

Routing through the Mist: Design and Implementation

Jalal Al-Muhtadi* Roy Campbell[∇] Apu Kapadia[†] M. Dennis Mickunas[∇] Seung Yi

Department of Computer Science,
University of Illinois at Urbana-Champaign,
{almuhtad, rhc, akapadia, mickunas, seungyi}@uiuc.edu

ABSTRACT

Ubiquitous computing is poised to revolutionize the way we compute and interact with each other. However, unless privacy concerns are taken into account early in the design process, we will end up creating a very effective distributed surveillance system, which would be a dream come true for electronic stalkers and “big brothers.” We present a protocol, which preserves the privacy of users and keeps their communication anonymous. In effect, we create a “mist” that conceals users from the system and other users. Yet, users will still be able to enjoy seamless interaction with services and other entities that wander within the ubiquitous computing environment. This report motivates our work, describes our approach, and details Mist’s design and implementation.

1. Introduction

The major advances in distributed systems and mobile computing have converged to enhance global interconnectivity. This has fueled the idea of ubiquitous computing and active information spaces where users can access services, run programs, utilize resources, and harvest computing power anytime and anywhere.

Physical spaces augmented with sensors and actuators that can locate users, detect their presence, and track their whereabouts will be commonplace in this new and exciting computing paradigm. These sensors will play a major role in bridging the virtual computing world with the physical world and boosting the productivity of users and the availability of computing resources. However, these very features could severely threaten the privacy of users. For instance, the mentioned services can be exploited by intruders, malicious insiders, or even “curious” system administrators to track or electronically stalk particular users. Although encryption provides confidentiality by hiding information flowing through communication channels from eavesdroppers (e.g., an insider or a system administrator), an eavesdropper can still gather the network addresses or physical locations of the communicating parties. The lack of privacy in today’s networks and distributed systems is well-documented [3]. Similar concerns arise for ubiquitous computing environments [2].

In this work, we aim to design and implement a privacy protocol that allows users of a ubiquitous computing environment to roam and communicate freely while preserving their privacy. The privacy protocol prevents insiders, system administrators and even the system itself from tracking users and detecting their physical location. Yet, the system will enable users to communicate with other users and access computing resources in an authenticated manner without disclosing the users’ physical locations or whereabouts. Further, users will be able to configure the level of privacy they wish to enjoy through the use of a user interface running on their mobile devices (e.g., mobile phone, laptop, PDA.) We plan to achieve this by allowing the ubiquitous computing environment to maintain sensors that can detect the presence of users in a room, but without the ability to positively identify the users. Combined with our routing protocol, this creates a “mist” through which users can communicate privately. In our system, we introduce a hierarchy of “*Mist Routers*” that perform “*handle-based routing*” to preserve privacy and hide information about the original source and the final destination. In short, we refer to this hierarchy as a “*Mist Hierarchy*.” The handle-based routing combines hop-to-hop routing based on handles with limited public-key cryptography to preserve privacy from eavesdroppers and traffic analyzers. Positive authentication and registration of users can be achieved at a higher level in the hierarchy, making it harder to infer the user’s current location.

* Jalal Al-Muhtadi is funded by a grant from the National Science Foundation, NSF CCR 0086094 ITR.

[∇] These authors are supported by grants from the National Science Foundation, NSF CCR 00-86094 ITR, NSF EIA 98-70736, NSF EIA 99-72884 EQ, and NSF CCR 00-86094.

[†] Apu Kapadia is funded by the Department of Energy High Performance Computer Science Fellowship through Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Sandia National Laboratories.

Our privacy scheme will be deployed in the Gaia system. The Gaia system [1][4] is an operating system being developed at the Department of Computer Science, University of Illinois at Urbana-Champaign to provide an infrastructure in the form of core services over which active information spaces can be constructed.

Section 2 describes our protocol in detail. Section 3 discusses the system's use cases. Section 4 presents our system design from an object-oriented perspective, highlighting the important classes in our implementation and illustrating their relationship with each other. Section 5 describes the user interfaces in our system. Section 6 explains how the work is divided. Finally, Section 7 concludes.

Assumptions

Privacy is a fuzzy term that is often overloaded to mean a large variety of things. Therefore, before proceeding any further, it is important to clarify the scope of user privacy that we strive to achieve in a ubiquitous computing environment. Our goal is to achieve the following:

1. *Location privacy:* Neither the system nor the users of the system will be able to know the exact physical location of a user, unless that user decides to disclose such information or if another person physically sees that user at that location.
2. *Anonymous connections:* If two parties decide to communicate with each other, then other users in the system will not know who the communicating parties are, unless one of the communicating endpoints decides to disclose such information.
3. *Confidentiality:* If both endpoints of a communication agree, they can make the content of their communication confidential, such that neither the system nor other users in the system can read the contents of the communication.

We assume that a PKI exists for users of the ubiquitous computing environment and for the Mist Routers in the system. However, we do not assume the existence of a third party that can be trusted to safely store sensitive information about users, like their physical location for instance.

2. System Design

In our system, Mist Routers are deployed in a hierarchical fashion. The system administrator will be able to start CORBA Mist Routers on different machines. A System GUI will be able to detect all running Mist Routers, and display them graphically. The System GUI will provide the administrator with the ability to develop a hierarchy of Mist Routers and to have a "central control" to the entire Mist system. Users connect directly to one of the leaf level Mist Routers, which we call "Portals" (implemented as a subclass of a Mist Router). Through a Portal, a user (or the user's device) sets up a "Mist Circuit" upwards in the hierarchy. A Mist Circuit is a handle-based virtual circuit between the user and a special Mist Router, which we call a "Lighthouse." Since the handles for the virtual circuit is set up on a hop-by-hop basis, unless enough Mist Routers in the path collude, none of the intermediate Mist Routers can deduce the two ends of the virtual circuit. A user uses Mist Circuits to contact one of the higher level Mist Routers who is willing to serve as a contact point for that user. This contact point will only have partial information on how to route to that user. We refer to this contact point as a "Mist Lighthouse" for that user.

Sections 2.1 and 2.2 discuss the arrangement of Mist Routers in a hierarchy and the establishment of Mist Circuits respectively. Section 2.3 describes how Lighthouses can be used to locate users in the system while preserving their privacy. Finally Section 2.4 describes how a communication channel can be established by two peers while maintaining their privacy.

2.1 Mist Hierarchies

Mist Routers are key elements in our system. These CORBA objects conceal the identity and location of communicating parties by rerouting packets among themselves using hop-to-hop handle-based routing (which we describe in more details in Section 2.2.) We envision that Mist Routers will be deployed in hierarchical clusters organized along physical space divisions, called domains. The hierarchical organization of Mist Routers would enhance the system's flexibility and scalability, allowing it to be easily deployed over multiple domains.

Initially, a Mist Hierarchy needs to be agreed upon and constructed between the different physical space domains that are willing to cooperate and provide privacy for users roaming in them. Meeting this requirement should not be a problem; this is because most physical spaces are organized into hierarchies by nature (as illustrated in Figure 1, for example). Further, such hierarchies can be constructed dynamically in a fashion similar to the way multicast protocols construct source-based trees [6] or Core Based Trees (CBT), which builds a "shared tree" rooted at a core router [5]. Allowing these hierarchies to spread over multiple domains make it harder for corrupt Mist Routers to collude.

As illustrated in Figure 1, Mist Routers at the leaves of the hierarchy represent "Portals." Portals are viewed as the gateways that bridge the virtual world to the physical one. In other words, they are connection points where users of an active in-

formation space can connect to the system. Portals are represented by a variety of hardware that can include a fixed workstation, a sensor, an access point for wireless devices, and an RF transceiver.

As previously indicated, the original objective of an active information space is to allow seamless interactions between the various virtual and physical entities in the space. Therefore, there should be a mechanism over which these interactions can take place in spite of the existence of this mist that blurs the true identities of users and hide their physical locations. Therefore, to access the system, to communicate with others, and to use available resources while maintaining privacy, user Alice, say, has to register herself in the system as shown in Figure 2. The registration takes place through Alice’s client device. The device talks directly to one of the available Portals in the surrounding physical space. The mechanism involves designating a special Mist Router for every user of the system. This special Mist Router will be referred to as a “Lighthouse” for that user. For example, a Lighthouse for Alice is a Mist Router that is an ancestor of the Portal that Alice is connecting to. Alice’s Lighthouse will have knowledge of her true identity as well as partial knowledge on how to route to Alice. However, it does not know the exact physical location of Alice. Whereas the Portal knows the exact physical location of Alice, but does not “realize” that this is actually Alice and does not know who Alice’s Lighthouse is. Going back to the registration process illustrated in Figure 2, Alice’s device sends a registration request to the nearby Portal. The Portal will reply back with a list of its ancestral Mist Routers that exist at a higher level within the Mist Hierarchy and are willing to act as a Lighthouse for the user. A trusted third party can be used to vouch for the trustworthiness of some of these Mist Routers, particularly the ones that exist near the root of the hierarchy, since these Mist Routers can be accessible from different spaces. This vouching process is similar to how certificate authorities vouch for other parties on the Internet.

User Alice, through her client device, can customize the amount of privacy she wishes to enjoy by selecting a Mist Router at a suitable height in the hierarchy to be her Lighthouse. Selecting a Lighthouse is a tradeoff between performance and privacy. Choosing a Mist Router that is closer to the root of the hierarchy provides better privacy because less information is inferred about the actual physical location of Alice, and the extra rerouting provides better concealment. Whereas selecting Mist Routers closer to the Portal helps performance by limiting the number of reroutes but decreasing the level of privacy. To illustrate, in Figure 1, Alice decides to designate the Computer Science building’s Mist Router as her Lighthouse. This information implies that Alice is currently located somewhere in the Computer Science building. Bob, on the other hand, chooses the campus Mist Router as his Lighthouse. This implies that he physically can be anywhere in campus. Ultimate privacy can be achieved when a user chooses the hierarchy’s root as its Lighthouse.

Upon the selection of a suitable Lighthouse by Alice, we establish what we refer to as a “Mist Circuit” between Alice and the selected Mist Router. We discuss Mist Circuits in more detail in the Section 2.2. In any case, the Mist Circuit will make it possible for Alice’s Lighthouse to authenticate Alice while hiding her exact physical location, and, at the same time, hiding her identity and her selected Lighthouse from the Portal she is connected to.

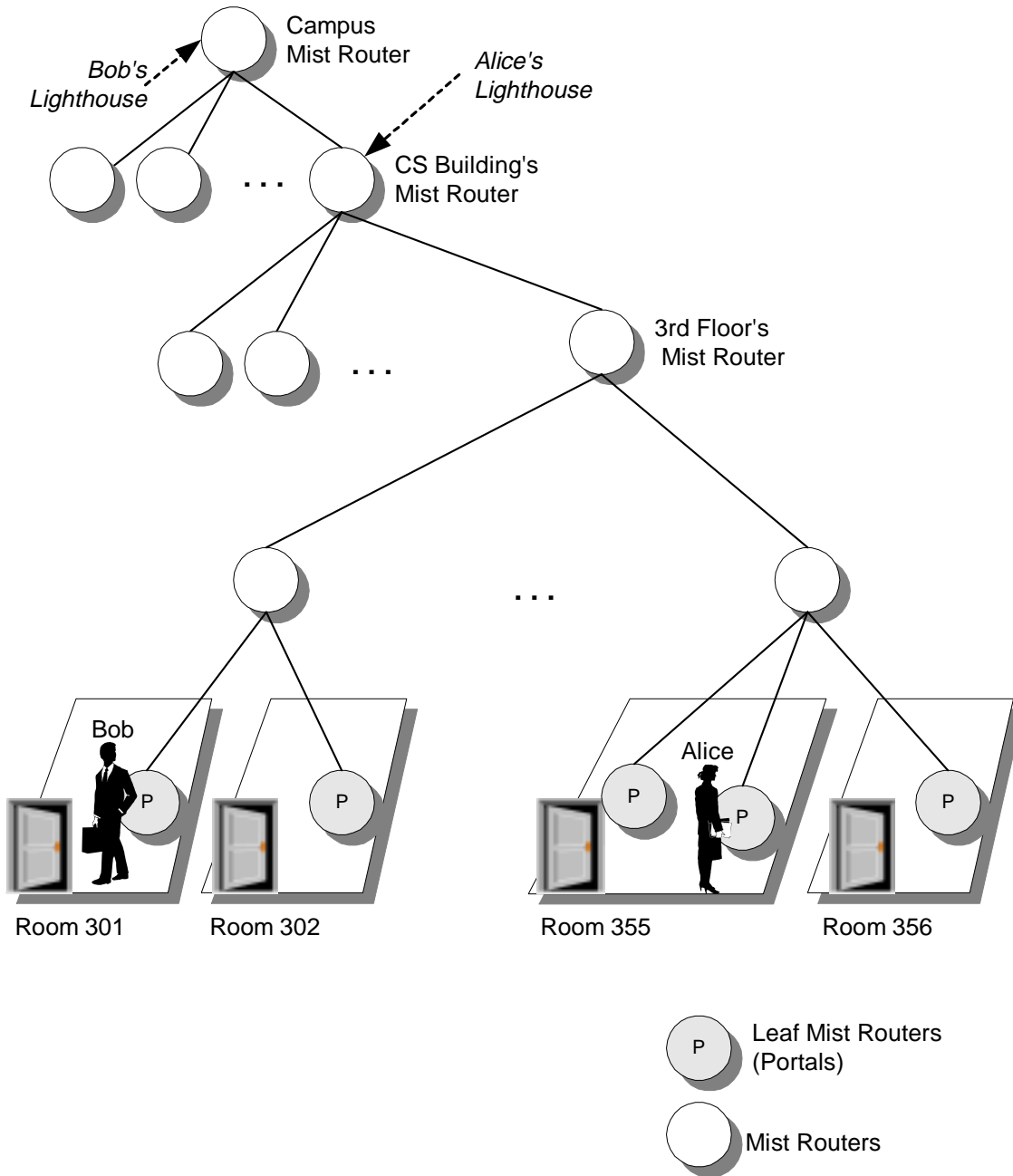


Figure 1: The Mist Hierarchy

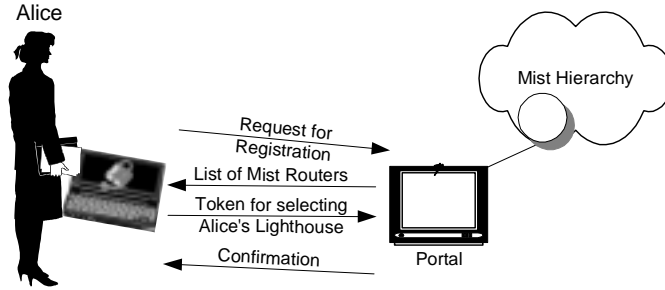


Figure 2: Registering in the System

2.2 Mist Circuits

Mist Circuits employ hop-to-hop, handle-based routing to send data packets back and forth between the source and destination through the mist. Combining this routing with limited public-key encryption allows data packets to be successfully routed through the mist while providing a higher degree of privacy and concealment. This prevents intermediate nodes from recognizing the identities of the actual endpoints or their physical location.

Recall that we establish a Mist Circuit between the user and its selected Lighthouse so that the user can reveal its true identity and authenticate it at the Lighthouse without disclosing physical location information. In this section we describe how a Mist Circuit is setup and used.

We go back to the example of Alice registering in the system. Her Portal fulfills her request for registration by replying back with a list of ancestral Mist Routers that are willing to act as Lighthouses. The list returned contains two pieces of information for each Mist Router. Each entry will contain an ID that uniquely identifies the Mist Router and a digital certificate for that Mist Router. The digital certificate can be issued by some trusted third party. The certificate could contain information about the how “high” in the Mist Hierarchy the associated Mist Router is. In other words, the list is of the form:

$\langle \text{Mist Router}_1, \text{Certificate}_1 \rangle,$

$\langle \text{Mist Router}_2, \text{Certificate}_2 \rangle,$

...

User Alice selects a suitable Mist Router, which she does not disclose to the Portal. To establish a Mist Circuit, Alice generates a Mist Circuit establishment packet. The general format of Mist packets are illustrated in Figure 3. The ‘Handle ID’ field represents a handle that is unique per Mist Router that helps identify the next hop on the packet’s route. A value of 0 in this field indicates that no value is assigned yet. How the handle is used is described later in this section. The ‘direction’ field is a single bit that specifies whether the packet is going upwards (toward the Lighthouse) or downwards (toward the Portal) in the hierarchy. The ‘packet type’ identifies the type of the packet, which tells the intermediate Mist Routers how they should handle the packet.

Assuming that Alice selects the Mist Router ‘Z’ in Figure 6 as her Lighthouse, then Alice’s Mist Circuit establishment packet will contain ‘0’ for the handle ID and ‘U’ in the direction field, indicating that this packet is going upwards. The type field will contain a value indicating that this is a Mist Circuit establishment packet. The payload will consist of the Message M :

$$M = E_{\text{public_key}_Z}(\text{Alice} // \text{TS} // K_{\text{session}} // \text{TKN} // \text{PP})$$

Where:

// stands for concatenation.

Alice: Alice’s unique ID in the active information space

TS: A timestamp to prevent replay attacks.

K_{session}: A random session key to encrypt further communication between the user and her or his Lighthouse. It is also used to add some additional randomness into the encrypted message.

TKN: A token to be presented to the user’s lookup service.

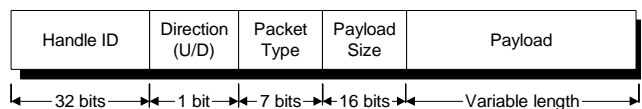


Figure 3: General Format for Mist Packets

0	U	MIST CIRCUIT EST.	Payload size & payload
---	---	-------------------	------------------------

$$M = E_{\text{public key of } Z} (\text{Alice} \parallel TS \parallel K_{\text{session}} \parallel TKN \parallel PP)$$

$$\text{Payload} = M \parallel S_{\text{Alice}}(M)$$

Figure 4: Alice's Mist Circuit Establishment Packet

254	D	MIST COMM.	Payload size & payload
-----	---	------------	------------------------

$$\text{Payload} = E_{K_{\text{session}}} ("Success", TS_2)$$

Figure 5: Registration Confirmation Packet

Details about the user's lookup service and the contents of this token are given in Section 2.3.

E_k : Means encrypt using the key 'k'.

PP : A predetermined "fixed" phrase. In our current implementation, we are using the string "Mist Circuit Establishment Message." The use of this will be described below.

The actual payload is:

$$\text{Payload} = M \parallel S_{\text{Alice}}(M),$$

where $S_{\text{Alice}}(M)$ indicates Alice's digital signature over M.

The contents of the Mist Circuit establishment packet are shown in Figure 4. Alice then transmits this packet to her Portal, without informing the Portal of the selected Lighthouse. Portals will maintain a table that is referred to as the "Presence Table." Since the Portal detects nearby people without positively identifying them, whenever a new person is detected, he or she is entered into the Portal's presence table as an "anonymous" person. Additionally, the Portal assigns for every user a handle ID that is unique within that table only. So in the scenario depicted in Figure 6, Alice is represented as "Anon-1" and is assigned a handle ID of 10, say. If other users exist in the same physical space and the Portal is able to communicate with them, then similarly, they will be entered into the presence table. The "link" field should contain a value that identifies the

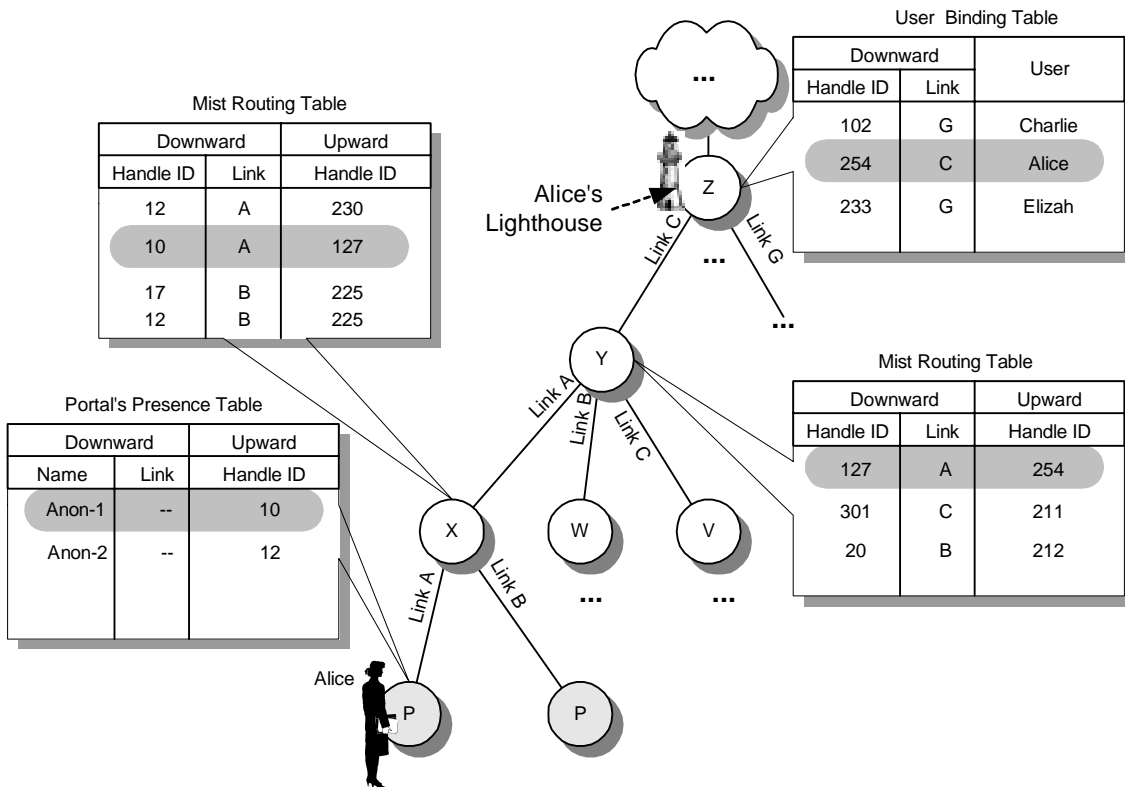


Figure 6: Mist Circuit Setup

network link or port number over which the Portal can communicate with the corresponding user. We assume that if a Portal supports communication with more than one physically present user, then it should be able to recognize which user sent a particular packet. Upon receipt of the Mist Circuit establishment packet from Alice, the Portal will replace the value in the packet's handle ID field with the handle ID that was assigned to Alice in the presence table, which is 10 in the example shown. Next the Portal will transmit the modified packet "upward" to its parent Mist Router.

From now on, upon receiving the circuit establishment packet every intermediate Mist Router will attempt to decrypt the encrypted portion of the payload using its private key. If the decryption fails, (the predetermined phrase can be used to indicate whether or not the decryption failed) then the Mist Router will infer that this packet is not meant for it. Instead, the packet has to be passed upward to its parent. Each Mist Router will maintain a "Mist Routing Table." This table will associate handle IDs used over downward connections with handle IDs that will be used on the upward connection. Note that within the downward column of the Mist Routing Table, the combination of Handle ID and link ID is unique per Mist Router, whereas, within the upward column the handle ID value is unique per Mist Router. The current Mist Router does a quick lookup on its Mist Routing Table to see if it has an entry for the handle ID and the link over which it received the packet. If it does not, it creates one, and associates an upward handle ID for it. The Mist Router then substitutes the value of the packet's handle ID with the newly assigned value and passes the message to its parent. The process is repeated for every intermediate Mist Router.

On the other hand, if a Mist Router successfully decrypts the encrypted portion using its private key, then this indicates that the user actually chose the current Mist Router as his or her Lighthouse. All Mist Routers that are willing to act as Lighthouses for users should maintain a 'User Binding Table' as shown in Figure 6. The Mist Router can now authenticate the user by verifying his or her signature and checking the freshness of the timestamp. The handle ID and the downward link above which it was used will be stored in the User Binding Table, along with the actual ID of the user.

Figure 6 shows the actual entries in the presence, routing and binding tables when user Alice registers and chooses 'Z' as her Lighthouse. The shaded entries in the figure represent Alice's entries. In effect, this process has established a "circuit" over which Alice can communicate with her Lighthouse securely. Note that while Alice's Lighthouse can infer that Alice exists somewhere in the hierarchy underneath Mist Router 'Y', the exact location cannot be determined unless enough Mist Routers agree to cooperate. Therefore, the longer the path between Alice and her Lighthouse the more "private" her location becomes.

To complete the Mist Circuit establishment, the Lighthouse confirms the registration of Alice by sending back a reply packet. The format of this reply is shown in Figure 5. For the example shown, the handle ID will be set to 254, because this is the value bound to Alice. The packet should be sent downward (D). The packet type is set to "MIST COMMUNICATION" which indicates that intermediate Mist Routers should not attempt to decrypt the contents, rather, they should just route it to the next hop.

$K_{session}$ is the session key between the Mist Router 'Z' and Alice that was transmitted through the Mist Circuit establishment packet. Note that to improve performance from this point on, we use symmetric encryption to achieve confidentiality between the user and the chosen Lighthouse. TS_2 is a timestamp to prevent replays. This packet can now be routed back to Alice in a manner similar to what was described above. Now Alice can communicate securely with her Lighthouse while preserving her privacy. In the next section, we describe how other entities in the ubiquitous computing environment can communicate with Alice.

2.3 Locating Users

Once the Mist Circuit-Setup has been completed, the Lighthouse Mist Router acts on behalf of the end-user. All communication with the user will take place through its Lighthouse, since only the Lighthouse knows how to route packets to the user. However, we first need to locate the current Lighthouse for a particular user. Only then can one communicate with the user. Locating users involves the *registration* of <user, Lighthouse> pairs, and the *lookup* of <user, Lighthouse> pairs.

2.3.1 LDAP Servers

RFC 1777 describes the Lightweight Directory Access Protocol (LDAP). In essence, users can register with LDAP servers, which can consequently be looked up with a subset of these attributes. Mist users will have a unique LDAP *Distinguished Name (DN)*. Mist users can look up information about other Mist users either based on their DN's, or on their attributes. For example, one could look up a user based on the last name and university, "Doe from University of Illinois." Once a user has been located, the attribute corresponding to the current Lighthouse can be retrieved. We plan on using an existing open-source implementation of LDAP to serve our purpose.

2.3.2 Security issues

We would like to prevent malicious Lighthouses or attackers from falsely registering users with them. To achieve this, the user constructs a special token (*TKN*) signed by the user’s private key. This token will contain a timestamp and the unique ID of the chosen Lighthouse. This token is propagated to the Lighthouse during the Mist Circuit setup as described in Section 2.2. Once the Mist Circuit has been established, the Lighthouse presents this token to the LDAP Server. The updates will be secure, and cannot be forged or replayed by an attacker. If the timestamp has already been seen before, or if it has expired, the token will be discarded. Naturally, if the signature cannot be verified, the token is also discarded. The format of this token, *TKN*, is as follows:

$$TKN = (User\ ID \ || \ Lighthouse\ ID \ || \ Timestamp \ || \ S_{User}(User\ ID \ || \ Lighthouse\ ID \ || \ timestamp))$$

This tells us that *TKN* contains the user ID, the Lighthouse ID (this could be the DNS name) and the timestamp are signed by the user’s private key. *TKN* contents do not need to be encrypted because the contents are already known by the Lighthouse anyway. Hence, only integrity of this message, not confidentiality, needs to be guaranteed.

2.4 Mist Communication Setup

Once we have located the Lighthouse for a particular user, we need to set up a communication channel through it. In our system we assume that both users in the communication setup have established their own Mist Circuits and are both registered with their respective Lighthouses. Communication will now take place through the two Lighthouses. We will use the notation $Lighthouse_X$ to mean “Lighthouse of User X.” Let us say that Bob is trying to initiate communication with Alice. Bob and Alice are registered with $Lighthouse_{Bob}$ and $Lighthouse_{Alice}$ respectively.

Bob generates the following message for its Lighthouse:

$$M_{Lighthouse} = E_{K_{Session}}(COMM_SETUP \ || \ Alice's\ ID\ or\ attributes \ || \ TS)$$

Note that all messages in this section are actually the payload of Mist Communication packets. Since handles have been set up in both directions during the Mist Circuit Setup phase, this message will travel up to $Lighthouse_{Bob}$. Note that intermediate Mist Routers are never aware of the user’s Lighthouse. When the message arrives at $Lighthouse_{Bob}$ it is able to uniquely determine that the message is from Bob based on the arriving handle. It decrypts the message with session key $K_{Session}$ and determines from the *COMM_SETUP* message type that communication must be set up with Alice. If Alice’s ID is included then the lookup for $Lighthouse_{Alice}$ is straightforward. However, if Bob specifies attributes, then $Lighthouse_{Bob}$ must perform a lookup based on these attributes. If a unique match for Alice is found based on these attributes, $Lighthouse_{Bob}$ can determine Alice’s ID. In both cases, Alice’s ID is used to lookup $Lighthouse_{Alice}$. The timestamp *TS* is used to prevent replay attacks.

$Lighthouse_{Bob}$ uses asymmetric key encryption with $Lighthouse_{Alice}$ to determine $Lighthouse_{Alice}$ ’s handle for Alice. Since this is straightforward, we avoid the details of this communication. We will call this the destination handle for Alice, or $dest_handle_{Alice}$. In Figure 7, we can see that $Lighthouse_{Bob}$ determines $dest_handle_{Alice} = 254-C$. $Lighthouse_{Bob}$ then generates a unique handle that Bob can use to address Alice. We will call this handle src_handle_{Alice} . In Figure 7 $src_handle_{Alice} = 689$. $Lighthouse_{Bob}$ sets up a binding of the form $\langle src_handle_{Alice}, dest_handle_{Alice}, Lighthouse_{Alice} \rangle$. In Figure 7 we can see the binding $\langle 689, 254-C, Y \rangle$. We call this a Mist Communication Binding. All messages from Bob that arrive for src_handle_{Alice} (689) will be tunneled to $Lighthouse_{Alice}$ (Y) and indexed with $dest_handle_{Alice}$ (254-C). Similarly, $Lighthouse_{Bob}$ will supply the handle for Bob to $Lighthouse_{Alice}$ that will set up a binding of the form $\langle src_handle_{Bob}, dest_handle_{Bob}, Lighthouse_{Bob} \rangle$ in the same way. In Figure 7 we can see this binding as $\langle 412, 100-A, X \rangle$.

Once $Lighthouse_{Bob}$ and $Lighthouse_{Alice}$ have setup their bindings, they need to inform Bob and Alice of the $src_handles$. $Lighthouse_{Bob}$ sends src_handle_{Alice} to Bob in the following message:

$$M_{Handle} = E_{K_{Session}}(HANDLE_MSG \ || \ Alice's\ ID \ || \ src_handle_{Alice} \ || \ TS)$$

In Figure 7 this message corresponds to “For Alice use 689.” Similarly, $Lighthouse_{Alice}$ sends src_handle_{Bob} to Alice.

Now Bob can send $Lighthouse_{Bob}$ messages destined to Alice by simply using src_handle_{Alice} (689), and Alice can send $Lighthouse_{Alice}$ messages destined for Bob using src_handle_{Bob} (412). This is done to hide Alice’s identity from intermediate routers. These intermediate routers are hence unaware of *both* the endpoints of the communication. To communicate with Alice, Bob constructs messages of the following form, where ‘M’ is the message for Alice:

$$M_{For\ Alice} = (COMMUNICATION_MSG \ || \ src_handle_{Alice} \ || \ M)$$

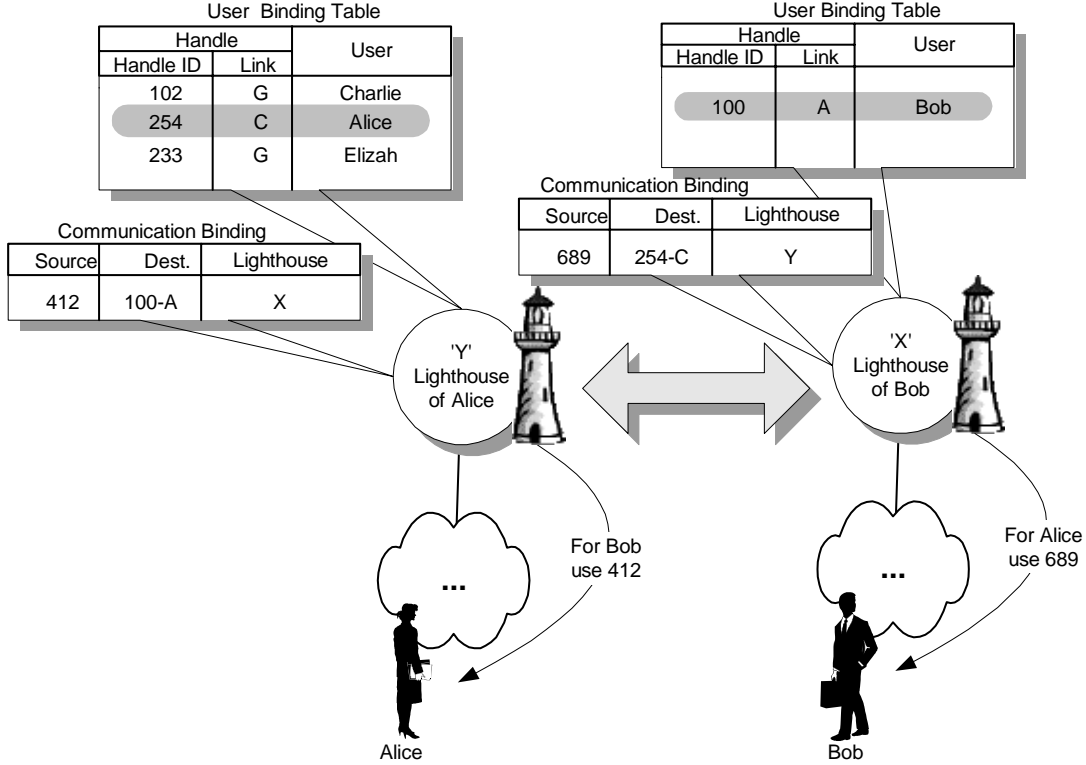


Figure 7: Mist Communication Setup

This message will propagate upstream until it reaches $\text{Lighthouse}_{\text{Bob}}$, which uses $\text{src_handle}_{\text{Alice}}$ (689) to determine $\text{Lighthouse}_{\text{Alice}}$ (Y) and $\text{dest_handle}_{\text{Alice}}$ (254-C). Note that the Message passes in the clear, and the use of handles does not disclose the endpoints of the communication. Alice and Bob are now free to choose an end-to-end encryption scheme if desired. Using this method, there is no duplication of encryption by the Mist. Once $\text{Lighthouse}_{\text{Alice}}$ is determined, the Message M needs to be forwarded to $\text{Lighthouse}_{\text{Alice}}$. $\text{Lighthouse}_{\text{Bob}}$ sends the following message to $\text{Lighthouse}_{\text{Alice}}$. We use the subscript of “crossing” to suggest that the message is crossing over from one Lighthouse to another.

$$M_{\text{Crossing}} = (\text{dest_handle}_{\text{Alice}}, M), \text{ e.g., } (254\text{-C}, M)$$

When $\text{Lighthouse}_{\text{Alice}}$ receives this message, it uses this $\text{dest_handle}_{\text{Alice}}$ to route message M to Alice. Similarly, $\text{Lighthouse}_{\text{Alice}}$ can route messages to Bob using:

$$M_{\text{Crossing}} = (\text{dest_handle}_{\text{Bob}}, M), \text{ e.g., } (100\text{-A}, M)$$

Note that these “crossing” messages between Lighthouses are not the Mist communication messages described before. The Lighthouses use their own packet formats to exchange the crossing messages.

2.5 Security issues

Here we discuss how the described scheme achieves location privacy for Alice and Bob. We also present an enhancement that adds anonymity to Alice and Bob’s connection.

2.5.1 Privacy

Note that $\text{Lighthouse}_{\text{Bob}}$ and $\text{Lighthouse}_{\text{Alice}}$ are aware of the identities of the endpoints of the communication, but they are not aware of Alice and Bob’s locations. Hence the privacy of Alice and Bob is preserved. In addition, all intermediate routers are unaware of the endpoints of the communication, and hence cannot deduce the locations of Alice and Bob. In fact Alice and Bob can communicate anonymously with respect to all other routers, with the exception of the two Lighthouses. With respect to this communication, the Lighthouses are trusted entities, and hence fully anonymous connections are not provided. In the next section we describe how fully anonymous connections can be achieved. In what we have described so far, we achieve our goal of preserving Alice and Bob’s location privacy from all intermediate routers, including the Lighthouses. The most important thing to note is that Alice cannot deduce Bob’s location, and Bob cannot deduce Alice’s location. Hence communication between Alice and Bob is privacy preserving.

It is worth mentioning that if the user’s Portal colludes with the user’s Lighthouse, then the location and identity of the user may be compromised. However, note that the user’s Portal does not know which Lighthouse the user selected (and vice versa). Hence, for this to be practical, the Portal has to employ a trial-and-error approach to try to get accessible Lighthouses to collude and help in decrypting the initial packet that was received by the Portal from the user. Our system distributes the trust, and assumes that such Lighthouses and Portals span various domains, and collusion between such entities is not feasible.

2.5.2 Anonymous connections

As noted in the previous section, even though Alice and Bob have achieved location privacy, their connection is not anonymous to the Lighthouses. Ideally, we would have a situation where $Lighthouse_{Bob}$ does not know that Bob is exchanging messages with Alice, and where $Lighthouse_{Alice}$ does not know that Alice is exchanging messages with Bob. In such a case Alice and Bob can communicate with full location privacy and connection anonymity, and the only information available to $Lighthouse_{Bob}$ is that Bob is communicating with “somebody,” and likewise $Lighthouse_{Alice}$ knows only that Alice is communicating with “somebody.” We detail an enhancement to the protocol described above to achieve this.

We first modify the message $M_{Lighthouse}$ Bob sends to its Lighthouse to include an encrypted token with Alice’s ID. We will call this token T_{Alice} , which is encrypted with the lookup server’s key (we assume this key to be well known). Hence $Lighthouse_{Bob}$ is not aware of the identity of Alice.

$$M_{Lighthouse} = E_{K_{session}} (COMM_SETUP || T_{Alice} || TS)$$

Next, $Lighthouse_{Bob}$ sends T_{Alice} to the lookup server, which decrypts Alice’s ID and determines $Lighthouse_{Alice}$. The lookup server creates another token $T_{Lighthouse_{Alice}}$, encrypted with $Lighthouse_{Alice}$ ’s key, which contains Alice’s identity. Now $Lighthouse_{Bob}$ uses this token instead of Alice’s ID with $Lighthouse_{Alice}$ to determine $dest_handle_{Alice}$. The rest proceeds as before. Hence Bob can route packets to Alice without $Lighthouse_{Bob}$ knowing that the packets are for Alice. Similarly, $Lighthouse_{Bob}$ will provide src_handle_{Bob} to $Lighthouse_{Alice}$, but without disclosing Bob’s identity. Hence Alice has a reverse communication path in which $Lighthouse_{Alice}$ does not know that Alice is communicating with Bob. Even if there are repeated communication setups between Alice and Bob, the token (e.g., T_{Alice}) will differ each time due to the inclusion of a timestamp. We mention this to emphasize that $Lighthouse_{Bob}$ cannot determine whether Bob is talking to the same person as before, or not. At most, $Lighthouse_{Bob}$ knows that Bob is communicating with various people registered at the same Lighthouse.

Note that if the two Lighthouses involved in the communication collude, then the connection is no longer anonymous. Likewise, $Lighthouse_{Bob}$ can collude with the lookup service to determine Alice’s identity. However, we assume that this is not trivial. Ideally, a large number of Lighthouses spread over different domains will be available making it harder for two particular Lighthouses to collude. Further, recall that certificate authorities will vouch for the trustworthiness of a Lighthouse, and hence a paranoid user could pick a Lighthouse with stronger assurances (higher in the hierarchy). Again, we are distributing the trust in Mist, rather than centering it at one particular place.

3. Use Cases

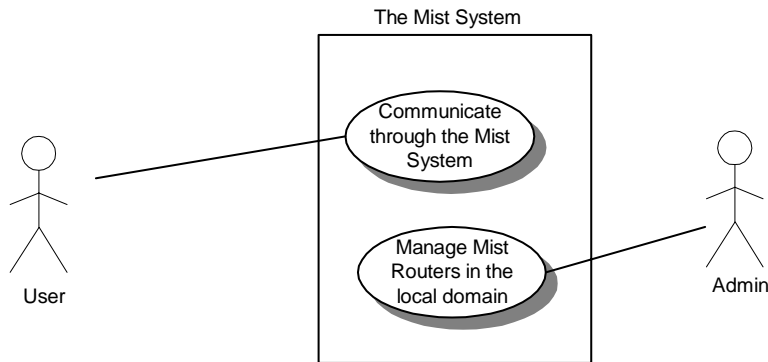


Figure 8: High-Level Use Case Diagram

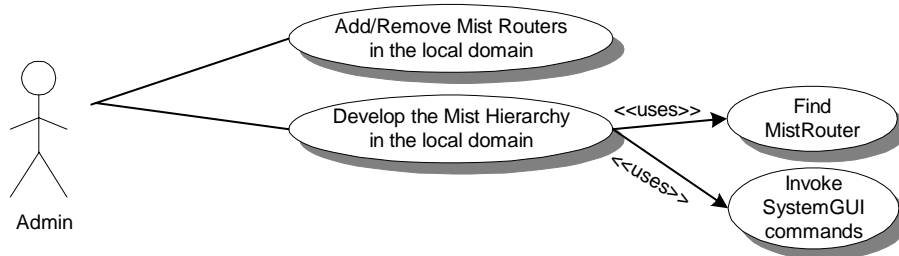


Figure 9: Mist Administrator Use Case Diagram

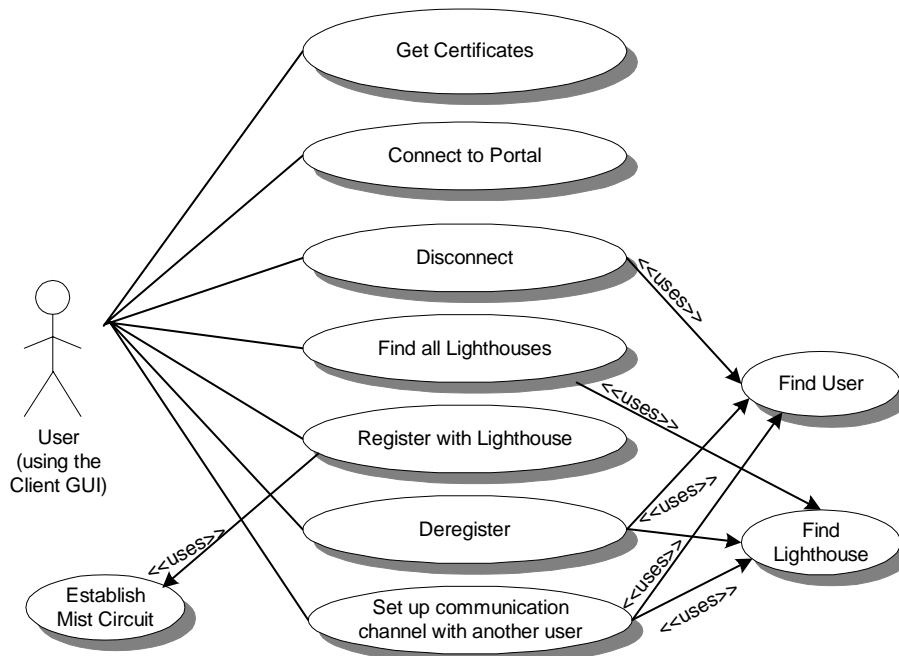


Figure 10: Mist User Use Case Diagram

Figure 8 shows the high-level use case diagram. In essence, the Mist system can be actively used by two actors. The first is the Mist user, which is a user that wanders around in the ubiquitous computing environment, and attempts to use his or her mobile device to privately communicate with other users in the system. The other actor is the system administrator who administers some portion of the Mist system. Note that for better concealment, the Mist should spread across multiple domains

Figure 9 shows the Mist administrator's use cases. An administrator can:

1. Create or launch as many Mist Routers in the local domain as he or she wants. As the number of Mist Routers increase, users' privacy increases.
2. Develop Mist Hierarchies by using the System GUI's drag and drop features to link Mist Routers in the local domain to each other and build a hierarchy. Links can be added, updated and removed as necessary.

Figure 10 illustrates the use cases for regular users of the Mist system. A possible scenario for Mist users is as follows:

Step 1: User Alice wants to use the Mist system. The first step is to connect to the nearest Portal.

Step 2: Alice needs to register with a Lighthouse, so the Portal presents Alice with a list of Lighthouses. Since Alice does not want to reveal her identity to any Mist Routers other than her Lighthouse, Alice builds a Mist Circuit upwards to her selected Lighthouse.

Step 3: Now, another user, Bob, wants to talk to Alice. Bob follows the above steps to connect to a Portal and register with a Lighthouse.

Step 4: Bob "informs" his Lighthouse that he wants to talk to Alice.

Step 5: Bob's Lighthouse locates user Alice, maintains a communication binding entry in its internal tables, and provides Bob with a handle through which he can communicate with Alice. Similarly, Alice's Lighthouse sets up handles to Bob.

Step 6: Using the handle provided, Bob can communicate securely (if he chooses) and privately with Alice.

Step 7: When the communication is over, Bob or Alice can deregister from the Lighthouse and disconnect from the Portal.

Further, users may query the certificate authority (CA) to obtain public keys of the users they are trying to communicate with.

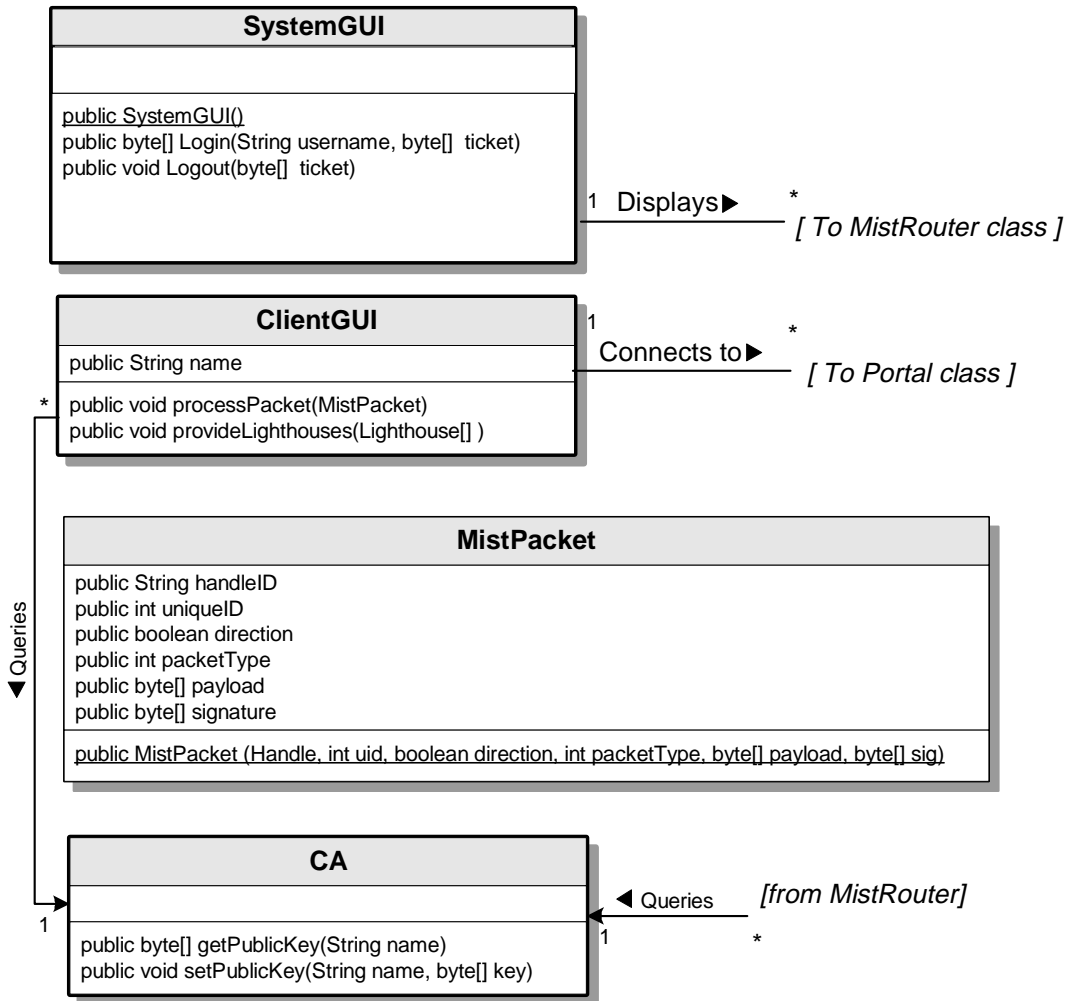


Figure 12: Mist Class Hierarchies - Part II

4.1 Assumptions

Figure 11 and Figure 12 show the Mist’s class diagrams. Since we are implementing the classes in Java, we write the methods in Java syntax. For every method, the data types of the parameters are given; however, the names of the parameters are given only for standard Java types. In addition, we omit the “default” class constructors, which do not take any parameters. The small arrow beside the labels on connectors indicates the direction over which the label applies. For example, in Figure 11 a Lighthouse “has” a MistCommBindingTable.

Note that boxes with bold borders represent CORBA objects in our system, whereas boxes with light borders represent regular Java classes.

4.2 Class Descriptions

4.2.1 MistRouter Class

This class represents a Mist Router. A Mist Router has a unique name, and can be connected to other Mist Routers in a hierarchical fashion. The “deliver” method enables the delivery or routing of packets between endpoints.

4.2.2 Portal and Lighthouse Classes

These are special types of Mist Routers; therefore, they are subclasses of the MistRouter class. A Portal has additional functionality to allow users to connect to and disconnect from it, as well as the ability to discover ancestral Lighthouses. The Lighthouse allows users to register with itself.

4.2.3 Handle Class

The Handle class represents a generic “handle.” Handles are used to conceal information about the endpoints of a communication. They are instrumental in establishing Mist Circuits and enabling hop-to-hop handle-based routing (as described in Section 2.2).

4.2.4 MistRoutingTable Class

This class represents Mist Routing Tables. As described in Section 2, all Mist Routers will have a Mist Routing Table that associates incoming handles with outgoing handles. These tables are used to perform hop-to-hop routing to help conceal the identities of the endpoints.

4.2.5 UserBindingTable

As described in Section 2, a Lighthouse maintains a User Binding Table. This table stores the handle corresponding to users registered at this Lighthouse.

4.2.6 MistCommBindingTable Class

As described in Section 2, a Lighthouse maintains a Communication Binding table. This table keeps track of other users whom the registered users in this Lighthouse are communicating with. The details of this binding table are discussed in Section 2.4. Each entry in this table consists of a “communication binding,” which is a tuple consisting of three elements: the source and destination handles for the target user, and the Lighthouse with which that user is registered.

4.2.7 MistCommBinding Class

This class represents a tuple within the Mist Communication Binding table. The tuple was described in 4.2.6.

4.2.8 LookupService Class

This represents the user lookup service, which is a wrapper for the LDAP service mentioned in Section 2.3.1. The Lookup service will be queried by the Lighthouse during user registration, deregistration, and when locating other users’ Lighthouses. The token is a cryptographic structure generated by the user and can’t be fabricated. This token serves to “prove” that the lookup registration is actually initiated by the user and not fabricated by some Lighthouse.

4.2.9 ClientGUI Class

This represents the Client GUI, which will be running on the user’s client. This allows the user to register/deregister with the Mist and to communicate with other users. See Section 5.2 for details.

4.2.10 SystemGUI Class

This represents the System GUI, which enables the administrator of a particular domain or subdomain to configure and develop the Mist Hierarchy for the local domain. See Section 5.1 for details.

4.2.11 MistPacket Class

This represents a Mist packet. Packets are routed hop-to-hop from one Mist Router to another.

4.2.12 CA Class

Since we assume the existence of a PKI, this class represents a Certificate Authority, which maintains the public keys of all entities in the Mist system. These include users and Mist Routers. The CA can be queried by the Client GUI or Mist Routers.

5. The Mist GUIs

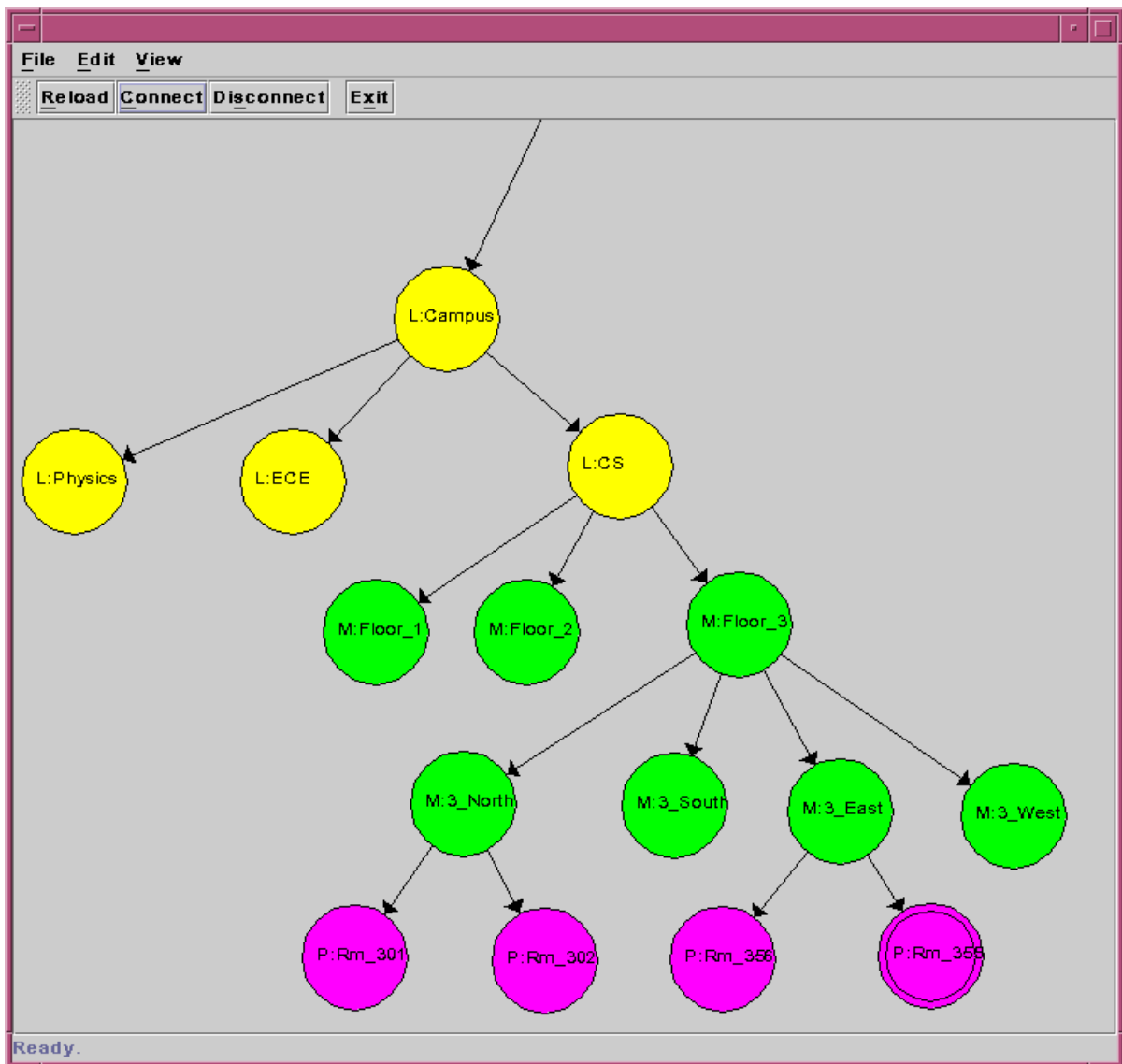


Figure 13: System GUI – Shows the local Mist Hierarchy

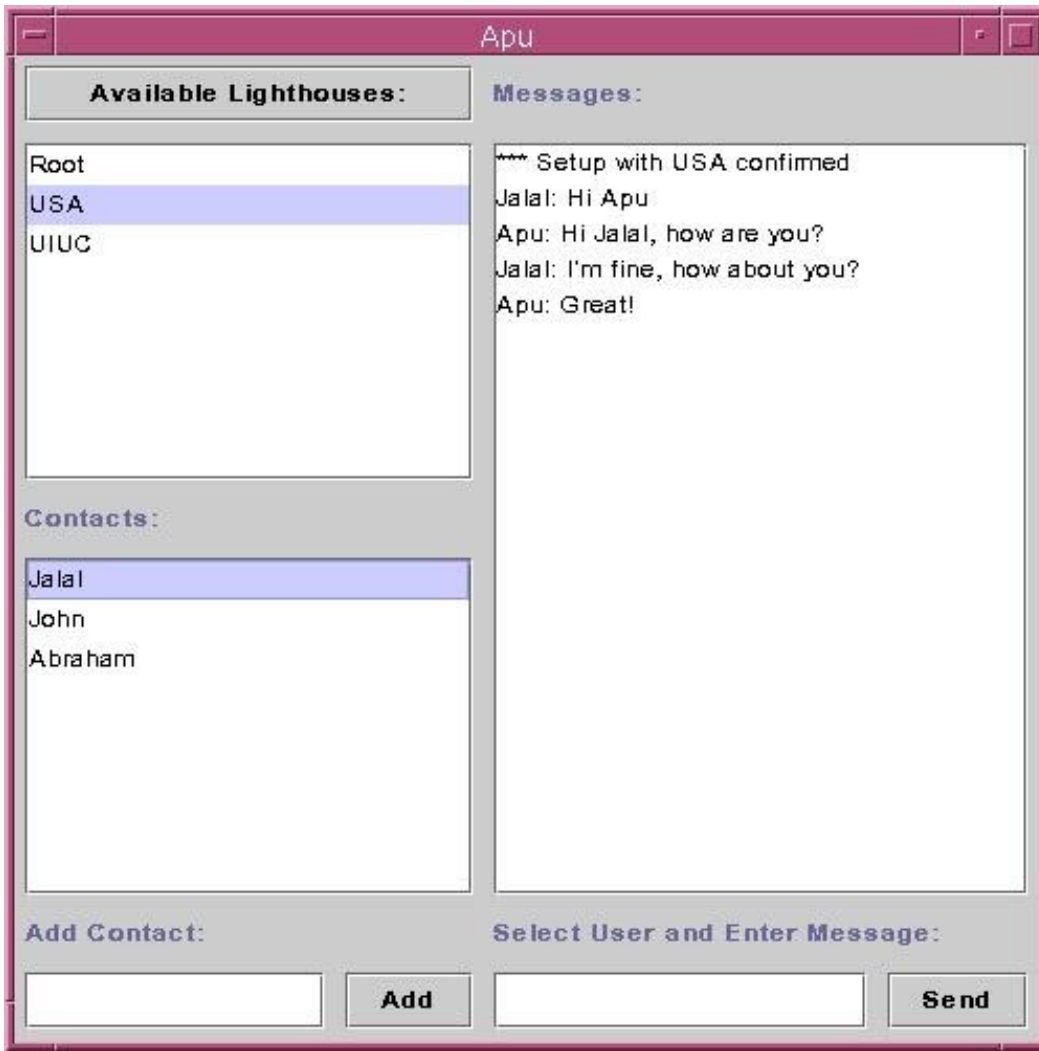


Figure 14: The Client GUI – User can select a Lighthouse, add contacts, and chat with contacts in the Mist

5.1 System GUI

The administrator makes use of the System GUI to administer that portion of the Mist Hierarchy that lies within his or her administrative domain. On startup it contacts the Naming Service and obtains references to the entire domain's Mist Routers (including Lighthouses and Portals) that are registered with the Naming Service. The administrator can now connect the various Mist Routers in the form of a hierarchy. The layout in the GUI is done automatically, although the administrator can use the *manual arrange* mode. Figure 13 shows a demo scenario with 15 Mist Routers in the local domain. The System GUI can be seen in *auto-arrange* mode.

5.2 Client GUI

Our Client GUI Figure 14 is a sample application of how users can interact within the Mist. This GUI models an instant messaging application. The name of the user and the portal name are specified at startup. Before displaying the GUI, a public/private key-pair is generated. The public key is then registered with the Certificate Authority (CA).

Once the GUI is displayed, the user can click the *Available Lighthouses* button to obtain a list of Lighthouses from the Portal. This corresponds to the list of all Lighthouses along the path from the Portal to the Root. When the user selects a Lighthouse, a secure circuit establishment message is sent to the Lighthouse (see Section 2.2). The Lighthouse then sends a secure response encrypted with the session key. This response is recognized by the type `COMM_SETUP_CONFIRM`, and an appropriate confirmation message is displayed in the message area (top right portion of the GUI).

Next the user can add contacts to the contact list as seen in the bottom left region of the GUI Figure 14. The user can now select a contact and type in messages in the bottom right part of the GUI. These messages are tagged with the type `COMMUNICATION_MSG`, encrypted with the session key and sent to the Lighthouse. The Lighthouse then examines the packet and forwards it to the contact's Lighthouse. When a message is received from another user, it recognizes the `COMMUNICATION_MSG` message type, and displays it in the message area.

In future work, we would like to make the confirmation messages from Lighthouses to the Client GUI have a stronger effect on the GUI. For now, it is treated as just another message that the user can read in the message area. However, a negative confirmation should disallow the user from sending messages (since the user is not registered with a Lighthouse).

We would also like to maintain separate message areas for each contact. Hence clicking on *Jalal* should only display messages from Jalal, and the same for *Abraham* and *John*.

6. Optimizing Mist Communication

We believe our protocol can be enhanced further by making it more transparent such that it requires less user intervention and optimizing communication. For instance, turning our attention to Figure 1, we can see that all communication from Bob to Alice will first travel up the hierarchy to $\text{Lighthouse}_{\text{Bob}}$, then to $\text{Lighthouse}_{\text{Alice}}$, and finally to Alice. Communication from Alice to Bob is similar. We can see that Alice and Bob pay the penalty of privacy in terms of extra "hops." Ideally we would like to "short circuit" their communication to take the shortest path possible, while still maintaining location privacy and communication anonymity. With respect to the hierarchy, the shortest path will go through a "lowest common ancestor" Mist Router with respect to Alice and Bob. We call this $\text{Lighthouse}_{\text{LCA}}$. We would like to redirect all communication between Alice and Bob to go through $\text{Lighthouse}_{\text{LCA}}$, while maintaining privacy and an anonymous connection. Finding suitable schemes for such an optimization of Mist communication is a subject of future research. However, we outline one approach that we plan on exploring.

After Alice and Bob have setup their communication, they use end-to-end secure communication to negotiate a number n . They then generate vectors V_{Bob} and V_{Alice} , each containing a list of N (where N is suitably large and $\geq n$) random Mist Portals, where the n th Mist Portals in the two vectors correspond to the actual portals of Bob and Alice respectively. Alice and Bob send these vectors to $\text{Lighthouse}_{\text{Bob}}$ (we could have also chosen $\text{Lighthouse}_{\text{Alice}}$). We observe that $\langle V_{\text{Bob}}(n), V_{\text{Alice}}(n) \rangle$ corresponds to the actual end-to-end locations of Alice and Bob, however $\text{Lighthouse}_{\text{Bob}}$ does not know the value of n , and hence cannot guess the actual endpoints of Alice or Bob with certainty. Obviously, as N increases, the uncertainty of Alice and Bob's actual locations increases. In fact, if N is taken to be the maximum of the total number of Portals under $\text{Lighthouse}_{\text{Alice}}$ and $\text{Lighthouse}_{\text{Bob}}$ then $\text{Lighthouse}_{\text{Bob}}$ does not have any more information about Alice and Bob's locations than it already had. In the next step, $\text{Lighthouse}_{\text{Bob}}$ cycles through the N tuples of the form $\langle V_{\text{Bob}}(i), V_{\text{Alice}}(i) \rangle$, where $i=0, 1, \dots, (N-1)$. For each such tuple, $\text{Lighthouse}_{\text{Bob}}$ calculates a candidate $\text{Lighthouse}_{\text{LCA}}$. We assume that $\text{Lighthouse}_{\text{Bob}}$ is aware of the topology of the hierarchy, and is able to deduce $\text{Lighthouse}_{\text{LCA}}$. For each pair, $\text{Lighthouse}_{\text{Bob}}$ generates $\text{Lighthouse}_{\text{LCA}}(i)$, and constructs a return vector V_{LCA} , where $V_{\text{LCA}}(i) = \text{Lighthouse}_{\text{LCA}}(i)$. $\text{Lighthouse}_{\text{Bob}}$ sends V_{LCA} to Alice and Bob, which now can set

up communication through $\text{Lighthouse}_{LCA}(n)$. All the other entries in V_{LCA} are discarded, since they were used to obfuscate Alice and Bob's locations from Lighthouse_{Bob} .

In this scheme we use Lighthouse_{Bob} to optimize communication without disclosing location information with high certainty (this depends to the value of N). We also rely on Lighthouse_{Bob} to not send V_{Bob} to Alice and V_{Alice} to Bob, because this would disclose their locations to each other! Anonymous communication is not affected by this optimization. However, there are caveats to this approach, since Alice and Bob can now deduce that the other user is somewhere in the Mist Hierarchy "under" Lighthouse_{LCA} , but we assume that both Alice and Bob do this consensually, and hence this is not a problem. We are exploring approaches to get rid of this restriction. For example, Alice and Bob could specify ranges in the Mist Hierarchy over which a Lighthouse_{LCA} can be selected. If Lighthouse_{Bob} cannot discover one that satisfies both ranges, then the optimization is aborted.

7. Conclusions from our Implementation

Ubiquitous computing is an emerging research area with great potential. However, without careful consideration for user privacy from the ground up, there is a fair possibility of creating a ubiquitous 'surveillance' system instead. To avoid this undesirable future, we contend that the privacy and anonymity of users in ubiquitous computing environments should be considered seriously and carefully from the very beginning of the system design phase.

We present and implement a scheme to preserve privacy in ubiquitous computing environments. We have met the privacy objectives that we set forth, namely, location privacy, anonymous connections, and confidentiality. This was achieved through the use of Mist Routers, hop-to-hop handle-based routing as well as a hybrid of symmetric and asymmetric encryption to provide confidentiality. This work shows that, with a careful design, privacy can be effectively preserved in a ubiquitous computing environment.

8. References

- [1] The Gaia Homepage, <http://choices.cs.uiuc.edu/ActiveSpaces/index.html>
- [2] M. Langheinrich, "Privacy by Design – Principles of Privacy-Aware Ubiquitous Systems," *ACM UbiComp 2001*, Atlanta, GA, 2001.
- [3] M. Reiter and A. D. Rubin, Crowds: Anonymity for Web Transactions, *ACM TISSEC* June 1998
- [4] M. Roman and R. Campbell, "GAIA: Enabling Active Spaces," *9th ACM SIGOPS European Workshop*, September 17th-20th, 2000, Kolding, Denmark.
- [5] T. Ballardie, P. Francis and J. Crowcroft, "Core Based Trees (CBT), An Architecture for Scalable Inter-Domain Multicast Routing," *ACM SIGCOMM*, 1993.
- [6] Y.K. Dalal and R.M. Metcalfe, "Reverse Path Forwarding of Broadcast Packets," *Communications of the ACM*, 21:1040-1048, December 1978.