5-7-2007

The InterMesh Network Architecture

Joud Khoury

Jorge Crichigno

Henry Jerez

Chaouki Abdallah

Wei Shu

See next page for additional authors

Authors
Joud Khoury, Jorge Crichigno, Henry Jerez, Chaouki Abdallah, Wei Shu, and Gregory Heileman

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

SCHOOL OF ENGINEERING
UNIVERSITY OF NEW MEXICO

The InterMesh Network Architecture

Joud Khoury Jorge Crichigno Henry Jerez Chaouki Abdallah Wei Shu
Gregory Heileman [1][2]

UNM Technical Report: EECE-TR-07-007

Report Date: April 15, 2007

[1]Joud Khoury, Jorge Crichigno, Chaouki Abdallah, Wei Shu and Gregory Heileman are with the Department of Electrical and Computer Engineering, of the University of New Mexico, Albuquerque NM 87131, {*jkhoury, jcrichi, cabdallah, shu,heileman*}*@ece.unm.edu*. Henry Jerez is with the Corporation for National Research Initiatives, Reston VA 20191, *hjerez@cnri.reston.va.us*.

[2]The work presented in this report is partially funded by the National Science Foundation NSF under the Future Internet Design (FIND) Grant CNS-0626380.

Abstract

The rapid spread of mobile devices, the emergence of key wireless technologies, and the nomadic user and computing lifestyles on current networks are continuously evolving in synergy. MANETs, WSNs, and WMNs are examples of self-organizing unstructured networks that have their local communication paradigms and are optimized to perform under their particular physical constraints. Wireless Mesh Networks (WMNs) are particularly interesting because of their ability to operate in a pure ad-hoc mode or to include some infrastructural components, making them suitable for a multitude of applications. Inter-networking among the heterogeneous access networks is currently offered by the Internet Protocol (IP). However, the evolution of and the innovation within these networks is greatly hindered by the rigidity of the current Internet implementation and its lag in efficiently supporting flexible unstructured communication paradigms. To broaden the user's innovation space and to efficiently embrace the characteristics of emerging networks, clean-slate architectural approaches are being pursued. In this paper, we propose InterMesh, a novel iner-networking platform for wireless mesh networks. InterMesh enables heterogeneous access networks to converge at a novel Persistent Identification and Networking Layer (PINL), providing a seamless service to individual network entities. This paper identifies the key concepts behind the InterMesh network platform, presents an interesting prototype implementation that can coexist with today's Internet while still be able to evolve separately, and discusses some preliminary performance results of the prototype.

Keywords

InterMesh, Wireless mesh networks, Ad hoc networks, inter-networking, persistent identification

1 Introduction

The increased deployment cost and centralized architectures within structured networks are hindering innovation for both users and service providers on current networks. This, compounded by the limitations of today's Internet and telecommunication networks, is pushing vendors and enterprizes to adopt decentralized network architectures. For instance, wireless mesh networks WMNs are evolving as an alternative to traditional wireless networks. Several reasons are favoring the deployment of WMNs including, but not limited to, their low cost, self-organization independent of any infrastructure, robustness to link failures, and broader service coverage. WMNs support different connectivity paradigms ranging from infrastructure/backbone networks that include quasi-static mesh router components, to pure ad-hoc client networks (or a hybrid of both), making them suitable for a multitude of applications [5]. Currently, WMNs have been deployed for community networks, broadband home networks, wireless metropolitan area networks, enterprize networks, transportation networks, and wireless local area networks (WLAN). The mesh WLAN [13], for example, eliminates the need to wire access points (APs) within a LAN by connecting the APs with a wireless mesh reducing deployment and configuration costs. According to [5], WMNs are not a type of ad-hoc networks. On the contrary, ad-hoc networks can be viewed as special forms of wireless mesh networks. This is mainly because of the introduction of mesh router components into the mesh network. These routers can provide broad coverage, robustness to the overall network, integration with other wireless access networks (e.g., WiMax, Wi-Fi, and cellular networks), reduction in requirements on network clients (especially in terms of configuration and routing functionality), and mobility support.

There is an indispensable need for inter-networking of WPAN-, WLAN-, WMAN- and cellular-based wireless mesh networks. Currently, the Internet protocol (IP) is employed to provide this functionality at the network layer [17]. However, today's Internet implementation, aside from its great success, has several limitations, including: 1) the overloading of the IP address to simultaneously indicate network location and node identity, 2) the absence of a trustworthy environment for users to communicate, and 3) the questionable availability of centralized infrastructure and services. The advent of ubiquitous computing paradigms and the success of emerging access networks present an inflection point for introducing fundamental paradigm shifts towards designing a future inter-network. Hence, inter-networking mesh networks over IP will potentially limit the innovation within such networks for both the users and the service providers.

In this paper, we present the InterMesh architecture, a novel architecture with the goal of inter-networking heterogeneous mesh networks to provide a seamless service to individual network entities. The following key design concepts distinguish InterMesh:

- Intrinsic support for unstructured networks;
- persistent identification/naming and certification of network entities;
- a novel approach to dynamic and extensible network management and service provisioning using mobile agents; and
- seamless mobility.

InterMesh achieves convergence through a uniform Persistent Identification and Networking Layer (PINL), allowing mesh communities to form "on-the-fly" and merge with other networks. The PINL identifies network entities with persistent identifiers (PIs), that are globally unique, secure and accountable by design. We present a prototype implementation of InterMesh, and we demonstrate its successful operation over native mesh networks (of pure ad-hoc clients), and as an overlay on top of traditional IP networks, thereby achieving inter-operability with the current Internet.

The remainder of this paper is structured as follows. Section 2 discusses the principal design decisions that guides the development of the architecture. Section 3 describes the complete InterMesh system model, as well as its functionality. Mobility is briefly addressed in section **??**. A prototype implementation of the model is then presented in section 5 and the deployed test-bed is discussed in section **??** with the performance results. Section **??** reviews related work before concluding in section 8.

2 Design Guidelines

We have previously introduced a general architectural vision for a possible future Internet what we call the Transient Network Architecture TNA [4]. TNA represents an abstract vision from which InterMesh is instantiated. In this section, we layout the principal design guidelines that pertain to TNA and that guided the development of InterMesh.

2.1 Area of Influence - AoI

In this paper, whenever we speak of a local network, we are referring to what we call the Area of Influence (AoI), i.e. the AoI captures the scope of "local" when trying to understand how local is "local". Briefly, an AoI is a local communication community that defines its own communication protocols and network architecture implementations. Examples of local implementations include, but are not limited to, LANs, Cellular networks, MANETs, sensor nets, and mesh networks. These networks implement their own communication mechanisms and protocols and can survive independently of the global system. A sketch of how currently available networks can fit into the AoI framework, is shown in Figure 1. The figure shows how the nodes of a mesh network, for instance, may form into an AoI. The AoIs themselves may define their own local communication implementation such as Ethernet, RF or Bluetooth, and even their own local identification mechanisms. The basic constituents of AoIs are network entities which we formally define in section 3.1. Processes, services, devices are examples of network entities which we refer to hereafter simply as entities.

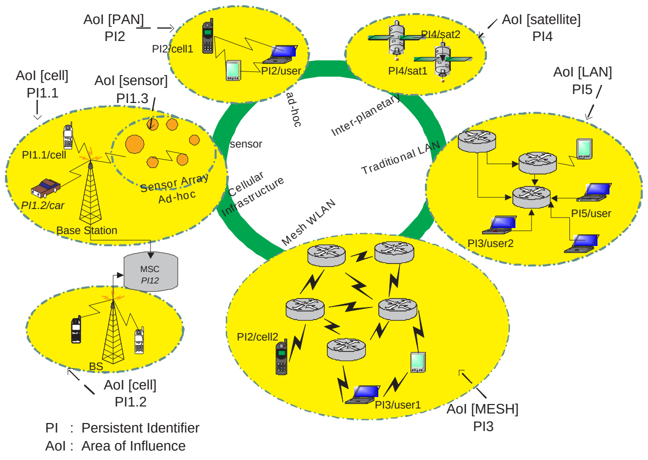


Figure 1: Examples of different Areas of Influence that form TNA

2.2 Entities and Communication

A major design decision that we adopt is the persistent identification of individual network entities, whereby each entity has its own PI that is globally unique, and secure by design. How is this different from the traditional Internet and what advantages does it offer? To answer this question, it is instructive to understand the relation between the entity and the attachment point to which the entity is bound. Traditionally, the Internet and particularly IP has taken a *location-oriented* paradigm to naming entities, i.e. the most basic entity identifier expressed as a tuple $\{$*IP address*, *port number*$\}$ is directly dependent on the topological IP address. This approach is intuitive in wired networks where an IP address identifies the wire connected to some device interface. But, does the *location-oriented* paradigm hold in wireless environments? So far, the IP address performed well as a location identifier since it inherently embeds topological information and thus fosters routing scalability. However, when mobility is introduced as in the case of wireless networks, IP looses any meaning of identity reference and degenerates into a pure routing identifier. Coupling the entity identifier to the routing identifier hinders mobility and poorly-identifies the actual entity, which should exist independent of network location. Several proposals have focused on solving the mobility problem by decoupling the host identity from the attachment point [18, 21, 25]. Most of the efforts propose inserting a level of indirection on top of the network layer that manages the abstraction of host identities. These proposals share the overlay approach on top of IP whereby a high level address is translated to an IP address at some point and routing is an end-to-end IP-based mechanism. The bottom-line is, *it is extremely hard and inconvenient to initiate communication with an arbitrary entity on the current Internet*, unless that entity has a public IPv4 (or an IPv6) address. Unfortunately, this is not the case with the mainstream adoption of Network Address Translators (RFC 1631) at the borders of stub domains. Additionally, even when a public IP address is available, inefficient mobility management schemes prevail requiring centralized infrastructure and continuous end-to-end negotiations between the endpoints over a dumb core.

We take an *entity-oriented* approach to naming and communication, in which the entity becomes the first-class network citizen. Contrasted to the traditional IP approach, our starting design point is an entity with a globally unique PI that is independent of any topological information. Starting with that, we try to design a practical and scalable network whose main currency is the PI. For example, the network should be intelligent enough to be able to efficiently route traffic based on the PIs. Aside from being an illustrative exercise, our approach natively supports mobility of entities and eliminates redundancy introduced by indirection on top of IP. Within InterMesh, and entity (e.g., a process executing on a mesh node) can initiate communication to any other persistently identified entity within the system without having to worry about delivery.

2.3 Naming and resolution

After reviewing the limitations of the IP address and introducing our *entity-oriented* approach to naming, its time to answer the main question: *How do we identify entities*?

We start by asserting the importance of identifier *persistence*. Persistence of the identifier is an attractive property in naming. It is essential when the attributes (e.g., state and location information) of the identified entity change continuously, but the identifier itself persists i.e. remains unchanged.

Before delving into other details, we try to answer a fundamental question: how do entities get introduced and how are namespaces composed? There are different approaches to solving this problem and these are mainly classified as either “top-down” or “bottom-up”. The main difference between the two is the dependence on a third party in the former case, versus the direct exchange of identifiers in the latter. Ostensibly, the “top-down” versus “bottom-up” paradigm is tightly coupled with trust flow [7]. For example, when a third party naming/trust system exists, identification flows “top-down” from the system towards the entity. To clarify our position, we separate the two critical mechanisms attached to the persistent identifier, *certification* and *resolution* as follows.

[[Aside:A note about namespace composition: It is more efficient, socially and economically, to compose the namespaces bottom-up. This means that local namespaces are formed first before any global namespace. This allows two key properties of a future naming system:

- In real life, trust is not provided by a single system as is the trend in the current Internet. Many systems should be able to provide trust and coexist, and it is up to the user to trust whichever suitable system,
- There can be many different naming schemes that should all be supported and respected instead of imposing a single global scheme on everyone,

But bottom-up name composition is pretty hard to achieve while maintaining scalability. My preliminary reflections for solving this problem are based on the democratic political election analogy: people are the most important part of the political system; a government is just a small set of people that are elected and trusted by the people. Thus governments (certification authorities) are composed bottom up. A new set of people can overthrow their government if they get enough people to elect them. The same analogy can be applied to the three namespace abstraction levels above.]].

Certification

An identifier is used by the entity for interaction with the rest of the system provided the identifier can be challenged and certified within the environment of communication whenever this is necessary. We isolate three certification realms, depicted in Figure 2, as follows:

- *Instance (Red Realm)* is defined relative to the user. It represents the authoritative domain of the user to which a set of entities belong. For example, Figure 2 shows a user instance (the bottom red ellipse) on two devices (Laptop, PDA) with several active entities (processes) certified by the user instance.
- *Local (Yellow Realm)* is defined relative to the local network, AoI. This realm represents the authoritative domain of an AoI which is essential for local communication among the entities of the AoI. Figure 2 shows this realm as the middle yellow layer that aggregates and certifies several *Red Realm*s belonging to the AoI.
- *Global (Green Realm)* is perhaps the most challenging to create and maintain, simply because it has to simultaneously guarantee global certification and scalability. The *Green Realm* represents globally trusted authorities. Note that at this level, many globally trusted authorities can co-exist and inter-operate avoiding the pitfalls of a single trust system as is the case with the current Internet.

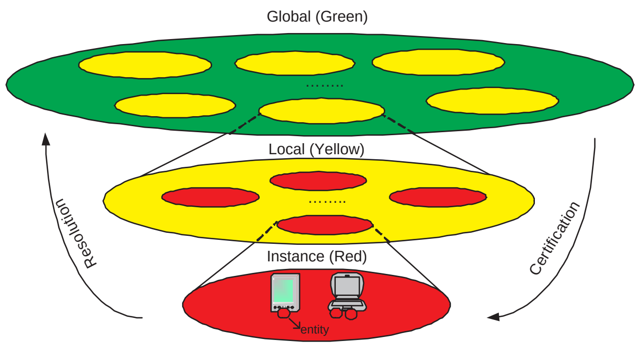


Figure 2: Naming abstraction levels

The colors of the realms indicate the degree of trust within each level. For example, certification by the *Green Realm* represents the highest level of trust with respect to the overall system. Certification is performed "top-down" as shown in Figure 2. For example, an identifier belonging to the *Yellow Realm* (the authority of an AoI) may be used for interactions within the AoI, provided the identifier can be challenged and trusted by the AoI if needed. The moment it is required that the identifier be used for interactions beyond the AoI (globally), the identifier might need to be trusted within the *Green Realm*. The trust mechanisms involved will depend on the particular architecture implementation.

Resolution

In general, the PI is resolved into some information useful for the interaction between the communicating entities. The result of the identifier resolution requires certification by the respective realm. The level (and mechanisms) of resolution depends on the particular architecture implementation. Our design decision is to perform resolution in a bottom-up fashion as follows: First, try to resolve against the *Red Realm*. A failure here will percolate the resolution one level up against the *Yellow Realm* and so on to reach *Green Realm*.

2.4 Distributed Network Management with Mobile Agents

Fully decentralizing and automating the network management tasks has been a topic of research in distributed systems [9]. The concept of mobile agents provides a novel approach to flexible and scalable distributed network management by better utilizing the network resources and minimizing human intervention [12]. For example, an agent can move the intelligence to the resource instead of moving the resource itself which can save bandwidth. Technologies to support mobile agents (e.g., JINI, JAVA, and CORBA) are becoming more popular and are moving closer to mainstream acceptance. A generic definition of the term "Mobile Agent" is: an autonomous software agent comprised of code and state which may both be mobile. The agent makes its own decisions and listens to external requests. It can execute custom business logic, move itself across the network, terminate itself, etc. The agent resides on customized infrastructure of agent hosts.

Within our system, we refer to mobile agents as GHOSTs, and to mobile agent hosts as SHELLs. We identify a minimal set of GHOSTS that are disseminated into the network to provide dynamic service provisioning and resource management. Additionally, we define a neutral interface that SHELLs expose in order to host agents and allow them to operate securely. We describe the GHOST/SHELL model in the context of InterMesh in section 3.1.

3 InterMesh System Model and Operation

We now introduce the InterMesh system model and functionality. To allow scalable inter-networking of the private address spaces (AoIs), InterMesh assumes the existence of a "core", a globally known rendezvous point for the AoIs. In other words, we introduce a topology consisting of the edge AoIs connected to a "core". The reason for this assumption is to enable practical and scalable routing mechanisms between the AoIs which is hard to achieve over a flat collection of private address spaces. With this in mind, an entity wishing to communicate beyond its AoI must specify an entry point into its AoI that correspondent entities can forward towards. The entry point is a routing GHOST as we shall see later.

A simplistic sketch of the model is shown in Figure 3. There are three essential components within the system: *entity*, *neutralization environment*, and *network substrate*.

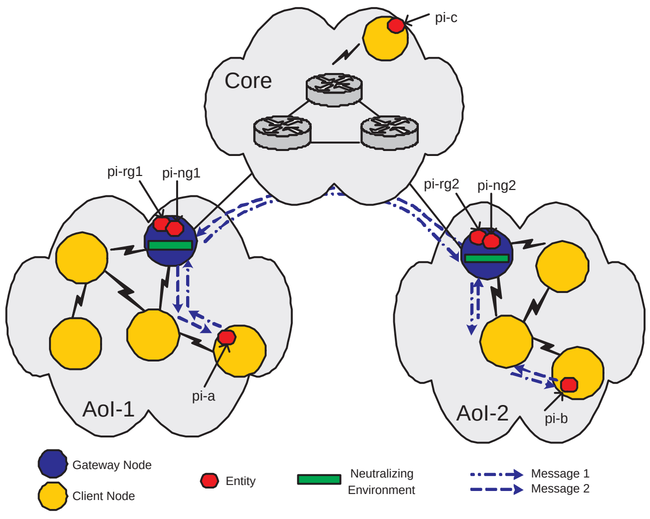


Figure 3: InterMesh reference model

3.1 Components

Entity

Based on the definition in [8], an entity is the end-point of communication. It is an abstract construct that can represent different network elements including, but not limited to, a process, a thread, a device, a cluster of devices, or a service. The entity is the smallest indivisible element on the network that can be mobile. All entities within InterMesh are persistently identified and addressed.

Neutralization Environment: The GHOST/SHELL virtualization Model

As mentioned earlier in the text, GHOSTs are necessary in our system to provide efficient and extensible service provisioning and network management. A set of nodes within the mesh AoI network provides the SHELL framework for GHOSTs. Part of those nodes, usually referred to as the gateway nodes, connect the mesh AoI to the "core" substrate (e.g. Internet). Obviously, a SHELL on a gateway node exposes at least two network interfaces to the GHOST, an internal interface to the mesh and an external interface to the "core". Another type of a gateway node might directly bridge different AoIs, a case we do not consider in this paper. There is nothing that precludes any node within the mesh to provide a SHELL. The SHELL exposes a neutralizing interface which in essence virtualizes the actual device's hardware resources to the residing GHOSTs. Figure 3 shows the neutralizing interface in green exposed by the SHELLs of the gateway nodes within the mesh AoIs. GHOSTs are themselves entities, hence, they are persistently identified. Figure 3 shows the GHOST entities in red (*pi-rg1*, *pi-ng1*, *pi-rg2*, *pi-ng2*) executing on top of SHELLs. The GHOSTs can move between SHELLs for reasons of resource optimization and fault tolerance. For example, in the latter case, a GHOST that detects lack/failure of physical resources on the hosting SHELL (storage, processing) can seamlessly relocate itself (with its state) to a different SHELL maintaining undisrupted service. This can happen, for example, during the course of a denial of service attack, and so GHOSTs provide a possible approach to recover from it. GHOSTs also relocate for control reasons, particularly to optimize the limited network resources. We have specifically addressed the agent relocation problem for the maximization of the network resources as an optimization problem [23].

As the SHELL and GHOST interfaces are beyond the scope of this paper, we just point out the essential properties of these interfaces. Every GHOST has a self-certification module that acts like a passport which the GHOST can present to the SHELL for authentication through the neutralizing interface. An administrator ships the GHOSTs from any point within the network after which they securely self organize on top of the SHELLs. An administrator must provide the GHOST with the administrative privileges over the SHELL. Besides the authentication operations, the SHELL's neutralizing interface exposes I/O and Digital Rights Management operations to the GHOSTs.

Note that the GHOSTs do not represent infrastructural components within the AoI, but on the contrary, they provide dynamic on-the-fly services for the rest of the entities in the AoI. For example, in an emergency (first responder) network, we envision a set of nodes rapidly forming into an AoI with the necessary GHOSTs automatically initializing and relocating to optimize the network utility. The Routing GHOST, for instance, locates the set of nodes with Internet connectivity bridging the emergency network to the Internet.

Network Substrate

In general, the network substrate is composed of the underlying communication infrastructure and services, as well as the mesh nodes. The infrastructure includes the edge mesh networks (AoIs), and a common "core" such as the Internet, the cellular infrastructure, or any other access/distribution network. As to the mesh nodes, we distinguish gateway nodes which specifically serve as entry/exit points between the the mesh network and the "core". Other types of nodes include client nodes or mesh routers. Usually mesh routers are quasi-static, have multiple interfaces as well as router/gateway functionality that is not present in clients (simple devices).

3.2 Functions

Entity Naming

With the proliferation of mobile devices and the anticipated large scale of the network, comes the challenge of how to design a system that is capable of naming individual entities at a large scale. InterMesh takes some assumptions in this regard in order to organize entities within the system. First, in order to participate in the system, an entity must acquire a *stamped* PI, i.e., a PI associated with a *stamp*. The latter is a credential acquired from a certification authority (CA) to authenticate the owner(s) of a PI. For example, a stamp can be the private key of a public/private key pair where the CA keeps the public key and only the owner(s) of the PI keeps the private key. Second, we isolate two classes of entities: *stand-alone* and *delegated*. A *stand-alone* entity obtains a globally unique stamped PI and can present the stamp on its own when necessary. On the other hand, a *delegated* entity exists within a user's execution environment. It is specifically introduced for scalability reasons by aggregating a set of entities under one authority: the user. This follows from the observation that processes (and threads) within an operating system normally run within an execution environment that includes ownership and rights information based on the user [9]. We expand on this fact to formalize the *delegated* entity abstraction in terms of the User-Entity Instance Model depicted in Figure 4:

- A User Instance (UI) represents the presence of a user on a set of devices. The UI has a *stamped* PI.
- A User-Entity Instance (UEI) represents an activity/service that exists and is unique under the UI. The UEI is a delegated entity certified by the User Instance.
- A particular UI can own multiple unique UEIs. The UEI can be instantiated at most once under a UI at any given time. All the delegated UEIs share the the authority and management scope of the UI.
- The UEI can potentially become a *stand-alone* entity that can migrate for example to a device under a different user instance. In this case, the UEI must be self contained and must carry a valid stamped PI.

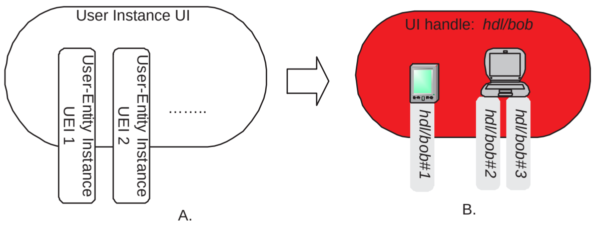


Figure 4: A. Entity Model, B. Example of UI on 2 devices with 3 UEIs.

Note that the User Instance is an instantiation of the *Red Realm* shown in Figure 2. Practically, one can think of the UI being initialized by the operating system when a user logs into a device and provides a stamped PI that persistently identifies the UI. One scenario here might use RFID tags [14] provided by a third party CA that stamps the actual tags. A user presents the tag to an RFID reader on the device and the OS automatically initializes the UI. Entities created on the device hereafter are managed by the UI.

Protocol Stack

Figure 5.A shows the logical layered structure of the native protocol stack within InterMesh. We have as well ported an overlay version of the stack that operates as an overlay on top of IP networks (Figure 5.B). The ported version is necessary for inter-operability with current IP networks. We focus in this subsection on the native stack operations.

The lower physical and link layers are common to all mesh networks. Different mesh networks have different physical constraints and performance requirements that necessitate proactive, reactive, hybrid, hierarchical, geographic etc. routing protocols to establish mesh connectivity [5]. The Mesh Structuring Layer - MSL abstracts these connectivity specifics and routing protocols from the upper layer enabling a wide range of emerging ad-hoc/mesh networks to be part of our architecture. The Microsoft Mesh Connectivity Layer [11] is a perfect example of a currently implemented MSL which we discuss later.

The PINL layer is an inter-networking layer that provides the necessary network services to foster evolution and innovation of the network. Protocols related to mesh network establishment, self-configuration, discovery, and packet delivery between persistently identified entities belong to this layer. Presented with a PI, this layer is intelligent enough to deliver a packet to its destination(s). Reliable delivery mechanisms can either be part of this layer or of a separate upper layer, but their specification is beyond the scope of this paper. In this paper, we solely focus on the PINL layer. What are the advantages of using the PI as the network address?

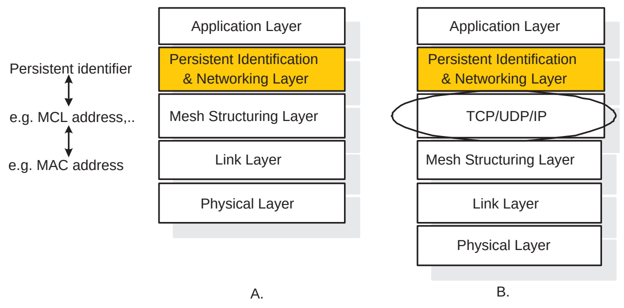


Figure 5: InterMesh logical protocol stack layering A. Native, B. Overlay

1. Persistence: The independence of the PI from its attributes is an attractive property for a network layer identifier. The direct advantage of persistence is mobility since an entity that is persistently addressed by the network layer is reachable on that address at all times. Consequently, mobility occurs natively eliminating the network layer indirection introduced by other proposals [18, 26, 25].

2. Security: The PI address is stamped i.e. it is inherently associated with security information (e.g. public/private keys) which can be used at all times by the communicating parties (and the network if necessary) for accountability, authentication and confidentiality.

3. Distributed administration: The PI has an incorporated administrative model that assigns administrators to PIs with access rights down to the level of the attribute. An administrator of a PI can securely self-administer the PI at anytime, and from anywhere within the network.

On the other hand, the implication of using a PI as the network address is, obviously, a more complex routing mechanism. Routing based on PIs is essential for our network to function properly. The routing mechanisms, introduced in [15], are not the subject of this paper. However, we discuss our experiences with a preliminary PI routing protocol in the context of the InterMesh prototype in section 5.

The application layer is basically the entity's interface to the network through which the service is provided. Finally, the cross layer interactions can improve network performance if applied cautiously [16].

Dynamic and Extensible Network Services

Routing across heterogeneous mesh networks is an example of a PINL layer service. To avoid ossification and to embrace major paradigm shifts within the network, it is essential to design services that are easily extensible and adaptable to change. We have so far identified two essential network services: naming and routing. InterMesh offers these services by utilizing the GHOST/SHELL model described earlier. As such, a minimal set of GHOST types is required to operate the InterMesh network classified into *Naming Ghost* - NG, and *Routing Ghost* - RG.

- Naming Ghost: This agent is particularly disseminated into the network with the goal of implementing the identification (name) service for the AoI. The GHOST appears as an entity to other entities in the AoI and is identified with a PI. Managing the namespace including creating, removing, and updating persistent identifiers within the AoI are operations of the name service which the GHOST implements. The nodes within the AoI are oblivious of the actual implementation specifics. In our model of Figure 3, the naming GHOSTs are represented by entities *pi-ng1* and *pi-ng2* providing the name service for *AoI-1* and *AoI-2* respectively.
- Routing Ghost: It is similar to the naming GHOST except for its functionality. The routing GHOST implements the actual overlay routing protocol that delivers packets between the local AoI and the rest of the network. Its operation is similar to the default gateway concept in the traditional Internet except that it routes based on the PI instead of IP address. Routing GHOSTs usually run on gateway nodes to perform packet routing beyond the AoI. Figure 3 shows the routing GHOSTs represented by entities *pi-rg1* and *pi-rg2* providing the gateway service for *AoI-1* and *AoI-2* respectively.

Both GHOSTs provide a discovery interface that allows other entities within the network to automatically discover the suitable GHOST. An administrator of the AoI ships the GHOSTs from any point within the network to the particular SHELL. After authenticating itself to the SHELL, a GHOST implements its intelligence independent of the actual SHELL hardware that is abstracted through a neutralizing interface. For example, an upgrade to the AoI routing algorithm simply requires shipping an upgraded GHOST to replace the old routing GHOST. In another scenario, a hardware failure on the SHELL will cause the GHOST to automatically move to a backup SHELL. In both scenarios, the routing service on the network is undisrupted and the administrator need not be physically present in the AoI.

4 Seamless Mobility

We need to devise an efficient mobility management scheme here to handle all the possible cases of mobility. Particularly, we need to efficiently cache PIs on the routing GHOSTs to minimize lookups, processing, and storage on the agents by shifting as much intelligence as possible to the endpoints (e2e). For example, when ghost2 in Fig. 3 receives a packet over the external interface (Internet) destined to pi-b, should ghost2 resolve pi-b or should it assume that pi-b is local since the message was received over external interface and thus forward locally with no resolution? This should probably depend on whether reliability is required or not.

We start by clarifying some of the concepts. Note that the association with a routing GHOST is performed by the device not the process i.e. all the entities on a particular device share the routing information in terms of available GHOSTs and their priorities. However, each process is responsible for updating its identifier binding in the system, even though the processes might share the same naming daemon on the device.

Our mobility scheme introduces a mobility index μ. The index will specify whether the mobility rate of the entity participating in communication is very slow (static), slow (partially mobile) or fast (highly mobile). This information is calculated by the entity (or per device shared among entities) and exchanged in communication instructing the routing ghosts about caching decisions. So, for example, a GHOST is likely to cache an entity's PI for a short time in case the entity has a high mobility index.

Also study the inter-domain mobility for multi-homed mesh networks where there are more than one entry points into the mesh [1].

5 Prototype Implementation

Having discussed the overall system design and functionality, we now present the implementation details of an InterMesh prototype. We particularly focus on the implementation of the naming, the protocol stack, and the mobile agents functionality.

5.1 Naming

We have implemented the entity identifier as a 2-tuple $\{PI,type\}$ constituting a globally unique and persistent identifier for the entity. The *Type* field serves a dual purpose. First, it differentiates between multiple entities sharing the same *PI* as in the case of multiple *delegated* UEIs under the same UI. The second purpose of *type* is partly analogous to the *port number* concept in traditional IP networks used to demultiplex the packet from the stack up to the entity abstraction. Particularly, it is useful for broadcast/multicast services (e.g. Discovery) on the network. The *type* is a *1* byte integer value. As to the *PI* part of the tuple, we have reused a current implementation of a persistent identifier called the *handle* which is part of the Handle System [2].Briefly, the Handle System provides a distributed, secure and global name service for administration and resolution of *handle*s over the Internet. A *handle* is a persistent identifier that can be associated with a set of attributes. Some of these attributes describe location, permissions, administrators and state. The fact that *handle*s are defined independently of any of the attributes or public keys of the underlying objects, makes them persistent identifiers [28]. Consequently, our *PI* implementation is a *handle* and we use the two terms interchangeably hereafter. For example, a possible identifier for the entity *pi-a* in Figure 3 is the tuple $\{2118/a, 5556\}$ where $2118/a$ is *handle* existing under the 2118 naming authority.

Additionally, the Handle System provides the *Green Realm* of certification within our prototype. Security is a crucial property of the Handle System. All the *Yellow Realms* represented by the Naming GHOSTs within the AoIs trust the Handle System as the top level certification authority. We note here that the Handle System allows for secure name resolution and administration in a distributed fashion making it extremely scalable [1] and suitable to operate in highly mobile environments.

5.2 Protocol Stack

The logical stack layering was depicted previously in Figure 5. In Figure 6, the detailed stack composition of the InterMesh prototype is shown. We note that the TCP/UDP/IP component of the stack (dotted box) is optional. It is only required when a node needs to operate as an overlay on top of IP. For example, the entity *pi-c* in Figure 3 is operating as an overlay over the traditional Internet and is part of InterMesh. We now explain the role of each of the components in the stack.

Microsoft Mesh Connectivity Layer - MCL

We have chosen to reuse the open-source Windows MCL driver implementation [3] as our Mesh Structuring Layer. MCL is an ad-hoc routing framework [11] developed as a open-source loadable Windows driver logically located between layer 2 (link layer) and layer 3 (network layer) within the traditional TCP/IP stack. On each node, MCL implements a virtual network adapter that appears to the higher layer software as an ethernet link. At the MCL layer, each node is addressed with a 48-bit virtual address. MCL routes packets through the network interface cards ($NICs$) using the Multi-Radio Link Quality Source Routing ($MR-LQSR$) protocol. MCL handles the underlying NICs as port interfaces through which it routes frames. To upper layers (i.e., PINL NDIS driver

[1]The largest individual Handle System implementation to date is deployed at the Los Alamos National Laboratory LANL, which is intended to support more than a half billion identifiers while providing internal resolution services to one of the largest archival collections in the United States.

and IP), MCL offers connectionless primitive services. So, to send a packet, the upper layer passes the packet along with the MCL virtual address of the destination node.

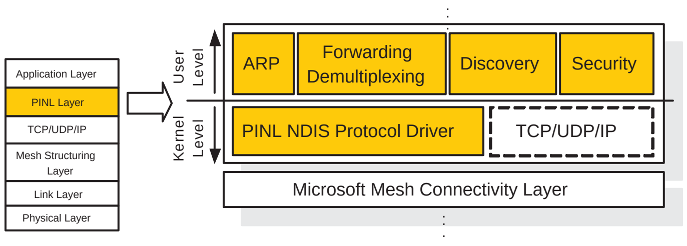


Figure 6: Detailed stack composition.

The PINL layer is implemented over the Windows XP operating system as a combination of user level and kernel level processes as shown in Figure 6.

PINL NDIS Protocol Driver

In order to interface with MCL, we have implemented a NDIS protocol driver [19] that allows the upper layers located at the user-level to directly send and receive raw MCL frames by interacting with the driver. That is, the PINL NDIS driver establishes a binding to MCL and exposes its services to the upper layers.

ARP Module

Bits	0-7	8-15	16-23	24-31
0	Hardware Address Type		Protocol Address Type	
32	Hardware Address Length	Src. Protocol Address Length	Dst. Protocol Address Length	Operation
64	Sender Hardware Address			
⋮	Sender Protocol Address			
	Destination Hardware Address			
	Destination Protocol Address			

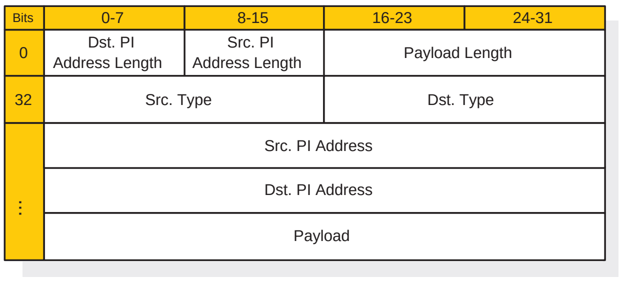
Figure 7: Extended ARP packet format.

The ARP module is mainly based on RFC 826, but slightly extended to resolve PI addresses to 48-bit MCL virtual addresses. The extended ARP packet format is illustrated in Figure 7. *Hardware Address Type* and *Protocol Address Type* identify the types of addresses of the lower (e.g., MCL) and upper (e.g., PI) layers respectively. *Hardware Address Length* gives the length in bytes of lower layer addresses. For MCL, it is set to *6*. Similarly, *Src Protocol Address Length* and *Dst Protocol Address Length* give the length of the source (sending ARP module) and destination PI address (the one that is being resolved). It is important to highlight the need for defining a different field for the length of *Src* and *Dst* protocol addresses (the PI addresses), since they can have different lengths. The operation field indicates whether the packet corresponds to a request or to a reply.

We explain the mechanism with an example by referring to our model in Figure 3. When the PINL layer at the sender stack receives a packet from a source entity *srcPI* (say *pi-a*) addressed to an unknown destination entity *dstPI* (say *pi-b*), the ARP module broadcasts an ARP request for that PI address on the local network. If the destination entity is located in the same mesh network (*AoI-1*) i.e. in the same broadcast domain, it responds with its MCL virtual address and the arp process is completed. On the other hand, if the *dstPI* corresponds to an entity in a remote mesh network (as in the case of *pi-b*), the sending ARP module eventually timeouts causing the PINL layer to send the data packet to a previously discovered Routing Ghost (*pi-rg1*), which acts as the default gateway. The discovery process of the Routing Ghost is explained later in subsection 5.2.

Forwarding-Demultiplexing

The Forwarding-Demultiplexing module forwards the data received from the upper layer (application layer) sending process and demultiplexes the incoming packets from the lower layer (either from NDIS or from standard IP in the case of overlay mode) to the correct receiving application process. The format of the PINL layer packet is illustrated in Figure 8. This is the most basic unit of communication that all entities within the InterMesh prototype use to communicate. The source and destination PI addresses are variable length with a max size of



Bits	0-7	8-15	16-23	24-31
0	Dst. PI Address Length	Src. PI Address Length	Payload Length	
32	Src. Type		Dst. Type	
⋮	Src. PI Address			
	Dst. PI Address			
	Payload			

Figure 8: PI packet format.

32 bytes. In order to send a packet to a destination entity, the sender entity addresses the packets to the 2-tuple $\{DestinationPI, DestinationType\}$. Figure 9 illustrates the flow of a packet with payload size of *40* bytes between the communicating entities. The packet will have to cross the boundary of the local mesh network only if the destination entity belongs to a remote mesh network.

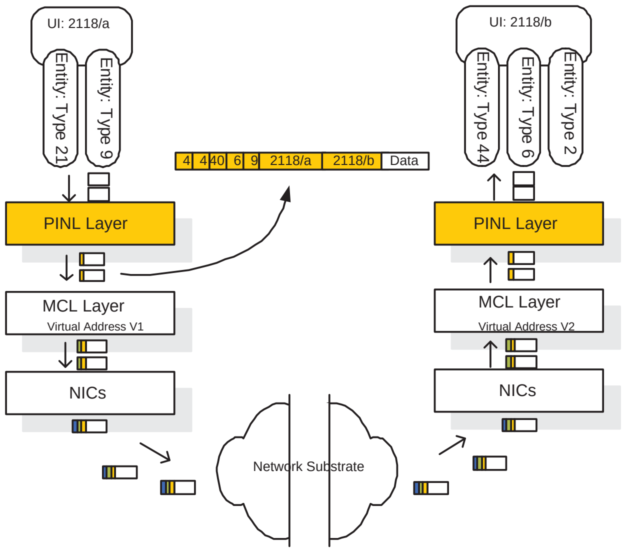


Figure 9: Packet flow from the entity $2118/a$, type *9*, to $2118/b$, type *6*.

Discovery

As mentioned previously, it is necessary that the client device discovers the GHOSTs organizing the network. These include the Naming GHOST and the Routing GHOST. The Discovery modules implements this functionality keeping the device (and thus entities) initialized at all times within the network on which the device is present. This module automatically detects the local network settings and initialize the device accordingly. For example, when a node is operating as an overlay node with a public IP address (IPv4 or IPv6), the discovery module will not try to discover a Routing GHOST since routing will be performed directly over the Internet. However, in a native mesh network, the Routing GHOST needs to be discovered.

The Discovery module implements a proactive algorithm to discover the GHOSTs. Discovery messages are broadcasted on a well-known *Type*. For example the routing GHOSTs use *Type 5678* whereas the naming GHOSTs use *Type 5679*. Once discovered, the algorithm keeps the GHOST bindings fresh by sending periodical *refresh* packets.

5.3 GHOST/SHELL model

We have implemented our own customized GHOST/SHELL model on top of the JINI [ref JINI] framework. Alternative mobile agent frameworks such as IBM aglets, JADE, and Voyager exist. Figure 10 shows how an administrator can create a GHOST and ship it to a particular SHELL (red circle). The procedure includes four

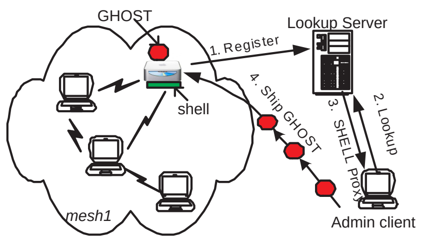


Figure 10: Mobile Agent framework (GHOST/SHELL).

steps. In step 1, the SHELL registers with a lookup server that can be located either locally or on the Internet. In steps 2 and 3, the administrator discovers the SHELL and downloads a proxy to the SHELL which is basically the service offered by the SHELL. The SHELL service allows hosting of GHOSTs provided the GHOST is shipped with valid credentials so that it can be accepted by the SHELL. Step 4 shows the GHOST migrating to the SHELL to provide a particular network service. We use this exact model in our prototype to disseminate routing GHOSTs and naming GHOSTs within the AoIs. Notice that the admin client (administrator) need not be present within the AoI to send or replace a GHOST, providing an easier and more distributed network administration model.

6 Test-bed and Performance Results

Our implemented prototype is based on the model of Figure 3. We set up two distinct WMNs having SSIDs *mesh1* and *mesh2* corresponding to *AoI-1* and *AoI-2* respectively. The WMNs are formed of pure client ad-hoc networks with no infrastructure routers. One gateway client connects the mesh to the Internet and offers the SHELL for the GHOSTs. All our entities are assigned *handle*s under the same naming authority *2118*. Back to our test-bed (Figure 3), the entity *pi-a* is identified with the tuple $\{2118/a, 5556\}$ and similarly for *pi-b* and *pi-c*. The entities *pi-rg1* and *pi-ng1* have the tuples $\{2118/rg1, 5678\}$ and $\{2118/ng1, 5679\}$ respectively. Similarly, for *pi-rg2* and *pi-ng2*.

We have measured the average application end-to-end round trip time (RTT) between the sender (*pi-a*) and the receiver (*pi-b*). Each of the results is averaged from *10,000* samples measured over 2 days. During the experiment, the 2 entities entities exchange traffic at the VoIP rate of *64* Kbps i.e. a *160* byte packet every *20* ms. *pi-b* just reflects all the received packets back to *pi-a* and packet delivery is best effort with no undertaken reliability measures. The performance results are plotted in Figure 11 showing 3 experiment scenarios. In the first scenario, the sender and the receiver belong to the same network and are communicating locally in ad-hoc mode. In the second and third scenarios, the sender and receiver belong to two different mesh networks, *mesh1* and *mesh2* respectively. The gateway nodes are connected either over the same LAN (scenario 2), or over the Internet (scenario 3). In scenario 3, one gateway node physically connects *mesh1* to the Internet at the ECE department - University of New Mexico (with IP address *129.24.27.152* and bandwidth *1* Mbit/s up, *6* Mbit/s down,), while another gateway node physically connects *mesh2* to the Internet through a home DSL connection (with IP *76.18.66.45* and bandwidth *384* Kbit/s up, *1500* Kbit/s down,). The communicating nodes as well as the gateway nodes have similar device configurations (with 2Ghz processor speed and 512 Mbytes of RAM). We can see from the results that the delay is mostly governed by the Internet delay. Several factors add to the depicted RTT delay including the implementation of a large part of the PINL layer at the user level. Note that the change in average RTT was negligible when the traffic exchange rate between the entities increased from *32* kbps to *64* kbps. This is intuitive since the end-2-end available bandwidth is much larger than both traffic rates. The results depicted are very encouraging and comparable to RTT delays over current IP networks.

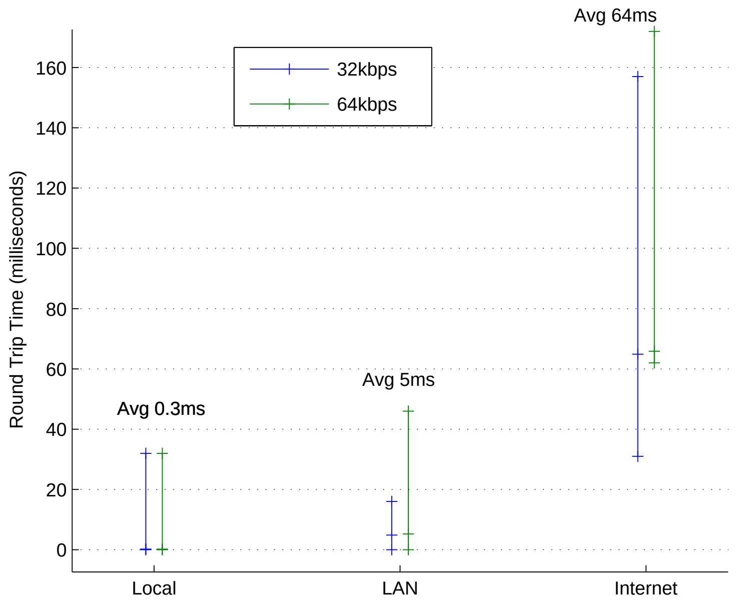


Figure 11: Application end-2-end Round Trip Times between two entities when communicating over the same mesh network (Local) or over different mesh networks that are connect through a LAN or through the Internet.

7 Related Work

We classify related work into two main categories. The first class of proposals addresses clean-slate architectures, while the second class of proposals is based on overlays. The distinction between the two classes is relevant to this work in terms of how much intelligence is injected into the network, and consequently, to what extent do the proposals respect the end-to-end argument [24].

We start by relating our work to the FARA [8] class of architectures. The basic idea in FARA is to the separate the network address (attachment point) from the entity's address by a "red line" to allow evolution of functionality above and below the line and to foster mobility. M-FARA [22] is an instance of the FARA architecture. The InterMesh prototype shares a lot with M-FARA: the idea of viewing the inter-network as a collection of private address realms is related to (AoIs), the "core" construct is also similar. Besides, an entity's FDdown represents the actual routing GHOST within the AoI. FDup information is obtained through the entity's initialization process. Communication between devices within the same realm needs no FDs, whereas communication between processes in different realms requires FDup, FDdown information of realm boundary agents (our Ghosts). However, InterMesh assumes globally unique and persistent identifiers for entities and designs the network starting from the entity down.

Recent work by Vicente etal. [29, 10] addresses the problem of creating distributed services over a flexible overlay of wireless mesh networks. They take a network-centric approach that uses virtualization of the mesh nodes to implement overlay services based on concepts from the PlanetLab framework. The paper is rather introductory and we assume it achieves the inter-connection of different mesh networks using IP (because of evolving PlanetLab to WLAN networks). Another paper by Pangalos [20] proposes an inter-networking framework between mobile and broadcast networks. A gateway in each network is used to provide an interface to the network and the Internet backbone is used to inter-connect the gateways via IP. In [1], an overlay structuring approach is proposed to inter-connect multi-homed mesh networks using an inter-domain routing scheme. Mesh Internet gateways from each mesh network organize into a full mesh overlay by exchanging IP addresses over a multicast infrastructure (overlay) to inter-connect the networks. Also, the access points inside each mesh network (not the Internet gateways) locally organize into an overlay using the wireless links.

In [27], an IP-based heterogeneous framework is proposed to transparently interconnect 802.11 and Bluetooth networks. The framework is implemented as a *virtual interface* between the layers 2 (MAC) and 3 (IP). The virtual interface behaves similar to a 802.x bridge; i.e., it interconnects 802.x networks and hides the heterogeneity of the used devices from the upper layers. The proposed framework only considers the interconnection of local 802.x networks; therefore, it does not support vertical handoff, which should be implemented separately with some pre-post registration protocol on upper layers to support vertical handoff and maintain ongoing communication sessions.

As to the proposals that deal with overlay routing, we mention I3 [26], Tapestry [31, 30], and RON [6]. These architectures introduce more intelligence into the network in terms of identifying entities, locating them and routing traffic to them.

8 Conclusion

To efficiently embrace the characteristics of emerging access networks, and to broaden the user's innovation space clean-slate architectural approaches are being pursued towards designing the future Internet. We have previously introduced a general architectural vision for a possible future Internet which we call the Transient Network Architecture from which InterMesh is instantiated. In this paper, we have presented the InterMesh platform that achieves convergence of heterogeneous mesh networks through a novel PINL layer providing a seamless service to individual network entities. We have identified the key concepts behind the InterMesh architecture and presented an interesting prototype implementation that can coexist with today's Internet. As part of our current work, we are extending InterMesh to include generic mobile ad-hoc networks (MANETs) and wireless sensor networks (WSNs). We are also investigating a completely distributed naming system formed of the naming GHOSTs that organize into a resilient P2P network.

9 Acknowledgements

The work reported in this paper is part of the Transient Network Architecture project funded by the National Science Foundation NSF under the Future Internet Design (FIND) Grant CNS-0626380. More detailed technical

reports can be located at the project website http://hdl.handle.net/2118/tna.

References

[1] an inter-domain routing protocol for multi-homes wireless mesh networks.

[2] The handle system. http://www.handle.net.

[3] Self-organizing neighborhood wireless mesh networks.

[4] The transient network architecture. http://hdl.handle.net/2118/tna.

[5] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, March 2005.

[6] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Symposium on Operating Systems Principles*, pages 131–145, 2001.

[7] Steven M. Bellovin, David D. Clark, Adrian Perrig, and Dawn Song. A clean-slate design for the next-generation secure internet, March 2005. This is the report of an NSF workshop held in July, 2005.

[8] David Clark, Robert Braden, Aaron Falk, and Venkata Pingali. Fara: reorganizing the addressing architecture. In *FDNA '03: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, pages 313–321, New York, NY, USA, 2003. ACM Press.

[9] George F. Coulouris and Jean Dollimore. *Distributed systems: concepts and design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1988.

[10] Gang Ding, John Vicente, Sanjay Rungta, Dilip Krishnaswamy, Winson Chan, and Kai Miao. Overlays on wireless mesh networks: Implementation and cross-layer searching. to appear in 9th IFIP/IEEE International Conference on Management of Multimedia and Mobile Networks and Services, 2006.

[11] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 114–128, New York, NY, USA, 2004. ACM Press.

[12] Timon C. Du, Eldon Y. Li, and An-Pin Chang. Mobile agents in distributed network management. *Commun. ACM*, 46(7):127–132, 2003.

[13] S. M. Faccin, C. Wijting, J. Kenckt, and A. Damle. Mesh wlan networks: concept and system design. *Wireless Communications, IEEE*, 13(2):10–17, April 2006.

[14] Amal Graafstra. How radio-frequency identification and i got personal. *Spectrum, IEEE*, 44(3):20–23, March 2007.

[15] Henry Jerez, Joud Khoury, and Chaouki Abdallah. A mobile transient network architecture, 2006. Pre-print available at https://dspace.istec.org/handle/1812/55.

[16] V. Kawadia and P. R. Kumar. A cautionary perspective on cross-layer design. *Wireless Communications, IEEE*, 12(1):3–11, 2005.

[17] M.R. Kibria and A. Jamalipour. On designing issues of the next generation mobile network. *Network, IEEE*, 21(1):6–13, January 2007.

[18] R. Moskowitz, P. Nikander, and P. Jokela. Host identity protocol. Internet Draft, work in progress, draft-moskowitz-hip-09.txt, IETF, 2004.

[19] Walter Oney. *Programming the Microsoft Windows Driver Model, Second Edition*. Microsoft Press, Redmond, WA, USA, 2002.

[20] Paul Pangalos, Juan Martin De La Torre Velver, Mohammad Dashti, Ali Dashti, and Hamid Aghvami. Confirming connectivity in interworked broadcast and mobile networks. *Network, IEEE*, 21(2):13–20, March 2007.

[21] Charles E. Perkins. RFC 3220:ip mobility support for ipv4, January 2002.

[22] V Pingali, A Falk, T Faber, and R Braden. M-fara prototype design document. USC Information Sciences Institute, 2003.

[23] Jorge Piovesan, Chaouki Abdallah, Herbert Tanner, Henry Jerez, and Joud Khoury. Resource allocation for multi-agent problems in the design of future communication networks. Technical Report EECE-TR-07-001, University of New Mexico, April 2007. [online]: http://hdl.handle.net/2118/jp_tech_07.

[24] J. H. Saltzer, D. P. Reed, and D. D. CLark. End-to-end arguments in system design. *ACM TOCS*, 2(4):277–288, November 1984.

[25] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, August 2000.

[26] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *IEEE/ACM Transactions on Networking*, 12(2):205–218, April 2004.

[27] Patrick Stuedi and Gustavo Alonso. Transparent heterogeneous mobile ad hoc networks. In *MOBIQUITOUS '05: Proceedings of the The Second Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, pages 237–246, Washington, DC, USA, 2005. IEEE Computer Society.

[28] S. Sun. Establishing persistent identity using the handle system. Tenth International World Wide Web Conference, May 2001.

[29] John Vicente, Sanjay Rungta, Gang Ding, Dilip Krishnaswamy, Winson Chan, and Kai Miao. Overmesh: network-centric computing. *IEEE Communications Magazine*, 45(2):126–133, February 2007.

[30] B. Zhao, L. Huang, A. Joseph, and J. Kubiatowicz. Rapid mobility via type indirection. San Diego, CA, February 2004. Proceedings of the 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04).

[31] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, January 2004.