$PEACE$: A Policy-based Establishment of Ad-hoc Communities

Sye Loong Keoh, Emil Lupu and Morris Sloman
Department of Computing
Imperial College London
180, Queen's Gate, London, SW7 2AZ, UK
{*slk, e.c.lupu, m.sloman*}*@doc.ic.ac.uk*

Abstract

Ad-hoc networks are perceived as communities of autonomous devices that interconnect with each other. Typically, they have dynamic topologies and cannot rely on a continuous connection to the Internet. Users' devices often do not have a priori knowledge of each other and cannot rely upon pre-existing shared information. This introduces difficult security issues when attempting to provide authentication, membership management and access control. Designing a framework which allows the secure establishment and management of ad-hoc communities remains a significant challenge. In this paper, we propose a novel policy-based security framework to facilitate the establishment, evolution and management of mobile ad-hoc networks. We introduce a community specification, called doctrine, which defines the roles of the participants in the community, the characteristics that participants must exhibit in order to be eligible to play a role, as well as the policies governing their behaviour within the community. Based on the doctrine, we propose a set of security protocols to bootstrap the community, manage the membership, and govern the access to the services provided by the participants. We have investigated the impact of mobility on the proposed security protocols and observed that the protocol is robust to changes in the network topology.

1. Introduction

The proliferation of computing devices, which are being progressively embedded in the objects of everyday life, gives rise to numerous interactions and *collaborations* between these devices. Applications rely increasingly upon services provided by other computing devices and this dependence is exacerbated when devices are mobile, autonomous and interconnected through inherently unreliable wireless links. For example, implanted or wearable medical devices on a patient collaborate in order to monitor the healthcare of the individual; PDAs and laptops of different persons can interact in an ad-hoc business meeting and virtual organisations are frequently formed between businesses to share resources. Ad-hoc networking (e.g. MANET) is the platform that supports such collaborations, and the primary motivation of this work is to provide a policy-based security framework for establishing *communities* of autonomous devices in the wireless environment.

There is little related work addressing the issue of how to establish an ad-hoc network, determining who can participate and join the collaboration, what resources and services are needed, and who can access these services. All these issues are difficult to solve because ad-hoc networks cannot rely upon the availability of any fixed network infrastructure. Thus, the provision of security services is difficult as the information available to the user is limited, e.g. Certification Authorities (CAs) cannot be reached at all time. Various security issues such as authentication, membership management and access control to the resources shared in the network need to be resolved. Some of these issues have been addressed independently and prior research has mainly focussed on the context of ad-hoc routing [24, 16, 14, 13], middleware support [18], and communication infrastructure. However, few attempts have been made to address authentication and access control issues in ad-hoc networks.

A security framework is needed to support the establishment, evolution and management of ad-hoc networks in which users can have the assurance that their devices would only interact with other devices that the user trusts to be safe. Interactions between devices need to be controlled in order to prevent unauthorised access to system resources and services. A policy-based approach is flexible, scalable and permits adaptation to changes in security requirements and context of the ad-hoc network by dynamically loading and removing policies from the system without interrupting its functioning. In this paper, we propose a novel policy-based security framework and a set of security protocols to bootstrap an ad-hoc network, manage its membership, and control access to the services provided by the participants.

We also show that the proposed solution is robust to changes in the network topology.

The paper is organised as follows. In section 2, we present the notion of community in the context of ad-hoc networks. In section 3, we describe our framework, while section 4 outlines the protocols used to establish and manage an ad-hoc community. Section 5 describes the architecture and presents some preliminary results. In section 6, we describe related work, and this is followed by discussions and future work in section 7. Section 8 concludes the paper.

2. Ad-hoc Communities

In socio-economic studies, the term *community* commonly refers to a group of people with common *interests* or *characteristics* [31, 1] who live in the same *locality* [9, 1] under the governance of a set of *laws* [8]; and that they *interact* [19] with each other.

In the context of ad-hoc networking, we perceive an ad-hoc network as a *community* of autonomous devices that collaborate and share resources with each other. It is similar to the virtual enclaves [30] that enables the sharing of resources and services between several organisations. The motivation for users to form a community stems from the need to access resources and use services that they do not have on their own. We first present a definition of ad-hoc communities and subsequently, we outline the properties of an ad-hoc community with regards to this definition.

Definition 1. An ad-hoc community interconnects a group of devices, maintains membership and ensures that only entities, i.e., users or computing services, which possess certain credentials, attribute information and characteristics can join the community (*common characteristics*). The members of the community rely upon each other to provide services and share resources (*interactions*). These interactions are regulated through a set of well-defined rules and policies (*law*) that govern the access to the services and resources in the community.

2.1. Common Characteristics

The main characteristics of the ad-hoc communities we seek to define are that behaviour within the community is regulated through a set of policies and that only participants who satisfy well defined admission criteria can be admitted in the community. An example of such communities are disaster relief operations that require a coalition to be set up among the police, fire brigade, ambulance etc. and where membership is restricted, in terms of the characteristics of the users who can participate. This is inline with the notion of *regularity-based trust* [22] that builds trust relationships among the users in the community. Trust arises from the fact that participants in the community know that all other participants have satisfied the admission criteria specified and, consequently, have the expectation that they will behave accordingly. The next section discusses the need for policies to regulate the behaviour of the participants.

2.2. Policies or Law

Since the purpose of an ad-hoc community is to enable interactions between its participants, it is thus important to ensure that these interactions are governed by well-defined *policies* that define the rules for accessing services and resources in the community. Policies are explicitly specified and known to all the participants.

The rationale for explicitly specifying the rules or security policies is to build *trust* between the participants. Trust in this context derives from the fact that participants' behaviour is expected to be consistent with both the characteristics dictated by the admission criteria and the policies governing the behaviour within the community. Typically, the participants that form the community have to rely on each other to provide the services that they do not have on their own and usually, they do not have any *a priori* knowledge about each other. As a result, collaborations among them cannot be set up because they do not trust each other to use their respective services and resources. Therefore, there is a need for explicit specification of policies for each community. By knowing the policies, a user is aware of the potential users that it might trust to interact with, the services and resources that it has access to, and the policies it must enforce in order to protect its resources and services.

3. A Model of Ad-hoc Communities

In this section, we present a model for ad-hoc communities. The community *doctrine* is a specification that clearly defines: the roles of the participants in the community, the rules (or policies) governing their behaviour in terms of authorisations, obligations and constraints which external entities have to satisfy in order to join the community. Thus, only the eligible users are allowed to participate in the community, and each of them is allocated a set of access privileges to use the resources and services provided by other members. In addition, a set of protocols for the formation and evolution of the community is required and forms an integral part of the management of the community.

In essence, a doctrine is an information model that comprises tuples $\langle R, P, S, TK, Sig \rangle$ of a community, C, where:

- R - denotes the *role types* of the participating users in the community.
- P - defines a set of *policies* that regulate the behaviour of the participants assigned to the roles. Authorisation policies associate the permissions with roles,

while obligation policies are *event-condition-action* rules that facilitate the adaptation and security management of the community.

- *S* - defines the *constraints* of the community. It is used to specify security requirements, e.g. separation-of-duty and cardinality constraints which must be preserved when the membership changes.

- *TK* - denotes public-keys of the credential issuers, e.g. Certification Authorities (CAs) and Attribute Authorities (AAs).

- *Sig* - Each doctrine must be signed by its issuer in order to preserve its integrity and authenticity.

The following scenario gives an example of an ad-hoc community:

Alice is on a business trip from London to Paris. She will have to spend two hours on the train. With a limited number of songs on her PDA, she would like to engage in a media files sharing activity, where she can exchange songs with other passengers onboard. Most of her songs are purchased from the iTunes Music Store and Napster.com. She is a premium subscriber of both music stores, so she has the privileges to download unlimited number of songs. Therefore, the exchange of songs can only be performed with other passengers who are the premium subscribers as well, otherwise, there will be a breach of copyright issues. She is also interested to browse the top stories of today's news, e.g. Avantgo news on PDAs if there are any news services provided by other passengers on the train.

Typically, *doctrines* are specified by a variety of issuers. In this scenario, $EuroRail$ can issue a doctrine to enable its passengers to set up communities on the train or alternatively any music providers. The doctrine can be used to instantiate several communities with different participants and each participant maintains a set of preferences specifying the characteristics of the communities that it is willing to participate in. The decision to join a community is based on the attributes of the doctrine, e.g. whether the user trusts the issuer of the doctrine, or the services which the user will be allowed to access if it joins the community.

3.1. Role Types and User-Role Assignment Policies

The doctrine defines role types, R and user-role assignment policies, URA, so that access control permissions can be associated with roles rather than individual identities.

Let $R = \{r_1, ..., r_n\}$ be a set of role types of a community, *C*, where there is a finite number, $n \geq 1$.

In our scenario, communities can be established to facilitate the exchange of music files and news stories. Participants of the community can be represented as three role types, $R = \{$*premium user, normal user, news server*$\}$.

Each role type has a user-role assignment policy, URA, so that only authorised users are allowed to join the community in that role. In order to be assigned to a role type, the user must possess the required attributes, e.g. be a subscriber of a music provider, a member of AOL etc. This is termed as credential requirement, *cr* as it designates the credentials or certifications that a user or a service must exhibit in order to demonstrate that it possesses certain attributes or characteristics. This could be the user's position in an organisation, professional membership, etc.

Let E be a URA policy specified for each role type in the community. E is expressed in disjunctive normal form (DNF) where the predicates are made up of credential requirements. Users have to present a set of credentials or attribute certificates which will be checked against the URA policy. If the URA policy is satisfied, the user is assigned to the role type and an instance of the role is created.

As an example, the URA policy for the roles *normal user*, *premium user* and *news server* can be defined in the corresponding DNF as follows:

$$E_{normal} = \{Subscriber_{iTunes} \vee Member_{AOL} \vee Subscriber_{napster}\}$$
$$E_{premium} = \{Premium_{iTunes} \vee Premium_{napster}\}$$
$$E_{news} = \{Passenger_{EuroRail}\}$$

where only the subscribers of iTunes, napster or members of AOL can join the community as *normal user*. To join as a *premium user*, the user must be a premium subscriber of either iTunes or napster, while any passenger who can provide news service can join as *news server*.

A user can satisfy more than one URA policy and hence can be assigned to more than one role in a community. Therefore, in order to ensure separation-of-duty when required, there is a need to define additional constraints to prevent a user from being assigned to more than one role type. This will be discussed in section 3.3.

3.2. Authorisation and Obligation Policies

Authorisation policies are specified in the doctrine to grant participants access to services and permissions to use resources, while obligation policies facilitate security management of the community. These policies are based on Ponder [10] and they are grouped according to the role type specifications. Hence, this is similar to the role-based access control (RBAC) model [28, 11]. An authorisation policy defines what actions a subject role is permitted to invoke on a target object. It protects the target objects from unauthorised actions and this policy is enforced at the target. An obligation policy specifies the actions that must be

performed by the subject role when an event occurs [10].

The proposed framework groups all authorisation and obligation policies according to the subject role, i.e., the rights and duties of that role within the community. In mobile networks, users who provide services are expected to enforce the authorisation policies and all service requests are then granted based on these policies. This implies that all service providers must be able to interpret the policies and subsequently enforce them [23]. The obligation policies are perceived as duties of the users to manage the security of their corresponding services as well as the community as a whole. In general, they are used to facilitate the management and adaptation of the community to changes in its membership or context.

```
inst role NormalUser {
    inst auth+ listenMusicAuth {
        Target NormalUser, PremiumUser;
        Action listen(); }

    inst auth+ readNewsAuth {
        Target NewsServer;
        Action read(); }
}
```

Consider for example the composite policy illustrated above, it specifies that the role *normal user* is authorised to $listen$ to songs (music files) shared by other *normal user*s and *premium user*s, it can also $read$ news provided by the *news server*.

```
inst role PremiumUser {
    inst auth+ downloadMusicAuth {
        Target PremiumUser;
        Action download(); }

    inst auth+ listenMusicAuth { ... }
    inst auth+ readNewsAuth { ... }

    inst oblig maliciousDownloadAction {
        On     3*maliciousDownload;
        Action disable() → log() → notify(); }
}
```

Similarly, the composite policy for the *premium user* comprises three authorisation and one obligation policies. The authorisation policies specify that a *premium user* can $download$ music files from other *premium user*s as well as tune in live, i.e. $listen$ to songs provided by other *normal user* and *premium user*. The same access privilege is given to $read$ news. The obligation policy specifies that when the system detects that there are three consecutive malicious attempts to download the user's shared music files, the user must $disable$ the file shares, log all the unauthorised access details, and subsequently $notify$ other members of the community. ($\rightarrow$ indicates that the actions are executed sequentially).

3.3. Constraint Specifications

Based on a doctrine, a community can be instantiated when a group of users who satisfy the URA policies wishes to engage in an ad-hoc collaboration. However, prior to the instantiation of the community, certain security requirements must be fulfilled. The doctrine provides the flexibility to specify constraints that express the security requirements of the community. These constraints are evaluated when the community is first instantiated and whenever the membership changes.

Three types of constraints can be specified: *separation-of-duty (SoD)*, *cardinality* and *community establishment*. SoD constraints ensure that a user is not assigned to two or more conflicting roles at the same time in the community. The purpose of SoD rules is to prevent one user from doing all parts of a task that should require two or more users, in order to prevent collusion or fraud [17]. The cardinality constraint restricts the number of role instances in the community as well as the total number of participants, while the community establishment constraint is a definition of conditions on the instantiation of a community. It ensures amongst others that the indispensable roles and services are available prior to the establishment of the community, and defines the minimum number of instances for each role type. In essence, a community can only be instantiated when the required role types have been instantiated.

For example, a *community establishment* constraint can be specified as there must exist at least two *normal user*s prior to the establishment of the community.

$$S_{com\_establishment} = \{|normal\_user| \geq 2\}$$

3.4. Trusted Key Specifications

The issuer of the doctrine can include the public-keys of the relevant CAs and AAs in the doctrine. These public-keys are required by the participants when verifying each other's credentials, e.g. public-key or SPKI/SDSI [27] certificates. Essentially, the issuer knows which public-keys of the CAs are needed for the URA policies, hence it can asserts the authenticity of these public-keys by including them as trusted key specifications in the doctrine. This is necessary because in most cases, it is unlikely that every participant maintains the public-keys of all potential issuers and due to the lack of a continuous online connection to the fixed network infrastructure, these authorities cannot be reached at all times. Thus, it is difficult to verify the

participant's credentials without the public-key of credential issuers. However, this implies that participants have to trust the issuer to provide accurate information regarding the public-keys that are needed to check URA policies.

In the scenario described in section 3, in order to verify the URA policies, one would need the public-keys of iTunes, napster, and AOL. These public-keys can be included in the doctrine as trusted key specifications.

$$TK = \{Pub_{iTunes}, Pub_{napster}, Pub_{AOL}\}$$

The doctrine is encoded using XML in order to provide interoperability between various mobile devices. The hash of the doctrine is used as its identifier, so that any changes to the doctrine can be detected. All doctrines are signed using XMLSignature [3].

4. Deployment and Enforcement Architecture

Having defined the policy specifications for the ad-hoc community, this section briefly discusses the assumptions in the deployment model, the process of creating and disseminating the doctrine, the process of bootstrapping a community and the management of community membership in terms of participants joining and leaving the community.

4.1. Assumptions

A community doctrine defines the rights and duties of the participants as well as what behaviour a user can expect from other participants in the community, so all participants need to be aware of the doctrine, (i.e., have a copy of the doctrine).

The devices of participants can have heterogeneous capabilities ranging from laptops, to PDAs and low powered sensors. Since some devices will not be able to perform complex cryptographic operations, it is useful to distinguish between high-capability devices that can be used to perform coordination tasks and the other devices. High-end devices will need to perform computationally intensive tasks on behalf of the other devices that must trust them to perform these tasks. As previously discussed, this trust is not unreasonable for two reasons: First, because without it the community would not be formed (or at least not with any form of security) and second because these devices have satisfied the URA policies.

We assume that there is an underlying routing infrastructure that supports the relay of data packets in an ad-hoc network. Ad-hoc routing is still an active research area, and many routing algorithms have been proposed [24, 16]. Some of them have been extensively tested in simulations.

Every participant maintains its own attribute certificates and key pair that must be stored in a secure keystore.

Doctrines for the instantiation of a particular type of community can be issued by any organisation or any individual, e.g., in a smart home scenario, a user can issue a doctrine to interconnect mobile devices at home. Doctrines can be disseminated through broadcast messages or made available on websites. Generally, their content is not expected to be confidential although their integrity must be preserved. However, users willing to join a community governed by a particular doctrine may need to trust the issuer of that doctrine with respect to the accuracy of the certification and attribute authority's public-keys the doctrine comprises. It is entirely at the user's discretion whether to request joining a community governed by a particular doctrine or not. In most cases, users will have defined a set of preferences in terms of the communities that they wish their devices to join spontaneously. Such preferences are specified as Boolean expressions on the attributes of the doctrine (including the privileges and obligations of the role the user would wish to play, trust in the issuer of the doctrine, and current context).

4.2. Bootstrapping a Community

When mobile users come into proximity of each other, they can set up an ad-hoc community. One of the users with a device of high CPU capability (a laptop or PDA), u_{co}, can initiate the bootstrapping of the community, by broadcasting a ⟨REQUEST⟩ message. The request must contain the doctrine and credentials of u_{co}. When other users receive the request, they evaluate the doctrine against their preferences and decide whether they would like to accept it. They also have to authenticate u_{co} and check that its credentials satisfy the URA policy for the role it proposes to fulfil. Subsequently, they send a ⟨REPLY⟩, which contains their respective credentials to u_{co} if they want to participate.

For all the replies received, u_{co} checks the credentials to ensure that they satisfy the URA policies in the doctrine. Then, u_{co} evaluates the constraint specifications of the doctrine, to ensure that SoD, cardinality and community establishment constraints are satisfied. All admitted users are then assigned a *node id* based on the time of admission and a membership list is created for the community. This list is then broadcast to all participants in a ⟨MEMBERSHIP⟩ message and u_{co} automatically becomes the coordinator of the community. The community is thus created. Throughout the lifetime of the community, the coordinator is responsible for enforcing the URA policies, evaluating constraints, maintaining the membership list and responding to admission requests. Thus, only devices with high CPU capabilities can be selected to act as coordinator.

The use of a single coordinator may be controversial as it introduces a central point of failure and also a vulnerability if the coordinator is malicious and/or compromised. However, the alternative, i.e., that every participant performs all

of the verifications, leads to numerous redundant computations (verifications of credentials and of constraints) and also excludes from the community all devices that do not have the computational capabilities to do it. Furthermore, to evaluate the community constraints, the information regarding the size of the community is needed and up-to-date membership information must be maintained. Replicating the coordinator requires strong consistency of membership lists. Although ISIS [4, 5] and other traditional group management systems [26, 20, 2] as well as *virtual synchrony* [6] defined an approach to maintain the consistency of membership lists, they are not suitable for ad-hoc devices that have scarce resources, because they impose significant communication overheads among the coordinators. Hence, the simplest way is to have a coordinator that maintains weak consistency of the membership list by periodically broadcasting it to all participants.

Note that the unavailability of the coordinator does not entirely prevent the community from functioning. The main role of the coordinator is to maintain the membership of the community and verify credentials and constraints for new members. Thus, if the coordinator becomes unavailable, new participants cannot be added to the community. However, existing participants can continue their interactions within the current community membership without interruption. A compromised coordinator is a more serious threat as it can admit rogue participants to the community. The only way to detect this is if the rogue participants violate the policies and the other participants detect this.

4.3. Joining the Community

A new user, u_{new} periodically discovers new communities in the vicinity. It automatically requests to join the discovered communities that use doctrines conforming to its preferences. This is achieved by sending a ⟨JOIN REQUEST⟩ to the coordinator of that community, which contains u_{new}'s credentials and the role, r_u that it wishes to join as.

Upon receipt of the join request, the coordinator checks that u_{new}'s credentials satisfy the URA policies and checks that the admittance of u_{new} does not violate the cardinality constraints. If u_{new} requests to join the community in more than one role, the coordinator must also ensure that SoD constraints are not violated. A *node id* is then assigned to the admitted user and the coordinator sends a ⟨JOIN REPLY⟩ to u_{new}. Subsequently, the membership list is updated and broadcast to all participants.

4.4. Service Access and Subsequent Interactions

Access to services provided by participants is regulated by the policies defined in the doctrine. When a service provider receives a request, it first checks the membership list in order to determine the validity of the requestor's role assignment. Then, it grants the requestor the permissions to use the service if the authorisation policies allow it.

If the provider has sufficient device capability it may decide to re-verify the user-role assignment. This helps to guard against a malicious or compromised coordinator having admitted a rogue user in the community. If a violation is detected, other participants can be notified and if needed the community can be reconstructed (see section 4.6).

4.5. Leaving the Community

Two scenarios can occur: either the user notifies the coordinator that it is going to leave the community or its unexpected absence is detected by others. If it is temporarily absent (e.g., user moves out of range) but its absence is not detected by other participants, no changes are necessary.

The first scenario is straightforward as the coordinator can remove the user from the membership list, which can then be re-sent to all participants. In the second scenario, we rely on the other participants detecting its absence, typically through a communication failure. When a communication failure occurs, a user will retry for up to τ times. If the failure is confirmed, the user notifies the coordinator by sending a signed ⟨FAILURE NOTIFICATION⟩.

Upon receipt of such a notification of failure to reach a user, u_{leave}, the coordinator will checks that u_{leave} is indeed a member of the community and attempt to reach that user for τ times. If u_{leave} is still unreachable, the coordinator removes it from the membership list, checks that the community establishment constraints are still satisfied and broadcasts the revised membership list. If the constraints are not satisfied the community can be dissolved.

4.6. The Coordinator is Unavailable

The coordinator can also become disconnected in an unexpected manner. When the disconnection has been confirmed, an ⟨UNAVAILABILITY⟩ message is broadcast to all participants. The community is considered by all participants to be in a *static* state until a new coordinator is elected. Static state (i.e., coordinator absent) implies that no new users can be admitted to the community and all participants disregard membership updates until a new coordinator is elected. The community continues to operate based on the last authenticated membership list.

Upon receipt of the ⟨UNAVAILABILITY⟩ message, each participant checks whether or not it has the resources and capability to serve as coordinator. Among all the eligible participants, only one participant is selected and the choice is arbitrary. For example, the participant with the *lowest node id* can be selected as it is the oldest participant

in the community amongst the eligible ones. The selected user, u_{lowid} then broadcasts a ⟨RECONSTRUCTION⟩ message and the other participants re-join the community by sending ⟨REJOIN⟩ to u_{lowid}. Subsequently, u_{lowid} checks all the URA policies and community constraints, and if they are fulfilled, u_{lowid} re-established the community. However, if no participant can take over the role of coordinator, the community remains in static state until it is dissolved.

5. Architecture

Figure 1 shows the overall architecture of the proposed framework. It is composed of five components: profile management, membership management, protocol management, policy enforcement and an event service. The framework runs on every user's device.

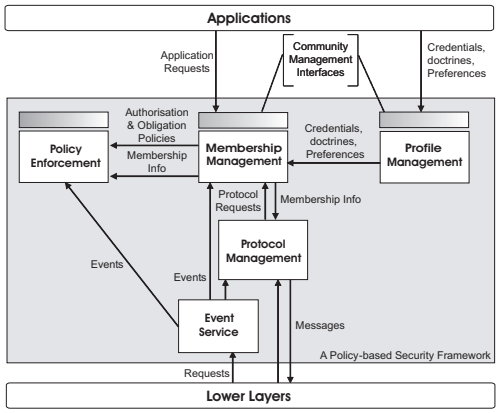

Figure 1. The architectural overview.

The *profile management component* maintains the user's credentials, such as public-key certificates, private-key stores, and attribute certificates. Users can manage their credentials and device settings through the *community management interface*. In addition, this component also maintains the user's preferences on which communities the device should automatically join.

The *membership management component* exposes the community management interface to the application level, so that applications can initiate the establishment of a new community, search for communities, as well as joining particular communities. Through this interface, the user can register the services that it is providing to other participants. The membership management component is also responsible for checking the authenticity of the doctrines and enforcing them by extracting and distributing the policy instances to various enforcement components. An optional module, known as *coordination service* can be dynamically loaded according to the user's device capability in order to enable the device to act as the coordinator. In this case, the membership management will also manage the membership of the community including the enforcement of URA policies and the community constraints.

The *protocol management component* executes various security protocols for the establishment, evolution and management of communities as discussed in section 4. The *policy enforcement component* enforces both the authorisation and obligation policies. Access requests are intercepted and then checked against the policies to determine if they are permitted. Obligation policies are enforced by subscribing to the specified event and executing the actions specified in the policies when the events occur.

Lastly, the *event service* collects and aggregates events and subsequently forwards them to the policy enforcement, e.g. the triggering of the execution of obligation policies. System events are forwarded to the protocol management, so that appropriate protocols can be performed. Events regarding the discovery of new communities are forwarded to the membership management component.

5.1. Emulations and Preliminary Results

We have designed and implemented an emulation of a mobile environment in order to investigate, the robustness of the protocols against node mobility and to determine the impact of various pause times on the proposed protocol.

The architecture was implemented as a Java prototype and was tested using the MobiEmu [34] emulation tool. MobiEmu was chosen because it can emulate the mobility encountered in wireless networks and provides support for ad-hoc routing. Essentially, MobiEmu provides a software platform for testing and analysing ad-hoc network protocols and applications in a LAN setting. The emulation is scenario-driven and simulates the movement of nodes given a history of locations. In terms of routing, it enforces a partially connected topology at the data link layer and uses the best-case ad-hoc routing algorithm.

A testbed was set-up to emulate the establishment and evolution of an ad-hoc community. For simplicity, all machines were configured as capable to act as coordinator and each of them was loaded with attribute certificates. A doctrine corresponding to the scenario described in section 3 was also created. Mobility scenarios were generated using $setdest$ version two, which can be found in ns-2 [7]. Table 1 summarises the parameters used in our emulations.

When running the emulation over a period of time, the community could end up in the *static* or *established* state. We are interested to find out the probability that a community will end-up in the static state at the end of the emulation, i.e., that the community can no longer be reconstructed. This occurs when the participants are out of range and are no longer able to receive reconstruction messages even if a coordinator has been identified. Conversely, a community remains in established state with the possi-

Table 1. Overall emulation parameters

Transmission Range	200 m
Emulation Time	1000 s
Number of Nodes	5
Pause Time	30, 60, 300, 600 s
Max. Node Speed	3.0 m/s
Min. Node Speed	0.1 m/s
Topology Size	500 x 500 m

bility that it has been reconstructed or it has never been reconstructed. From Figure 2, we observe that the probability for a community to end up in static state is of 0.1 to 0.2. This probability is low and thus the protocol proposed is relatively robust to the mobility of devices. Most of the time, the community remains in the established state. More specifically the probability that a community remains established without being reconstructed for various pause times ranges from 0.3 - 0.7.

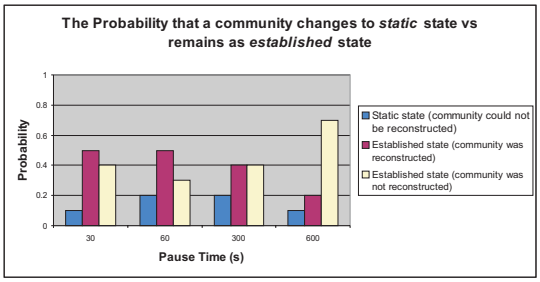

Figure 2. This graph shows the stability of the community against various pause times.

The second aspect is to investigate the probability of having to reconstruct the community as a function of the pause time of the nodes. As shown in Figure 2, we observed that the probability that a community goes through the *reconstruction* phase remains fairly high for pause times of 30, 60 and 300 s. For longer periods of time i.e., greater than 600s the coordinator remains within communication range of the other devices for longer periods and the probability of a reconstruction phase happening is relatively low i.e., 0.2. This finding reveals that the proposed framework could be efficient for scenarios such as communities in trains, libraries, and ad-hoc business meetings because in these scenarios, the users tend to move and then pause for a long period of time before they move again.

6. Related Work

Law Governed Interaction (LGI) [23] proposes a framework where interactions between a group of distributed agents can interact subject to communication rules (the law) specified using a simple Prolog notation. The underlying assumption is that all interactions must occur through trusted agents present at each of the sites to enforce the law. Our framework does not rely on trusted agents, but requires participants to monitor each other's behaviour.

Zhou and Haas [35] have proposed the use of threshold cryptography to thwart the problem of a single point of failure in ad-hoc networks. They proposed that multiple nodes can be used to act as CAs in an ad-hoc network. Each network has a certificate signing key that is divided into n shares and distributed to all nodes acting as CA. In order to participate in the network, a user has to get $t+1$ partial signatures from different CAs and submit to a combiner to generate a valid membership certificate, where $(n \geq 3t+1)$. This implies that at least n CAs must be available at any point in time and requires an efficient replacement scheme to substitute the departing CA nodes. The framework presented in this paper does not aim to establish CAs, but rely on a coordinator to manage the membership.

In [29], the authors have proposed an admission and membership control for P2P and MANET that uses limited consensus among current peers. The approach uses threshold cryptography. The authors also revealed that threshold cryptography seems to require significantly more time to complete the membership admission process.

The Terminodes [15] project aims at building a self-organised mobile ad-hoc network platform. Authentication aspects are mainly built on PGP [36]. A fundamental assumption is that all nodes have identical functionality and play an equal role to support self-organisation. Hence, there are no privileged nodes performing functions e.g., constraint verification, authentication, on behalf of others. Nodes issue their own public-key certificates and store them locally. Users issue certificates to each other, forming a PGP-like web-of-trust. In our framework, mobile nodes have different roles to play in the community and authentication is not fully decentralised.

Mäki et al [21] proposed a robust membership management framework where each ad-hoc network has a group leader that issues membership certificates to the mobile nodes. This is similar to the coordinator in our framework. The leader can delegate its leadership to other mobile nodes in order to avoid a single point of failure. However, more delegated leaders also means that there is a higher chance of compromising them.

The Resurrecting Duckling Protocol [33] is built on the notion of a master-slave relationship between mobile devices. The protocol has been extended to cater for secure transient relationships [32]. This work is complementary to the work presented in this paper. An integrated approach to formulating a security framework is required to express security policies, credential requirements and trust relation-

ships between autonomous devices in a consistent manner.

7. Discussions and Future Work

Our approach does not seek to establish CAs in mobile ad-hoc networks. Having a CA, does not resolve the issue of how the CA identifies and authenticates the mobile users, since they do not have *a priori* knowledge of each other. Current PKI models require users to show proofs of identification before the CA can issue a public-key certificate to the users. In a mobile environment, it is not possible for each user to physically prove its identity to the mobile user who acts as the CA when it wants to join the network. On the contrary, we leverage on existing security solutions where users already possess various certificates issued by their respective CAs and AAs in the wired environment. By using a well-defined doctrine, all participants have a common knowledge regarding the admission policies that need to be satisfied in order to join the community and can thus have expectations about the attributes of the other users in the community. We prefer broadcasting a membership list at regular intervals rather than issuing individual certificates of membership, which need to be revoked. For relatively small and dynamic communities, this appears to be more efficient than the use of certificate revocation lists.

Acting as a coordinator consumes substantially more computational resources than being a mere participant. However, there is no direct benefit from becoming the coordinator and no apparent motivation for a participant to take on this role. The role of the coordinator has been introduced in order to provide some form of security and coordination in communities where most of the devices would not have the computational resources for complex cryptographic operations. Without the coordinator, the community would not exist, or would exist without any form of security. The motivation for becoming the coordinator is therefore intimately linked to the need for the community to exist. This may be because the coordinator has an interest in obtaining information or specialised services from the other devices such as obtaining music files. Or it may be because there is an overall interest in the collaboration between various participants e.g., disaster-relief operations, business meetings, etc. Note that, it would also be possible to add a form of micro-payments to the framework presented in this paper, thus adding *financial incentives* to the coordinator role. However, we have not investigated this aspect in detail yet.

When running the emulations, we observed that an ad-hoc community could be partitioned, in which a community instance is split into two instances. This happens because a group of participants move out of the communication range of the coordinator at the same time, and they reconstructed the community and selected a new coordinator. As a result, the community split into two distinct communities. However, we argue that this is not an issue because a doctrine can be used to instantiate multiple communities with different participants. Hence, provided that the community establishment constraints have been fulfilled, a new community instance can be created although there is an existing community that uses the same doctrine.

Currently the broadcast of the membership list requires extensive use of digital signatures. We are investigating ways to eliminate this need through the use of TESLA [25].

Ensuring that all entities behave according to the specified policies in a distributed system remains an open problem. In our framework, the coordinator has to enforce the URA policies and community constraints, whereas each individual participant has to enforce the obligation and authorisation policies pertaining to their role. Ensuring that they do so remains very difficult. The LGI [23] framework achieves this by assuming that each site has a trusted controller that mediates all message exchanges. This would require the use of a Trusted Computing Platform [12], to ensure that the user's device runs reliable software that has not been tampered with. The approach we are currently investigating focuses more on detecting non-compliant behaviour through monitoring of interactions. In particular, the coordinator can be detected to be malicious if users without appropriate credentials are admitted into the community. Further, when a service provider denies an access request to a user who has the right to use the service, this indicates a violation of the policies. However, continuous monitoring also requires a great effort from all the participants to co-operate with each other to detect anomalies.

8. Conclusions

In this paper, we have presented the notion of community as a representation of an ad-hoc network. We claim that a community should comprise a set of users who have satisfied well-defined characteristics, and a set of policies governing the users' interactions. We advocate the use of a community doctrine that comprises the policies specified in terms of roles and that can be instantiated within the appropriate context. Users are then assigned to the roles subject to the constraints specified in the doctrine. This approach presents three advantages: First, it is well suited for autonomous mobile devices as it requires relatively little processing to instantiate a community and avoids the need for negotiation. Second, it allows additional information to be conveyed as part of the doctrine (e.g., CA keys) provided the doctrine issuer is trusted and third, it allows to build trust between the community participants as they have knowledge of all the policies applying to the other participants and can therefore have expectations regarding their behaviour.

We have described a set of security protocols to manage the evolution of the community, in terms of its membership.

We have chosen to rely on a coordinator node (with high processing capabilities) as this avoids redundant computations. Although, this implies a certain degree of trust in the coordinator, other community members may randomly verify its actions. A simulation in a mobile environment has also allowed us to identify the cases where this model would be best applied by evaluating the stability of the community as a function of the participants' mobility.

Finally, the proposed framework can be generalised and being applied to other application areas, e.g. peer-to-peer networks and the establishment of virtual organisations between different companies in the wired networks.

Acknowledgements

We gratefully acknowledge financial support from the EPSRC for AEDUS research grant GR/R95715/01 and from the EU FP6 TrustCOM Project No. 01945. In addition, we are indebted for many comments and suggestions to our colleagues Naranker Dulay, Dan Chalmers, Leonidas Lymberopoulos and Nilufer Tuptuk.

References

[1] *Oxford Advanced Learner's Dictionary*. Oxford University Press, December 1998.

[2] Y. Amir et al. Transis: A Communication Sytem for High Availability. In *22nd IEEE Fault-Tolerant Computing Symposium (FTCS)*, July 1992.

[3] M. Bartel et al. XML-Signature Syntax and Processing, 2002.

[4] K. P. Birman. The Process Group Approach to Reliable Distributed Computing. *Communications of the ACM*, 36(12):37–53, 1993.

[5] K. P. Birman et al. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, 1991.

[6] K. P. Birman and T. A. Joseph. Exploiting Virtual Synchrony in Distributed Systems. *ACM Operating Systems Review*, 21(5):123–138, 1987.

[7] L. Breslau et al. Advances in Network Simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[8] A. Cohen. *The Symbolic Construction of Community*. Tavistock, London, 1985.

[9] Collins. *Colins English Dictionary*. Collins, 2000.

[10] N. Damianou et al. The Ponder Policy Specification Language. In *2nd Int. Workshop on Policies for Distributed Systems and Networks (Policy'01), Bristol, U.K*, 2001.

[11] D. Ferraiolo and R. Kuhn. Role-Based Access Controls. In *15th National Computer Security Conference*. NIST, 1992.

[12] T. C. Group. Trusted Computing Platform Alliance (TCPA) Main Specification, 2003.

[13] Y. Hu et al. Ariadne: A Secure On-demand Routing Protocol for Ad Hoc Networks. In *8th ACM International Conference on Mobile Computing and Networking*, September 2002.

[14] Y. Hu et al. SEAD: Secure Efficient Distance Vector Routing in Mobile Wireless Ad-hoc Networks. In *4th IEEE Workshop on Mobile Computing Systems and Applications* (WMCSA '02), pages 3–13, June 2002.

[15] J. P. Hubaux et al. The Quest for Security in Mobile Ad Hoc Networks. In *ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHOC)*, October 2001.

[16] D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[17] D. R. Kuhn. Mutual Exclusion of Roles as a Means of Implementing Separation of Duty in Role-Based Access Control Systems. In *Second ACM Workshop on Role-Based Access Control (RBAC 97)*, pages 23–30, 1997.

[18] M. Kumar and B. A. Shirazi. PICO: A Middleware Framework for Pervasive Computing. *IEEE Pervasive Computing*, 2(3):72–79, 2003.

[19] D. Lee and H. Newby. *The Problem of Sociology: An Introduction to the Discipline*. Unwin Hyman, London, 1983.

[20] C. Malloth et al. Phoenix: A Toolkit for Building Fault-Tolerant, Distributed Applications in Large Scale. In *Workshop on Parallel and Distributed Platforms in Industrial Products*, October 1995.

[21] S. Mäki et al. Robust Membership Management for Ad-hoc Groups. In *The 5th Nordic Workshop on Secure IT Systems (NORSEC 2000), Reykjavik, Iceland*, 2000.

[22] N. Minsky. Regularity-based Trust in Cyberspace. In *1st Int. Conf. on Trust Management (iTrust)*, May 2003.

[23] N. H. Minsky and V. Ungureanu. Law-Governed Interaction: A Coordination and Control Mechanism for Heterogeneous Distributed Systems. *ACM Trans. on Software Engineering and Methodology*, 9(3):273–305, 2000.

[24] C. Perkins. Ad-hoc On-demand Distance Vector Routing. In *MILCOM '97 panel on Ad Hoc Networks*, November 1997.

[25] A. Perrig et al. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5), Sept. 2002.

[26] V. Renesse et al. Horus: A Flexible Group Communication System. *Communications of ACM*, 39(4):76–83, 1996.

[27] R. L. Rivest and B. Lampson. SDSI – A Simple Distributed Security Infrastructure. CRYPTO'96, 1996.

[28] R. Sandhu and E. Coyne. Role-Based Access Control Models. *IEEE Computer*, 29(8):38–47, 1996.

[29] N. Saxena et al. Admission Control in Peer-to-Peer: Design and Performance Evaluation. In *1st ACM Workshop on Security of Ad-hoc and Sensor Networks (SASN)*, Oct 2003.

[30] D. Shands et al. Secure Virtual Enclaves: Supporting Coalition Use of Distributed Application Technologies. *ACM Trans. on Information and System Security*, 4(2), May 2001.

[31] M. Smith. Community. *Encyclopedia of Informal Education*, 2001.

[32] F. Stajano. The Resurrecting Duckling – What Next? In *The 8th Int. Workshop on Security Protocols*, 2000.

[33] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. In *The 7th Int. Workshop on Security Protocols*, 1999.

[34] Y. Zhang and W. Lee. An Integrated Environment for Testing Mobile Ad-Hoc Networks. In *3rd ACM Symp. on Mobile Ad Hoc Networking and Computing (MobiHoc)*, June 2002.

[35] L. Zhou and Z. J. Haas. Securing Ad-Hoc Networks. *IEEE Network Magazine*, 13(6), November/December 1999.

[36] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.