS and the publisher to identify which particular plug-in has to be invoked when a subscription or notification arrives. In addition to SL-IDs, we use also Vocabulary IDs (V-IDs) that uniquely identify the particular vocabulary. The generic structure of subscriptions and notifications is illustrated in Figure 2.

<table>
<thead>
<tr>
<th>Table 2: Semantic Linked Data Dictionary Node 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept Term</td>
</tr>
<tr>
<td>temp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 2: Semantic Linked Data Dictionary Node 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>concept Term</td>
</tr>
<tr>
<td>temp</td>
</tr>
</tbody>
</table>

#### Figure 2: Generic structure of subscriptions and notifications

**Look-up:**

According to the requirements discussed in Section 2.2, subscribers have three options to specify the publisher. First, the node id of the publisher can be included in the subscription. Second, subscribers could use a wildcard, like all or any, or stating the number of nodes that should be used as publishers. In
this case, a language specific look-up function is needed to query the parser for finding the publisher(s). This function is a parser which knows the syntax of the language and which returns all the attributes in the filter. Nodes that are willing to share exactly these attributes are potential publishers. Since the KM manages metadata, including which node is willing to share which information, a query to the KM with the extracted attributes can return the node IDs of the publishers. Subscribers and publishers might use different vocabularies, even if they refer to information that is semantically equivalent. Therefore, it is also necessary to use all synonyms in the query to the KM. The KM therefore offers two functions to support the DENS in finding the correct publishers. We make the assumption that all attributes used in a subscription are concept terms from a known vocabulary. The function for mapping attributes between vocabularies is findSynonym(). It takes a vocabulary ID and the set of attributes from the subscription as parameters, and returns a set of vocabulary IDs and a list of synonym attributes for each vocabulary. To do this, the KM performs a thesaurus look-up for each of the parameter attributes for the given vocabulary ID. The second function, findSource(), finds the correct publisher nodes across vocabularies. This function takes a vocabulary ID and list of attributes, and returns a set of node IDs of nodes that will understand these attributes. The KM requests the Data Dictionary Manager for a concept search in the Semantic Linked Distributed Data Dictionary, finding all nodes associated/linked to the given vocabulary and attributes. The third possibility to specify publishers in a subscription is to describe certain properties of the publisher, like location or other context information. These properties are described with the same subscription language and vocabulary as the filter, and further restrict the set of potential publishers. Since the context information of nodes is data, not metadata, a single request to the KM is not sufficient to retrieve the publisher ID. Additional request to nodes that store this data must be made. Efficient solutions for this type of addressing are subject to ongoing research.

Filtering:
The filtering of events is done by a monitoring agent, called watchdog (WD), running on the publisher node (see Figure 3). The WD Manager implements the interface between the DENS and the WDs. The WD Manager interfaces DENS to obtain information about the data and content type which should be monitored for the received subscription. The WD is extensible for different filtering plug-ins providing abstract methods on how to handle the subscription and unsubscription requests, how to process the format of subscriptions, how to filter the events and how to generate notifications according to the implemented subscription language. When the DENS sends a subscription, the WD Manager registers it and checks to see whether the same subscription has been received before, and updates the record store on subscribers currently interested in events at this node. The WD Manager starts the WD Execution Environment with the correct filtering specific for the subscription language. The WD Execution Environment monitors data acquisition applications and processing of data according to their content type. When an event occurs that fulfills some condition specified in some subscription, the WD Manager generates notifications according to the specific subscription language and sends the notification to the DENS.

Figure 3: WD Execution Environment

Matching:
DENS nodes store all subscriptions and the subscriber IDs to be able to identify the destination for notifications. Notifications are sent with the PUB-DENS protocol to a DENS node. With the help of the SL-ID the DENS node calls the proper matching function. The matching function is matching the notification with the stored subscriptions and returns the subscriber ID(s). To support multiple vocabularies, the findSynonym function is used.

4. IMPLEMENTATION
As a proof-of-concept, we have implemented two different subscription languages: a simple attribute/value-based language, and a more complex subscription language using ideas from fuzzy logic to express uncertainties. We have tested these languages together with the DENS protocols. In these tests, we used a simplified implementation of the KM, i.e., a simple data structure for finding the publisher ID and a simplified Data Dictionary for providing the necessary metadata. As emulation platform we are using NEMAN [9]. NEMAN provides a virtual wireless network that can handle hundreds of nodes on a single machine. The advantage of an emulator to a simulator is that it allows real code to be run on the higher layers. The processes running on the virtual nodes bind to virtual network (TAP) interfaces. In this section, we describe the two subscription languages, a WD implementation and the Test Setup. The efficiency of the filtering and matching algorithms for the two languages is not the main focus of this article so the results are only briefly discussed. The performance of the DENS protocols is documented in [13].

4.1 Simple Attribute Value-Based Language
In this language, an event is a conjunction of pairs, each consisting of an attribute and a value. A subscription defines the user’s interest and contains a set of predicates combined with the Boolean operators AND, OR, and NOT in any combination. Each predicate is a triple of the form attribute-relational operator-value. The relational operators are =, !=, >=, etc. A subscription expresses a constraint over a cartesian domain of values and can be used as a filter over such values. An example is “Sub-1: Temperature tmp:: tmp>40 OR tmp<34”. A corresponding notification might be “(tmp, 41)”. Doing filtering and matching is thus a simple and not very costly operation. At the publisher node the WD summarizes and groups the different subscriptions, e.g. “[Temperature tmp:: (Sub-1: tmp>40) AND (Sub-2: tmp<39)], [Temperature tmp:: (Sub-1: tmp>34)]” means looking for events where tmp > 39 and events where tmp<34.

The matching and filtering algorithms in publish/subscribe systems can use indexes of predicates or not. If indexes are not
used, several evaluations per attribute are performed, i.e., the predicates are tested independently. All predicates of subscriptions involving a certain attribute are tested only once if index structures are used, and using indexes therefore results in faster filtering. However, storage of subscriptions is always required to support unsubscribing. Therefore, we are using the non-indexing approach which is best with regards to space efficiency.

4.2 Fuzzy-Concept-Based Language

Sometimes a subscriber may not be able to express its interest in exact terms. For attributes such as “road access”, terms like “clear”, “totally blocked”, etc. would be necessary. A matching between a subscription and a notification is therefore not necessarily true or false, but can be stated by the degree of matching, e.g., 87% match. For the rescue operation scenario a flexible subscription language could be particularly useful. In a hectic environment there may not be time to always give crisp values or crisp value ranges for a subscription predicate. Selecting terms such as “warm”, “high” etc. may be more valuable, and one may also allow different interpretations in different contexts. The most common way to handle uncertainty is to bypass it by replacing approximate values by crisp values, but then the subscription does not necessarily reflect the real interest of the subscriber. An approximate subscription contains a set of approximate predicates combined by Boolean operators. In the approximate predicate “t \text{op} A”, t is an attribute name, \text{op} is an operator, and A is a fuzzy set over the domain of t. The operator can be non-modifying, like is, is not, or modify the fuzzy set, like the qualitative operator is more or less or the temporal restriction is lately, as in “last few seconds”. An example of a subscription in this language is “Sub-3: Temperature tmp: tmp is more or less mild”. A priori defined parameters govern how each predicate is interpreted, in particular, how each operator modifies the different fuzzy sets. If for instance the fuzzy set “mild” covers the temperature range 8-14°C and the value of tmp is 7.5°C, different parameters for is more or less might result in different notifications like “(tmp, 7.5, mild: 93%)” and “(tmp, 7.5, mild: 82%)”, or even no notification at all if the match falls under a predefined threshold.

We have implemented and tested two matching and filtering algorithms, one using predicate indexing and one without indexing. Using predicate indexing at the DENS nodes reduces the number of subscriptions that must be examined. In the indexing-matching algorithm, the predicates are first matched, and then matching subscriptions are identified and evaluated by their overall scores. The overall score aggregates all subscription predicates whose similarity measure is greater than the threshold associated with them. In the non-indexing matching algorithm, we first compute the membership grades which tell how close the values of the notification and the constraints of the subscription predicates are according to their fuzzy set definitions. Then aggregating operators can be chosen to compute the overall degree of matching for each subscription. The notification is relevant if the score is greater than the subscription’s relevance threshold. We improve this algorithm by assigning features to each subscription. The features state which event should be monitored and which attributes of that event the subscription is interested in. One subscription may e.g. be interested in only the temperature values, another in both temperature and wind speed. We can therefore optimize the matching process by checking first the subscription and notification features for potential hits. Only in case of relevant features, approximate matching of notification values to constraints in predicates is performed in the second step.

4.3 WD Implementation

We have developed a light-weight portable MIDP Java WD application which implements a protocol handler interface for monitoring files on a local mobile file system (JSR75 API). The Java WD has been emulated in the Sun Java Wireless Toolkit 2.3. On the publisher we have implemented a filtering algorithm for the fuzzy concept-based language. Events in the system are monitored and categorized. The WD filters the events according to the active subscriptions and generates a notification. We optimize by aggregating active subscriptions by their covering as in [2] and/or sorting them according to their features. If two subscriptions SA1 and SA2 are considered, SA1 covers SA2 if and only if all the notifications that match SA2 also match SA1. If two notifications NA1 and NA2 match SA1 and SA2 respectively, where SA1 covers SA2, then NA1 covers NA2. Thus, many subscriptions and notifications can be evaluated using single matching. If subscriptions are sorted according to their features, the event-filtering algorithm will stop earlier rather than evaluate all subscriptions.

4.4 Test Setup

The test setup is described in Figure 4. On one machine (Dual Xeon 2.8 GHz running Linux 2.6.10) NEMAN is running and emulating a MANET. 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, relative pressure etc., and these sensors send values to a base station which logs them.

![Figure 4: Test Setup](image-url)
subscriptions were placed in a persistent record store on the mobile device, and 10-20 ms if the subscriptions were cached. If subscription covering was done on the publisher, the loading time for 100 subscriptions was 140 ms and the filtering time was 6.5-7 ms. When a notification was sent to the DENS overlay in NEMAN the time to do the matching of 100 subscriptions with 3 attributes and one notification was 10 ms for the fuzzy subscription language, and negligible for the simple attribute-valued one. If predicates of subscriptions are cached and indexed, the loading time was 270 ms and the matching time was 6.5 ms. For more complex aggregation functions, the matching time was 40-90 ms. The best matching algorithm uses sorted features and exploits covered subscriptions. Its matching time was 2.6 ms and loading time was 29 ms.

5. CONCLUSION

This paper presents a distributed event notification service for (DENS) for sparse MANETs which supports concurrently multiple content-based subscription languages, including complex languages such as subscriptions expressing uncertainties using fuzzy logic. Furthermore, the DENS decouples subscribers from publishers, thus they do not need to know about each other. In order to handle different subscription languages and transfer subscriptions and notifications to the right destinations, developers of subscription languages have to provide three language specific plug-ins: look-up, filter, and matching. Each subscription language identifier enables the DENS to call for each incoming subscription and notification the proper plug-in. In addition to multiple subscription language support, multiple vocabularies are supported for the subscription languages. We use a monitoring agent, a watchdog, to execute the filter in the Watchdog Execution Environment and to filter events at the source based on a summary of the relevant content-based subscriptions. The subscriptions are stored by the DENS overlay and when a notification is sent to the DENS overlay it checks a subscription language ID to call the right matching function. Since the dissemination process is independent of the subscription language, different applications may specify and use different languages, as long as the watchdog supports the language and the DENS overlay has the correct matching plug-in and look-up service for finding the correct publishers.

The design of the DENS is strongly influenced by the requirements of emergency and rescue applications, but we believe it can also be useful for other applications domains. We have implemented two subscription languages and tested them with the DENS protocols. First evaluation results indicate that matching and filtering of approximate content-based subscriptions can be performed rather efficiently. Open questions that are currently investigated include the matching process at the DENS overlay. Improved matching algorithms require more subscription loading time because subscriptions need to be covered or sorted. Depending on system partitions and how often the subscribing and unsubscribing are performed, a tradeoff between matching and loading time need to be evaluated. Besides more in depth performance studies, our ongoing and future research is also concerned with the development of efficient solutions for look-up in case of property descriptions of the publisher, like location. Furthermore, we target a full integration of the DENS and the Knowledge Manager.

6. REFERENCES