A Protocol for Event Distribution in Next-Generation Dynamic Networks

Carlos Pinho\(^1\), José Ruela\(^1\), Kostas Pentikousis\(^2\), and Cornelia Kappler\(^3\)

\(^1\) INESC Porto, Faculdade de Engenharia, Universidade do Porto, 4200-465, Portugal
\(^2\) VTT Technical Research Centre of Finland, Oulu, FI-905671, Finland
\(^3\) deZem GmbH, Berlin, 10623, Germany

With the increasing number of devices carried by users and the emergence of more dynamic network scenarios, new solutions have to be developed to support ubiquitous communications. The need for dynamic and transparent auto-configuration and adaptation of applications, devices and networks of devices to a wide variety of events is one of the major challenges. This paper presents a protocol to be used with notification systems, to support the registration, subscription and distribution of generic events among heterogeneous and independent entities, thus addressing the requirements of future communication scenarios.

Index Terms—Event Notification, Triggering, Next Generation Networks, Dynamic Networks, Self-Adaptation, Signaling.

I. INTRODUCTION

Users carry and interact with an increasing number of (mobile) devices with wired and wireless communication capabilities that enable them to be constantly in contact and with access to a wide variety of resources. Such requirements for ubiquitous communications are the main driving force behind the need for establishing relations and forming dynamic networks with those devices. However, the regularly changing environment, due to mobility, intermittent connectivity, access availability, and so on, burdens users with the need to perform network and device (re-)configuration tasks. Examples of these needs typically emerge in ad-hoc and mobile networks, cooperating devices and networks, and in other distributed communication environments.

The need for dynamic and transparent auto-configuration and adaptation of both standalone devices and networks of devices to a wide variety of events plays and will continue to play a major role in networking. In particular, the in-time distribution of relevant events or notifications between related entities is the key functionality that enables devices and networks to synchronize and react to changes.

Current protocols for supporting such dynamic behavior are typically associated with mobility [1][2][3]. From the services perspective, solutions for dynamic networked components, applications and services that scale from the device to the enterprise level have also been proposed [4]. Other solutions use a different paradigm to distribute specific types of information through the Internet, supported by the Session Description Protocol (SDP), such as, for example, multimedia session descriptions, to particular consumers [5].

In this paper, we argue for the need of new mechanisms that support dynamic and transparent auto-configuration and are capable of distributing generic information, from the applications or node/network management perspective, in a reliable way and addressing both online and off-line scenarios as well as local and remote communication. We present a new Inter-Function communication Protocol (IFP), which supports the registration and subscription of entities (as producers and consumers of events, respectively) as well as the distribution of generic events among interested entities.

This paper is organized as follows. The background of this work and the rationale for proposing IFP are presented in the following section. Section III introduces IFP and Section IV presents the protocol specification. The analysis of the use of IFP in the different scenarios it was designed for is made in Section V. Finally, conclusions are drawn and future work is outlined in Section VI.

II. MOTIVATION AND RELATED WORK

Applications, devices and networks need to be able to take into account the dynamic nature of current and future communication scenarios, not only required by the increased mobility but also by new types of networks, such as Personal Area Networks and spontaneous networks, as well as new user requirements. Consider the following scenario. A user carries his laptop and a smartphone in a train. Both devices are connected to the WLAN access network of the train and are associated with the notification system of the user’s office, which enables the user devices and applications to send and receive event notifications. The user is primarily using his laptop, which has a VPN connection to the office network, in order to run a distributed collaboration application. After a while the laptop runs out of battery and its energy management application sends that warning to the notification system at the office, which takes care of distributing this event to all interested and/or affected entities, such as, for example, other instances of the collaborating application and the user smartphone in order for it to assume the control of the collaborative application. Upon receiving such notification, the smartphone takes care of establishing the VPN connection, if
not previously established, and starts the collaborative application. After having concluded these tasks, the collaborative application of the smartphone sends an event to the notification system that distributes it to the other instances of the application informing them that the migration was successfully concluded. The user can then close the laptop and switch to the smartphone to continue his work.

When leaving the train, the access network management application running on the smartphone senses that the WLAN connection with the train network is degrading. It issues a notification about this situation to alert other instances of the collaborative application that it may go “off-line” and receives a trigger from the collaborative management application to turn on the 3G interface in order to avoid further disconnections. It then turns on the 3G interface, establishes the VPN and notifies other partners it is online again. The user continues his work until arriving at the office and possibly changing to another device.

Although there are today individual solutions that can materialize the scenario described above, we argue that a more generic framework is needed and explain how the solution proposed in this paper addresses this and other scenarios and tries to overcome the limitations of current solutions.

Dynamism in what concerns mobility is an issue being studied in several IETF working groups in a transparent self-configuring and self-maintained way. The MANET [1] and NEMO [2] working groups, for example, address mobile ad-hoc scenarios following particular approaches. While MANET adopts periodic broadcast and active polling mechanisms to maintain routing tables, NEMO uses a mobile router as a gateway to the outside that handles mobility transparently to the rest of the mobile nodes.

The IETF MIPSHOP working group [3] is developing a protocol to support the services of the emerging IEEE 802.21 media-independent handover standard [6] that enables optimal network selection across multiple different network access technologies. Nevertheless, all the solutions proposed by [1][2][3] focus on mobility and handover management, leaving other types of requirements unaddressed, such as the adaptation of applications.

From the perspective of the interaction between applications, dynamism can be addressed using Remote Procedure Calls (RPC) [7], Web Services [8] and Jini-based services [4]. All these can be employed in a custom-made, specific solution for the scenario described above. For example, RPC, based on a client-server model, enables an application to issue a request to another one typically running in a different machine. RPC is often used in distributed computing. However, due to the variants of different implementations, there are several incompatible RPC protocols. Another drawback is related with its functional principles: while the server is processing the call, the client is blocked.

Web services support interoperable machine-to-machine interactions over the Internet. They are able to locate and bind to other components at run time. These characteristics promote the composition of services, i.e. building new services based on existing ones, both within and between organizational boundaries. In spite of their platform and location independency and just-in-time integration, the web services heterogeneity can pose serious concerns to interoperability, namely due to the existence of different standards.

Jini [4] is a platform based on Java Remote Method Invocation. It is capable of delivering access to services over any network for any operating system. In spite of its interesting features in “single owner” scenarios, Jini introduces some problems in other contexts. For example, components have to know about each other during development; this requires tight levels of administration and changes at the stub or skeleton imply additional modifications.

Related with information delivery in dynamic networks, the IETF MMUSIC working group is developing a protocol to deliver Internet Media Guides (IMGs) [5]. IMGs are a structured collection of multimedia session descriptions (e.g. television programme schedules). They can be delivered to a set of known or unknown hosts by the IMG owner or, alternatively, the hosts can request a specific collection of IMG. Since it is being specified for the distribution of multimedia metadata, this protocol is not generic.

The analysis of the current state-of-the-art puts in evidence that current protocols that address dynamic network environments are very fragmented and focused on the transport of very specific information. Furthermore, some of them raise some serious concerns regarding interoperability in broad dynamic scenarios.

The framework developed around the trigger management mechanisms proposed in [9] can handle a variety of triggers originating from any protocol layer and any mobility management entity within the network. It aims at operating in a richer mobility management framework, and enables the deployment of new applications and services. Nevertheless, due to its origins in mobility management it is simply not generic enough for all possible uses. IFP, the protocol proposed in this paper, supports enhanced communication between entities for event notifications and borrows some ideas from this earlier work. This newly proposed event distribution protocol can be used in a wide variety of scenarios for transporting generic event notifications among heterogeneous and independent entities addressing the needs for dynamic and transparent configuration and adaptation. The following section introduces IFP, briefly summarizing its origin in Ambient Networks and presenting its operational principles.

III. The Inter-Function Protocol

The Inter-Function Protocol (IFP) enables the query, registration, subscription and distribution of notifications or events among distributed entities. Thus, due to the distributed nature of entities and their possible mobility, this protocol is not designed only for intra or inter-domain utilization since other
mixed cases need to be supported.

IFP is designed to work in scenarios involving three types of entities: producers of events (i.e., entities that generate events); event consumers (entities that receive and use or benefit from event notifications); and storage entities (that store information about producers and consumers as well as the mapping between events available and consumers' interests).

A. IFP and Ambient Networks

IFP was developed within the two-phase, four year WW1-Ambient Networks project [11], funded by the IST programme of the European Commission. This project aimed at mitigating existing obstacles to dynamic inter-working and developed a solution for the automatic cooperation between networks. Two of the fundamental concepts introduced by the project are Ambient Networks (ANs) and Network Composition.

ANs are autonomous and dynamic networks. They are capable of cooperating with each other and to compete and pursue specific goals defined by their users or administrators (e.g., through profiles), their policies and applications’ requirements. At the heart of an AN is its Ambient Control Space (ACS). It comprises a set of modules called Functional Entities (FE) that perform specific related tasks (e.g., Composition-FE, QoS-FE, etc.). These FEs can be introduced or removed in a modular way and can be implemented in a centralized or distributed manner. It is important to mention that an FE is not tied to a particular implementation. The emphasis when defining a particular FE is put on what each FE should do and what interfaces will it expose to other entities, not the implementation specifics. The Ambient Networks project has, as one might expect, implemented several of the FEs (see for example [9][10]) and integrated them into a single prototype. This prototype has to be seen only as one possible implementation instantiation.

The concept of Network Composition was introduced to deal with the cooperation between ACSs of different ANs going from simple inter-working to full merging of functionalities and physical resources.

IFP can be used in ANs to support the distribution of event notifications, such as “network composition agreement has been established/deleted” both for intra-ACS and inter-ACS communication between FEs. In the latter case, some type of Network Composition must have been established between them. For example, if AN-A and AN-B have composed already, and then subsequently AN-B composes with AN-C, FEs in AN-A can receive notifications related with this new composition. Without IFP, this form of dynamic communication is not possible.

IFP is based on the main protocol used for FEs and AN communications, the Generic Ambient Networks Signaling protocol (GANS) [12] and is built as a GANS Service Layer Protocol (GSLP). GANS in turn is based on Next Steps in Signaling (NSIS) protocol [13] and it has two layers as well: the Service Layer that transports the signaling messages, and the Transport Layer that sets up and maintains the signaling overlay through the network nodes.

The main difference between GANS and NSIS is that with GANS the sender does not have to know the IP address of the destination but only its “role” or symbolic name (e.g., “HomeAgent@network_id3.org”). This type of addressing is supported by the Destination Endpoint Protocol (DEEP) [14], which is a generic name resolution protocol for heterogeneous environments that takes care of resolving a symbolic name into the corresponding IP address before the message leaves the sender.

In spite of having been specified in the Ambient Networks project, IFP is not limited to that framework neither to scenarios involving ANs only. In fact, IFP and the associated notification system can be used in a variety of scenarios involving heterogeneous entities and for different purposes.

B. IFP Operational Principles

IFP can be used either when a so-called Inter-Function Events and Notifications entity (IFEN) is adopted or when a simple database/repository is used to store events, along with producer and subscriber information; the latter is called Events Database (EDB) in the remaining of the article. IFEN is an entity that evolved from the framework presented in [9], and is responsible for all tasks associated with the registration, subscription, management and distribution of events or notifications among entities with a specific relation. The new functionalities added to IFEN include the validation and authorization of entities and events, query of events and search for registered events based on keywords. Registrations and subscriptions can be stored either at IFEN or at EDB. However, adopting IFEN or EDB is more than an implementation decision. In fact, as analyzed in Section V, choosing IFEN or EDB should not be decided simply based on the desired complexity or scenario targeted for the notification system, since it may have a significant impact on the number of messages exchanged in the system.

As illustrated in Fig. 1 and 2, producers are the first entities to register the events they produce with IFEN or EDB. After events are available, consumers subscribe to receive notifications based on event occurrences.

During the registration or subscription of an event, the producer/subscriber of the event has to supply its own identification and specific security information (e.g., certificate and/or network membership certificate) that enables the verification that the entity is truly who it says it is, and that it has permission to register/subscribe to that event, the ID(s) of the event(s) it produces or wants to subscribe to as well as the type of event(s). The IFEN or EDB are responsible for validating and authorizing the registrations or subscriptions as well as the identifier of the event (“event ID” – note that these identifiers have to be standardized in some way, but this
is still an open issue) and update either the list of available events and corresponding producers, in case of registrations, or the list of subscribers, in case of subscriptions. These entities can also be configured to inform other registered and subscribed entities about any new event that is registered, so that they can subscribe to it, if they are interested.

As depicted in Fig. 1, when IFEN is present, producers send the event notifications to it. IFEN takes care of prioritizing events, determines which consumer(s) is interested in that type of event, filters it according to the filtering preferences specified by the consumer and sends it to all eligible consumers.

When IFEN is not used (Fig. 2), the producer sends its event message normally to GANS lower layers (like when IFEN is present), which have the responsibility of discovering that IFEN is not present. In this case, DEEP has to be involved in order to resolve the received “Event ID” into the corresponding symbolic names of the entities that subscribed to that event [15]. Thus, a “DEEP Lookup” message has to be sent by DEEP to EDB whenever an event has to be sent (case of Fig. 2) in order to receive the list of subscribing entities. Note that the “Lookup” message is a DEEP message rather than an IFP message. This message is included in the diagram for only to ease understanding of IFP operation when IFEN is not involved. After receiving the list of subscribers, the producer can send the event notifications to the consumers. If the producer does not have the IP addresses of the returned subscribers, DEEP will be invoked again, this time to solve the symbolic names of the entities into their addresses.

It should be highlighted that when EDB is used, the filtering conditions are not verified by the database, since this is not its purpose. Hence, the natural solution is to let the consumers perform their own filtering because DEEP is only used to resolve names such as an event ID into the event consumers’ address. DEEP has nothing to do with event filtering rules.

IV. PROTOCOL SPECIFICATION

IFP supports a set of messages exchanged among the different entities: producers, consumers and IFEN/EDB. This section describes the messages and the information they transport in more detail in order to illustrate its purpose and scope.

A. Signaling and Information Exchange

In order to be able to send events to potential interested entities, the producer has to register the events it generates either with IFEN or EDB, depending on the implementation adopted. Fig. 1 illustrates the Event Registration sequence.

First, the producer issues a “RegisterEvent” message (message 1.) comprising the Event IDs it produces and the type of those events (e.g., Boolean, integer, etc.). A description of each registered event is also included, to facilitate semantic searches of events. Depending on the adopted security scheme, a certificate can also be included to enable the receiver to validate the producer (see Table I). Optionally, the “registration timeout” parameter may be sent in order to enable the receiver to remove the events from that producer when that time interval expires, if not renewed. If not provided, IFEN or EDB will adopt their default timeout parameter. When receiving this message, the entity responsible for storing and managing the event registrations (IFEN or EDB), validates the producer as well as its Event IDs (message 2.), authorizes (3.) and if everything is correct, stores the registration (4.) and sends a confirmation message to the producer (5.). If something went wrong the producer will receive an error message (6.).

Event Subscription, illustrated in Fig. 4, is very similar to Event Registration. Also for subscribing, the entities must provide the event IDs they are interested in. If they do not know which events are available, they can query IFEN or EDB about the events associated with specific keywords or ask for all events available (message 1.).

Before providing the requested information, IFEN or EDB validates the sender (message 2.) and authorizes the request (3.). This validation and authorization procedures may depend on the level of trust and security implemented.

If validation and authorization conclude successfully, the IFEN/EDB searches for the requested information (4.) and
returns the events available according to the conditions requested by the consumer (5.). It is up to the consumer to analyze the events available (6.) and decide if it will subscribe to any of them (7.). When subscribing, it can specify a set of filtering rules associated with each event subscribed. Only events that match those conditions shall be sent to the subscriber.

As in the case of event registration, a subscription timeout exists also based on the same principles. Depending on the security level a certificate can also be included to enable the receiver to validate the subscriber (8.) and authorize it (9.). If the consumer and its events are successfully validated and authorized, the subscription is stored (10.) and a confirmation message is sent towards the consumer (11.). If validation or authorization fails, the consumer receives an error message (12.) and the interaction ends.

Depending on whether IFEN or EDB are used, different procedures are involved in the distribution of events. When using EDB, the producer, after knowing which the subscribers are, sends the event notification directly to them (see Fig. 2). The case where IFEN is used to support the distribution of events is illustrated in Fig. 5. When employing IFEN for events distribution, producers send the notifications to it (message 1.). IFEN then takes care of validating the entity and event (2.) authorize it (3.) as well as prioritizing events, identifying the subscribers, check their filtering rules (4.) and finally sending the event notifications to the appropriate subscribers (5.).

Depending on the message options indicated by the producer, it may receive a single confirmation from IFEN indicating that every subscriber is aware of its event (without knowing exactly who the subscribers are - message 9.) or it may receive a confirmation from each subscriber (10.). While in the first case the producer does not wish to know who the subscribers are, in the second case it needs to further communicate with them and needs to know exactly who they are (see further description below).

Table I summarizes the list of primitives specified for IFP.

### B. IFP Flags

In addition to primitives, a set of flags has also been specified to be supported by the protocol. These flags enable the following indications:

- **Mandatory Action (sent with an event):** set by the event producer in order to “force” subscribing consumers to take a specific, possibly predefined action.

- **Priority Level:** enables the indication of the level of priority associated with a specific event (relevant for fast handovers or events related with critical situations).

- **Processing Rules:** enables the source of the event to indicate “rules” that should be followed by the receivers of the
event when processing/taking action on that event. For example, in the case of mobility, this could be an indication to dissociate from a WLAN Access Point and connect to a 3G network.

- **Error Code**: a set of error codes can be defined in order to provide information to senders about the reason(s) that cause a problem (e.g., during subscription, registration or related with an event distribution/processing).

- **Event Lifetime**: the event sources may specify the event valid period that corresponds to the time interval that event receivers have to start processing the event. After that time interval, the event may not make sense any more (this information may complement the “Priority Level” flag).

- **Private Event**: occasionally it may be desired by some entities to exchange some events internally (e.g., belonging to the same domain), but refrain from forwarding them to other entities. The information exchanged may also be encrypted in some cases.

- **Encryption**: enables the exchange of event information only intended to a set of elements (those that can decrypt the information).

- **Event Confirmation Request**: enables the producer to request confirmation that the subscriber(s) received the message. The producer may also specify the type of confirmation it is interested in: (1) individual confirmations – sent individually by the subscribers directly to the producer (always used when IFEN is not present to aggregate the confirmations and optionally used when the producer wants to know who are the subscribers in order to further interact with them); (2) aggregated confirmations – sent by IFEN to the producer in order to confirm that all subscribers have received the message and will potentially process it (this type of confirmations can only be used when IFEN is present). Confirmation messages are sent in order to guarantee that subscribers are aware of the event message they received and not to confirm the correctness of the received message, which can be done by the Transmission Control Protocol (TCP).

- **Version**: helps extending the protocol.

The flags presented address the needs that were identified during the protocol specification. However, additional flags can be specified in the future.

### V. PROTOCOL ANALYSIS

IFP supports the usage of IFEN or EDB. Nevertheless, one scenario or the other implies the exchange of a different number of messages as well as state stored in the system. This section compares both scenarios regarding the number of messages exchanged in the system to distribute one event as well as the state that needs to be stored in the system.

The scenarios are compared by deriving the functions that

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
</table>
| RegisterEvent(Symbolic name, [Certificate], Event IDs, Event Types, [Event Description], [Registration Timeout]) | Entities: producer → IFEN/EDB
Goal: message sent by producers to register one or more events. |
| DeleteEvent(Symbolic name, [Certificate], Event IDs) | Entities: producer → IFEN/EDB
Goal: message used to delete the registration of one or more events. |
| QueryEvents(Symbolic name, [Certificate], Event IDs, Event Types, [Event Description]) | Entities: consumer → IFEN/EDB
Goal: message used to retrieve the events for subscription respecting specific conditions (default is all events). |
| SubscribeEvents(Symbolic name, [Certificate], Event IDs, Event Types, [Subscription Timeout], [Filters]) | Entities: consumer → IFEN/EDB
Goal: message used to subscribe to one or more events. |
| UnsubscribeEvents(Symbolic name, [Certificate], Event IDs) | Entities: consumer → IFEN/EDB
Goal: message used to unsubscribe to one or more events. |
| SendEvent(Symbolic name, [Certificate], Event IDs, Event Values) | Entities: producer → IFEN,
producer → consumer, IFEN → consumer
Goal: message sent in order to distribute one or more events. |
| EventsAvailable(Symbolic name, [Certificate], Event IDs, Event Types, [Event Description]) | Entities: IFEN/EDB → consumer
Goal: return message containing the set of events available for subscription. |
| RegistrationConfirmation(Symbolic name, [Certificate], Event IDs) | Entities: IFEN/EDB → producer
Goal: return message confirming event registrations. |
| SubscriptionConfirmation(Symbolic name, [Certificate], Event IDs) | Entities: IFEN/EDB → consumer
Goal: return message confirming event subscriptions. |
| SubscriptionDeleted(Symbolic name, [Certificate], Event IDs) | Entities: IFEN/EDB → consumer
Goal: return message notifying a consumer that its subscription has been deleted (i.e., in response to an “UnsubscribeEvents” message, due to the subscription timeout or event deletion). |
| RegistrationDeleted(Symbolic name, [Certificate], Event IDs) | Entities: IFEN/EDB → producer
Goal: return message notifying a producer that its registry has been deleted (i.e., in response to a “DeleteEvent” message or due to registration timeout). |
| Error(Symbolic name, [Certificate], Error Codes, [Error Cause]) | Entities: all
Goal: message used for notifying another entity that one or more errors occurred (e.g., during the subscription, during registration or related with one event). |
| ConfirmEvent(Symbolic name, [Certificate], Event IDs) | Entities: consumer → producer/IFEN, IFEN → producer
Goal: return message sent to the event producer to confirm the reception of one or more events (message sent directly by the subscribers, or sent by the subscribers to IFEN that takes care of aggregating the confirmations and issuing a single message towards the producer). |
| AlertNewEvent(Symbolic name, [Certificate], Event IDs) | Entities: IFEN/EDB → consumer/producer
Goal: notification message sent to other registered and subscribed entities informing about a new event available. |

*Note: optional parameters are indicated with “{}”.*
relate the number of messages exchanged with the number of subscribers. A single event producer is considered in all cases.

To derive the number of messages exchanged using IFEN and EDB, three cases are defined:

- **Case 1**: only messages directly related to the distribution of events are considered. DEEP messages sent to discover IFEN or EDB, registry/subscription and confirmation messages are not considered (i.e., assuming that producers and subscribers registered prior to evaluation).

- **Case 2**: all messages to distribute the events are considered; “new event available alert” and “query events” messages are not considered, since they are optional and add the same number of messages in both scenarios (i.e., IFEN and EDB).

- **Case 3**: all messages to distribute further events to the same subscribers are considered (i.e., after the first one has been sent); “new event available alert” and “query events” messages are not considered.

### A. Analysis of Case 1

Case 1 is the most simple, since it discards many of the messages that need to be exchanged in a real scenario prior to event distribution, unless the system was pre-configured with all the necessary information. The objective of considering this case is to highlight the differences regarding the number of exchanged messages that are totally related with the event distribution.

Using IFEN the function that relates the number of messages with the number of subscribers is:

- \( f_1(n) = 1 \text{ event} + n \text{ filtered events}^{1} \sim n. \)

Similarly for EDB:

- \( f_2(n) = 2 \text{ lookup messages (1 lookup; 1 reply)} + 2n \text{ DEEP messages}^{2} \) (1 discovery; 1 return message to discover “n” subscribers) + n events (distributed by the producer) \( \sim 3n. \)

The difference between the two results is a consequence of the functionalities of IFEN when compared with a Database: besides storing the mapping between events available and subscriptions, it also stores the IP addresses of consumers and takes care of distributing the generated events.

### B. Analysis of Case 2

Case 2 addresses the messages that would be typically exchanged in real scenarios, since it considers also the messages used to discover the necessary entities as well as registrations/subscriptions.

Using IFEN the function for the number of messages is:

\[ g_1(n) = 2 \text{ DEEP messages (discover IFEN)} + 2 \text{ registry messages (1 registration; 1 confirmation)} + 2n \text{ DEEP messages (each subscriber discovers IFEN)} + 2n \text{ subscriptions (1 subscription; 1 confirmation)} + 1 \text{ event} + n \text{ distributed events} \sim 5n. \]

And, when using EDB, it is:

\[ g_2(n) = 2 \text{ DEEP messages (1 discovery; 1 return message to discover EDB)} + 2 \text{ registry messages (1 registration; 1 confirmation)} + 2n \text{ DEEP messages (each subscriber discovers IFEN)} + 2n \text{ subscriptions (1 subscription; 1 confirmation per subscriber)} + 2 \text{ lookup messages (1 lookup; 1 reply)} + 2n \text{ DEEP messages (each subscriber)} + n \text{ events} \sim 7n. \]

The results of these two scenarios, when compared with the corresponding ones for the first case, highlight the overhead associated with the discovery of all entities, the registrations and subscriptions. Nevertheless, this overhead is required to guarantee the functioning of the system.

### C. Analysis of Case 3

Case 3 describes the system behavior after the first event has been sent and assuming that the subscribers are the same. The objective of this case is to compare the system without the overhead associated with the discovery of entities.

Using IFEN the number of messages exchanged is given by:

- \( h_1(n) = 1 \text{ event} + n \text{ filtered events} \sim n. \)

Using EDB the function is:

- \( h_2(n) = 2 \text{ lookup messages (1 lookup; 1 reply)} + n \text{ events (distributed by the producer)} \sim n. \)

The results indicate that in scenarios where the number of subscribers is constant (and their addresses are known from previous interactions) cases 1 and 2 provide the same results, as would be expected.

### D. Discussion

Based on the results presented and summarized in Table II, it is clear that using a single entity to store the registrations and distribute the events leads to fewer messages exchanged in the system. The exception is a “static” scenario where similar results are achieved with IFEN and EDB. This case is in fact the worst scenario for IFEN since the benefits of its functionalities (e.g., storing the IP addresses of subscribers) cannot be explored due to the static nature of the system. It should be highlighted that part of the load in the system is a consequence of using symbolic names instead of IP addresses. This is the price paid for flexibility and similar results would be achieved using HIP [16] or other solutions that separate the identifier from the location.

Another important comparison to be made is related with the state stored in each entity, since by separating identification from location each entity has to store the mapping between symbolic names and IP addresses.

---

1. Since IFEN stores the role of the entity as well as its IP address it does not need to invoke DEEP to resolve the IP addresses of the subscribers. It is the peer’s responsibility to keep its IP address updated at IFEN in order to receive event notifications.

2. Two DEEP messages is the minimum number of messages to discover one entity. Additional messages can be sent to discover one entity (i.e., similarly to what happens with DNS) but that situation is not considered.
of information (besides the ones already mentioned); however, exploring this possibility is beyond the scope of this article and is left to potential users of the protocol.

ACKNOWLEDGMENT

This work has been carried out in the framework of the Ambient Networks project (IST 027662), which is partially funded by the Commission of the European Union. The views expressed are solely those of the authors and do not necessarily represent the views of their employers, the Ambient Networks project, or the Commission of the European Union. The authors would like to thank their colleagues from the Ambient Networks project for inspiring discussions. This paper is based on joint work in the project.

REFERENCES

[8] Ying Yu, Ning Huang and Ming Ye, “Web Services Interoperability Testing Based on Ontology”, Proc. Fifth International Conference on Computer and Information Technology (CIT’05), Shanghai, China, September 2005