Person-level Routing in the Mobile People Architecture

Mema Roussopoulos Petros Maniatis Edward Swierk
Kevin Lai Guido Appenzeller Mary Baker
Department of Computer Science
Stanford University
Stanford, California, 94305
{mema, maniatis, eswierk, laik, appenz, mgbaker}@cs.stanford.edu, http://mosquitonet.stanford.edu/

Abstract

Ubiquitous network connectivity for devices does not automatically imply continuous reachability for people. People move from place to place and switch from one network device to another. As a result, phones ring in empty offices, email cannot reach most cell phones, and spam clogs expensive, low-bandwidth links to laptops. Whereas existing mechanisms have addressed host mobility or the mobility of people within one network, few have allowed *people*, the ultimate and most important endpoints of communication, to roam freely, without being constrained to one location, one application, one device, or one network.

We have designed the Mobile People Architecture (MPA) to maintain *person-to-person reachability.* The central component of MPA is a *person-level router* called the *Personal Proxy.* It tracks a mobile person's location, accepts communications on his behalf, converts them into different application formats according to his preferences, and forwards the resulting communications to him. In contrast to similar systems, the Personal Proxy protects the user's privacy, is easily extensible to new network devices and applications, and has been deployed with no modifications to the existing network and telecommunications infrastructure. In this paper, we describe the design, implementation, and preliminary evaluation of our prototype Personal Proxy, a service that integrates Internet and telephone communication and addresses the need for person-to-person reachability.

1 Introduction

One of the defining trends of this decade has been the explosive growth of the Internet and other communication networks. People have access to a growing number of networks (e.g., Internet, cellular, pager) on a growing number of devices (e.g., personal digital assistants, cell phones, smart cards) at a growing number of locations (e.g., work, home, on the road). Unfortunately a growing problem for these people is maintaining reachability: as network devices, applications, and accessible locations proliferate, it becomes less likely that other people (*correspondents*) can get in touch with a mobile person at any particular time. For example, a correspondent might not have the mobile person's hotel phone number, or the correspondent's email application might not interoperate with the mobile person's phone. We therefore believe that there is a need for a *person-level router* that meets the following goals:

Maintain person-to-person reachability. The router should direct the correspondent's communications to the mobile person, regardless of whether the two participants have direct access to the same kind of network, device, or application.

Protect the mobile person's privacy. To route communications to a mobile person, a person-level router must track the devices and applications through which the person is currently reachable. The router should not reveal this tracking information, whether current or historical, because it could be used to deduce the person's location and compromise his privacy. In addition, receiving unwanted messages is also an invasion of privacy. Many applications, such as those in many phone systems, have no way to deliver high priority communication intrusively while delivering low priority communication unintrusively. Users should be able to have all their incoming communications prioritized and filtered according to their preferences.

Extend easily to new network devices and applications. Given the rate at which communication networks, devices and applications proliferate, it is essential that a person-level router be easily extensible.

Be deployable without modifying existing infrastructure. Considering the fast pace at which new networking technologies develop, a communication system that requires changes to the existing network and telecommunications infrastructure is difficult to deploy and runs the risk of becoming obsolete before it is widely available. A person-level router must be easily and rapidly deployable to benefit the greatest number of people.

In this paper, we describe the design and implementation of the *Personal Proxy*, a person-level router which we believe meets all of these goals. In Section 2, we give an overview of the Mobile People Architecture (MPA), which includes the Personal Proxy. In Section 3, we describe the Personal Proxy in detail. In Section 4, we evaluate our prototype using our design goals. In Section 5, we describe related work. Finally, in Section 6 we conclude.

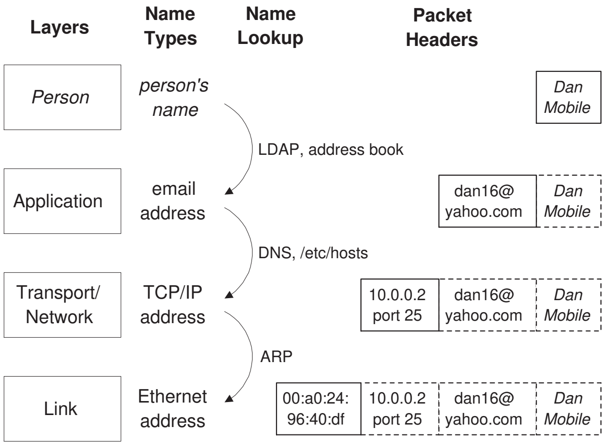

Figure 1: The Layering Model. We show the traditional networking layers, extended with the *Person* layer. *Name Types* shows examples of the kinds of names used at each layer. *Name Lookup* shows some methods of mapping names from each layer to names in the next lower layer. *Packet Headers* shows examples of actual names at each layer, and their relative locations in a typical email packet.

2 Architecture Overview

In this section, we describe how person-level routing fits into the overall picture of networking and argue that the Mobile People Architecture [MRS+99] is a logical extension of the current model of networking.

Networking systems are traditionally organized using a layering model composed of Application, Transport/Network, and Link layers (Figure 1). This model is useful in clearly defining the responsibilities and restrictions of software that exists at each level. To be implemented fully, a layer needs a naming scheme, a way to resolve those names, and a way to route communications.

The *Name Types* column of Figure 1 shows the naming scheme that Internet email uses at each layer. Some examples of names are shown in the *Packet Headers* column. These naming schemes usually mandate that the names are unique and change infrequently. In addition, each layer in the figure has a protocol to map its names to lower-layer names (the *Name Lookup* column in Figure 1). This mapping facilitates routing a communication to its destination.

To model the full process of person-to-person communication, we need to extend the model to include people (the new *Person* layer, as shown at the top of Figure 1), since most important communication is ultimately from one person to another, rather than merely from one device to another. Currently, the Person layer is implemented in an ad hoc, non-unified way. People are not always named in a unique way, although a name or nickname is often unique among those with whom a person communicates frequently. These names (e.g., Dan Mobile) are resolved into application-specific addresses (ASAs, e.g., dan16@yahoo.com) using a directory service (e.g., LDAP [WKH97]), an address book, or simply from a person's memory.

Consequently, applications have difficulty delivering communication to people who move from one application-specific address to another. For example, if Dan Mobile stops using his cell phone because he entered his office building—where land-line phones are cheaper and have better quality—and starts using his office telephone, he might not receive Jane Sender's phone call in a timely manner. Furthermore, if Dan is temporarily unavailable by phone but is reachable by email, Jane, who may not be near an email terminal, cannot communicate with him until he is available by phone. Unfortunately, even in the case where Dan's cellular carrier allows him to forward his missed phone calls to his office email address, Dan will still be unreachable when he moves from his office to a conference room. The problem is that Jane cannot identify Dan in a way that is independent of how he is reachable.

The solution is to create a unified implementation for the Person layer. Such an implementation needs to name people, map people's names to application-specific addresses, and route communications between people (which we refer to as *person-level routing*).

There are naming schemes that assign unique identifiers to people; we call such names Personal Online IDs (POIDs). A POID maps to those ASAs through which a person is reachable. The use of POIDs is discussed in [MRS+99]. However, the design and implementation of our person-level router does not depend on a particular naming scheme.

The role of a person-level router is similar to that of an IP router: it takes communication from a variety of interfaces and directs it out one or more interfaces, based on the recipient's preferences and on characteristics of the communication itself. The closest current approximation is a human assistant who answers Dan's phone, reads his email and forwards his messages by calling, emailing, or paging him. Aside from wasting the assistant's time, the human approach would have difficulty handling real-time communication (e.g., forwarding an IP telephony call to Dan's cell phone).

The *Personal Proxy* (see Figure 2) is our implementation of a person-level router. It performs three distinct tasks: tracking, converting, and forwarding. As a *tracker*, Dan's Personal Proxy maintains his current accessibility information (see Section 3.2). As a *converter*, the Proxy converts communication into a form that can reach Dan, regardless of how he is currently reachable (see Section 3.4). As a *forwarder*, Dan's Personal Proxy ensures that Dan is reachable only in the ways he wishes (see Section 3.3).

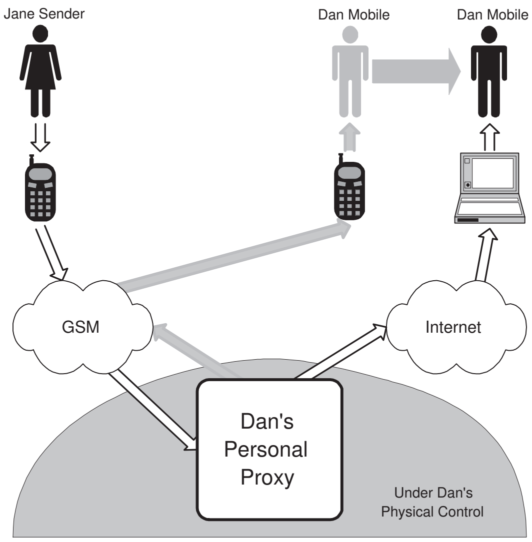

Figure 2: The Personal Proxy acts as Dan's online personal assistant. It forwards Jane's phone call to Dan's cell phone, while he is using it. When Dan stops using his cell phone and starts reading email on his laptop, the Proxy converts Jane's next call to an email message and forwards it on to the email address through which Dan is now reachable.

3 Design and Implementation

The Personal Proxy is the online analog of a mobile person's human assistant. It has its own set of application-specific addresses (ASAs), which the mobile person distributes as his own. Correspondents use these ASAs when they wish to contact him. Because the ASAs that the mobile person is really using are never revealed, his location privacy is preserved.

The Personal Proxy requires three types of information to determine how to route communication to a mobile person: the current applications through which he is available, his preferences in the form of rules, and the types of conversions the Proxy can perform. This information is used by the *Tracking Agent*, the *Rules Engine*, and the *Dispatcher*, respectively, and configured through a *User Interface*. In this section we present our extensibility model and describe each of these components.

3.1 Application Drivers

Application Drivers form the basis of our extensibility model. They are interchangeable components of the system, supporting specific programming interfaces for tasks such as protocol parsing, data transport, and filtering. Application-specific implementations of these programming interfaces can be

Dispatcher		
	Session	Input Session drivers receive incoming communication. Output Session drivers generate and send out communication in application-specific formats.
	Protocol	Parses metadata and content from application-specific communication formats.
	Conversion	Converts data from one content type to another.
Rules Engine		
	Condition	Checks a communication for particular properties.
	Action	Modifies a communication in a particular way.

Figure 3: The basic programming interfaces for Application Drivers, defined for each component of the Personal Proxy.

plugged into the system at runtime and are immediately operational. Application Drivers follow the traditional object-oriented model; abstract components are created for each programming interface, which are then subtyped for specialization. Figure 3 briefly describes the driver interfaces, their functions, and the components in which they reside. The following sections will present how Application Drivers are used in more detail.

3.2 Tracking Agent

The Tracking Agent monitors the mobile person's *connectivity state*. This is a list of applications through which the mobile person can currently receive communications. For each receiving application, the Tracking Agent keeps an ASA, a protocol type and a list of communication formats, such as HTML text or JPEG image, that can be handled by the application at that ASA.

To track a user's location accurately, the Tracking Agent supports three registration methods: *scheduled*, *manual*, and *automatic*. The scheduled registration method simply assumes a change in the user's location according to a preset schedule. Manual registration requires that the user manually send a registration message to the Personal Proxy by, for example, filling out a secure web form, calling the Personal Proxy's phone number, or sending it registration email. Automatic registration is more convenient for the user, but is feasible only if the device is able to determine the user's availability and send a registration message automatically. Whatever the registration method, all registrations must be authenticated and encrypted to preserve the user's location privacy and to prevent false registrations.

3.2.1 Registration API

The Tracking Agent requires all registration clients that register on behalf of the user to conform to a specific registration Application Programming Interface. The main types of functionality included in this API are as follows:

Introspection A registration client must be able to inform the user what kinds of communication protocols (e.g., email, telephony, fax), content types (e.g., GIF89 image, RIFF audio), devices (e.g., Nokia 6190, PalmPilot V) the Personal Proxy knows about.

Customization A registration client must be able to make it easy for the user to create an initial, custom environment that can be reused in the future with minimal tweaking. More specifically, frequently-used *endpoints*, i.e., combinations of devices, protocols and content types, should be easy to alias and reuse. Similarly, frequently-used ASAs (e.g., the mobile person's work phone or instant messaging ID) or POIDs (e.g., mom's, or an employer's) should be easy to alias.

Activation Given the two functions above, a registration client must be able to mark a nicknamed endpoint as active or inactive. It should also be possible to create and activate a single-use endpoint, without having to define an alias for it first.

Our prototype currently implements two forms of manual registration: the user either indicates his connectivity state by accessing a secure web page

Rule #1:

IF From = "mom@home.net" **AND**
(content *contains* "emergency" **OR** content.audio *audio.pitch-is* "high")
THEN content.text *truncate-to* "5 KB" **AND** *send-to* "cell phone 1"
THEN *send-to* "work email"

Rule #2:

IF content *contains* "make" **AND**
content *contains* "money" **AND**
content *contains* "fast"
THEN *drop*

Figure 4: A symbolic example of some rules accepted by the Rules Engine. Italics designate Condition or Action Drivers. *Contains* is a polymorphic condition (it can refer to text, audio, image, etc.). *Pitch-is* is a type-specific condition, as it refers only to audio content; it determines whether the pitch of the voice is relatively high. *Truncate-to* causes a content to be truncated. *Send-to* forwards a communication to the named aliased endpoint. *Drop* causes a communication to be discarded.

and downloading an applet that makes authenticated Java RMI calls to the Tracking Agent through the registration API, or sends signed email with his connectivity state. We plan to add other registration methods, including an automatic method where the user's location is tracked via a "smart badge."

3.3 Rules Engine

The Rules Engine uses the current connectivity state and the user's preferences to direct the routing decisions made in the Personal Proxy. The user enters routing preferences through the User Interface as an ordered list of *rules*. We first describe the structure of a rule. We then describe how the Rules Engine creates a set of *directives* that the Dispatcher will use when routing communication.

3.3.1 Rule Structure

Rules in the Personal Proxy follow the form

IF condition **THEN** action
THEN action ...

(see Figure 4 for an example).

Conditions are *generic*, *type-specific* or *polymorphic* and can be simple or composite (i.e., simple conditions combined using the logical connectives AND, OR, NOT). A generic condition is defined on the metadata of a communication. It is formed as a logical predicate on properties shared by all communication protocols and content types, such as size, sender or send date.

A type-specific condition is defined on one particular content type or communication protocol. For instance, such a condition could be placed on particular headers of an RFC822 email message (such as the X-Face header, which carries a low-resolution picture of the sender's likeness), or properties of an image (such as its color depth).

A polymorphic condition can be defined on many content types but has a different implementation for each one of them. Containment or similarity conditions fall within this category. Some examples are "Does the communication contain my company's logo?" or "Does the communication contain the words 'make,' 'money,' and 'fast'?" Containment in the former case invokes an image pattern-matcher whereas in the latter case it invokes a text matcher.

Actions serve two functions: *routing* and *content conversion*. Routing actions change the way a communication is delivered (or not) to the mobile person. For example, a simple one would be forwarding to a different person and a complex one would be prioritized forwarding requiring explicit acknowledgements. Content actions modify the data or convert the format of a communication. They try to massage contents to fit a particular format or set of requirements specified by the user or obtained from the properties of the receiving application. Examples include truncating a large email sent to a pager or reducing the resolution of an image sent to a PDA.

There are two ways actions can be combined within a rule. First, actions can be composed, as is the case with the first THEN clause in Rule #1 of Figure 4. Here, the truncation occurs first and its result is sent to the application represented by the nickname "cell phone 1". Second, actions can be juxtaposed independently. In the same rule, the two THEN clauses are applied independently, in parallel or serially; the side effects of the former do not affect the latter and vice versa.

Type-specific and polymorphic conditions, as well as content actions can be installed and removed at runtime (see Section 3.1).

3.3.2 Creating Directives

When a communication arrives at the Personal Proxy, the Rules Engine determines where the communication will be routed. It evaluates the generic conditions of each rule one at a time. Conditions dependent on the type of the content (i.e., type-specific or polymorphic conditions) are evaluated as part of the conversion planning process (see Section 3.4.1). When the Rules Engine encounters a rule whose generic conditions evaluate to *true*, it assembles a set of directives derived from the remaining conditions as well as the actions of the rule. A directive consists of a *destination* and a *goal state*. A destination is a set of ASAs where the communication should be routed. This set is determined by the routing actions of the rule. The goal state contains certain requirements the Dispatcher must fulfill when handling a communication:

1. the required output content type,
2. the as yet unresolved conditions of the rule that must evaluate to *true* before the communication is routed, and
3. the content actions that are to be applied to the communication before it is routed.

Independent actions of a rule result in multiple directives. For example, Rule #1 in Figure 4 would result in two directives, one corresponding to each THEN clause. Conflicting goal states are resolved at the Dispatcher.

3.4 Dispatcher

The Dispatcher routes incoming communications to one or more ASAs, possibly converting from one application type to another along the way.

The typical path of a communication through the Dispatcher is the following (see Figure 5):

- An Input Session Driver (see Figure 3) receives the incoming communication.
- A Protocol Driver parses the communication into its metadata and its content.
- The Dispatcher queries the Rules Engine for the directives that pertain to this communication.
- The Conversion Planner (see Section 3.4.1) constructs a path through Conversion, Condition and Action drivers to bring the communication into the desired format.
- An Output Session Driver sends out the resulting communication to its destination.

3.4.1 Conversion Planning

The heart of the Dispatcher is the Conversion Planner, which transforms incoming communication from the sender into a form understandable by the mobile person's current applications or devices. The Conversion Planner has two tasks. First, it must plan and invoke a sequence of conversion drivers that implement specific transformations on the incoming communication. Examples of transformations are converting from one data type to another (for instance, text to an audio format through speech synthesis) or reducing the size of the communication (for instance, cutting all but the first 100 bytes of text, reducing the sampling frequency of a sound, or increasing the compression levels of an image). Some of the transformations might be inherently required by the mobile person's device or application (for example, the available display might have a limited color depth), some are used in a preventive manner (for example, to avoid overloading

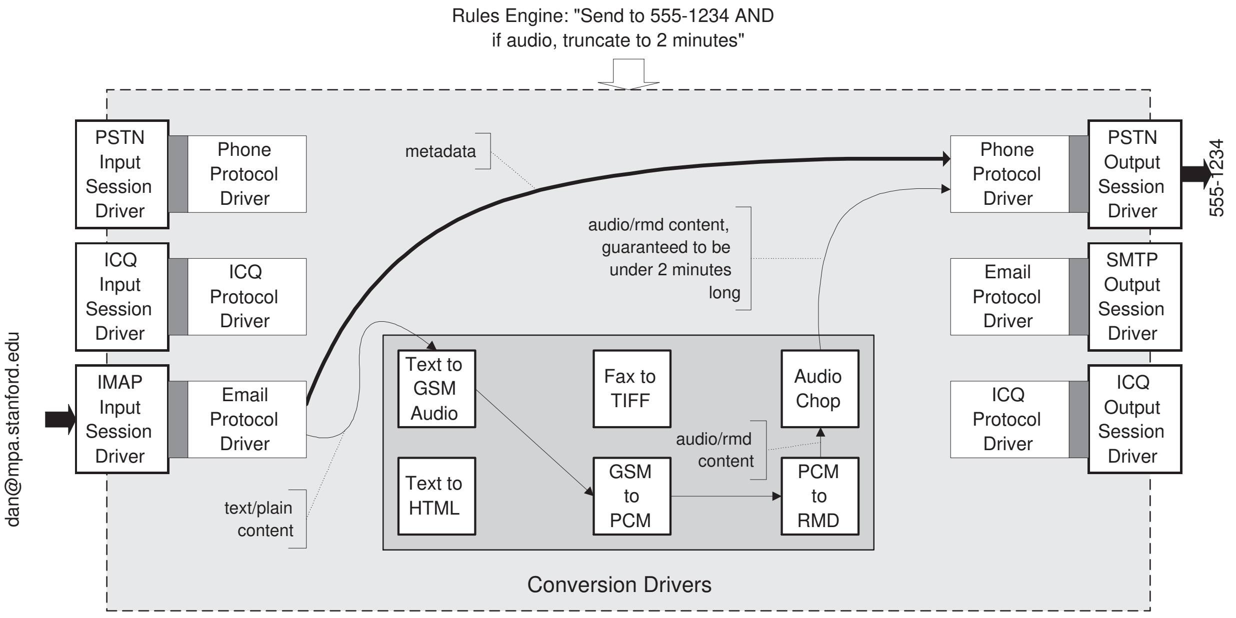

Figure 5: The Dispatcher. This figure shows how the Dispatcher routes email arriving at the Personal Proxy's email address to the telephone number at which Dan can be presently reached. Thin light arrows indicate content flow. Thin bold arrows indicate metadata flow.

the network connection or device memory), while others just aim to summarize unimportant communication selectively (for example, remove decorations from an incoming fax transmission, only leaving the text behind).

Second, the Conversion Planner must deal with type-specific and polymorphic conditions as well as content actions imposed on a communication by the Rules Engine. Such conditions or actions cannot always be applied by the Rules Engine because they might require a conversion first. For example, consider a scenario where the condition is "contains the word 'emergency'" and the incoming communication is fax. To evaluate this polymorphic condition when it has only been defined on text (i.e., when there is only a textual pattern-matcher available), the Rules Engine must invoke an optical character recognition utility to perform an image-to-text conversion, and then use the textual pattern-matcher to check for the word "emergency." To perform the appropriate invocations, the Rules Engine would in general need to implement its own planner. Rather than duplicating the planning work, the Rules Engine simply passes along to the Dispatcher those type-specific conditions, polymorphic conditions and content actions, that require some conversion before they can be used.

For each directive obtained from the Rules Engine, the Conversion Planner constructs a separate plan in three phases. In the first phase, the Conversion Planner constructs a sequence of conversion drivers. This is done through a simple breadth-first search of the conversion driver space. The end result is a sequence whose starting point is the incoming content type and whose endpoint is the required output content type listed in the goal state of the directive.

Conversion drivers are each given equal weights currently, although it would make sense to favor "cheaper" conversion drivers over more "expensive" ones, for whichever cost metric the user finds important. For example, postscript-to-text conversions require far less computation than text-to-speech conversions and if cycles are at a premium, we might want to favor the former over the latter.

In the second phase, the Conversion Planner determines when type-specific and polymorphic conditions (listed in the goal state of the directive) are to be evaluated in the sequence. If a condition cannot be evaluated anywhere in the sequence of conversion drivers, then the Conversion Planner inserts additional conversion steps to the sequence of drivers to ensure that the condition is evaluated. This might create a new branch in the sequence of drivers, since the conversion path to check a condition might not be the same as the path needed to dispatch the content to its destination. In the example described above, where we need to use a textual condition on a fax, if the destination is also fax there is no need to go back from text to image; as soon as the textual condition is evaluated (and provided it evaluates to *true*), the fax can just be dispatched as is, discarding the image-to-text conversion step. Otherwise, the whole process is aborted.

In the third phase, the Conversion Planner determines when the content actions (listed in the goal state of the directive) are to be performed. This involves adding action drivers to the sequence. To be applicable, content actions might also require extra conversions. Unlike the case with conditions, however, extra conversions inserted to accommodate content actions do not create a new branch in the converter sequence, since the results of the content actions must be included in the outgoing content. Again, in the fax example above, if it is required to use a textual spell checker before the fax is forwarded, then an image-to-text converter, the spell checker, and a text-to-image converter have to be inserted in the conversion sequence.

We plan to explore the utility of associating an importance factor with each condition or action affecting the conversion path. This would allow low-priority conditions or actions to be ignored if they cannot be used without adding extra conversion steps.

3.5 User Interface

Unlike a network-layer router, which is usually maintained behind the scenes by a system administrator, the Personal Proxy must be configured with personal information specific to its user most likely *by* its user, to route communications effectively. Since no single user interface can work on every device from which the user might want to configure the Personal Proxy, multiple user interfaces are supported via the Tracking Agent and Rules Engine APIs. In this section we describe the design principles underlying our development of the Personal Proxy user interfaces.

3.5.1 Functionality

Two key sets of information are required of the user: application and device registrations, which are sent to the Tracking Agent, and routing rules, which are sent to the Rules Engine.

Additional information is requested from the user to make the interface easier to use. When the user first configures the Personal Proxy, he is asked to select from a list of all the devices or applications that he might use, and assign each of them an easy-to-remember nickname. When he later registers an application as active, he can simply choose one of the previously configured nicknames. The user can also set up an address book containing the ASAs or POIDs of people to whom he might want to forward communications via routing rules.

3.5.2 Supporting Different Interfaces

In keeping with the extensible nature of the system, the Personal Proxy supports any kind of user interface as long as it conforms to the APIs for the Tracking Agent and the Rules Engine.

The user interfaces we are currently prototyping are an interactive, Web-based interface which provides full configuration functionality, and a non-interactive, email-based interface which only allows registering previously configured applications.

3.5.3 Design Issues

User interfaces for communication filtering have been notoriously hard to design for several reasons. First, the structural information enclosed in filtering rules is complicated, sometimes consisting of multiple levels of a logical tree. Second, it is difficult to give the user an intuitive idea of what a certain set of rules will do when applied to a particular communication, since many rules might participate in the decision and some of their mandated actions might be mutually contradictory. Third, in an extensible filtering system such as the Personal Proxy, the interface must be flexible enough to accommodate rule components (conditions or actions) that might not have been defined at the time of interface design. The usual result is that user interfaces are either too complex for the uninitiated to use, or too simplistic for experienced users to perform more ambitious tasks.

In the Web-based interface we are currently prototyping, we attempt to address each of these problems. To deal with the complex structure of filtering rules, the interface presents a two-pane display. For context, one pane shows the hierarchy of objects to which rules can be applied; the other contains the set of rules for the object currently in focus. To help the user anticipate the effect of a set of rules, the interface allows the user to send "preview" communications through the Personal Proxy and view the results without actually completing delivery. For real communications, the Dispatcher records the sequence of applied rules and conversions so that the user can later determine what went wrong if they are delivered improperly. Finally, we attack the extensibility problem by allowing developers of new rule components to override methods which define the visual presentation of each component.

4 System Evaluation

We evaluate our system using three criteria: preservation of privacy, extensibility, and ease of deployment.

4.1 Preservation of Privacy

A primary goal of our system is to preserve the privacy of the mobile person. While complete preservation of privacy is impossible [Lam73], our system goes much further towards this goal than current systems do (see Section 5). This is mainly accomplished in two ways: first, by hiding all locations visited by the mobile person and, second, by filtering incoming communications and routing them according to their desirability.

The only ASAs for Dan that Jane Sender knows of are those of his Personal Proxy. The Proxy can receive and forward all of his communications to his undisclosed, current ASAs. In addition, his registrations are encrypted and authenticated (see Section 3.2). The Proxy prevents Jane from determining Dan's exact network location, device or application, thus protecting his location privacy. Dan's location privacy is as secure as his Personal Proxy.

Table 1: Lines of code in selected Application Drivers.

Driver	Code	Doc	Total
Email Protocol	139	135	274
IMAP Input Session	108	119	277
SMTP Output Session	169	52	221
Phone Output Session	100	56	156
Text-to-Audio Conversion	39	6	45

The Proxy can run on a single host under the complete physical control of its user (although several users could share a Personal Proxy for lower cost). Consequently, our system is more secure than others that depend on several nodes, which are not all under the control of the user (or trusted third party), or of whose existence the user is not aware.

One unresolved issue concerns an adversary residing on the path between Dan and his Personal Proxy. In this case, encryption cannot preserve Dan's location privacy if registration flows are identifiable. So far we are considering disguising the flows or transmitting decoy registrations, but the adversary issue is known to be a difficult problem.

The Personal Proxy prevents unwanted communication by allowing the user to define content-based rules governing how unwanted or low-priority communication should be handled (see Section 3.3). Since rules can handle any protocols or content types, they can filter the user's communications more comprehensively than traditional filtering systems.

4.2 Extensibility

As explained in Section 3.1, extending support to a new messaging or transport protocol, or even a new content type, is merely a matter of writing appropriate new Application Drivers. The new drivers can be introduced to the running system, without requiring a restart.

We measured the extensibility of our system as the amount of programmer time and lines of code required to add functionality. Given a preexisting unofficial Java API for ICQ transport and messaging, an experienced programmer was able to create a working set of drivers for input/output sessions and messaging in less than an hour. (ICQ [ICQ] is an instant messaging program.)

The code required to glue various other functionality into the architecture is given in Table 1. The number of lines of code and the totals for code and comments measure approximately the difficulty of adding new functionality to the architecture.

Although this gives only a rough measure of the extensibility of the system, it is safe to say that extending the system does not require bringing the system down and can be done easily, using off-the-shelf software and hardware components. It is straightforward enough that an advanced user can add functionality without days of development.

4.3 Ease of Deployment

The most important metric of a communication system is the number of people who can use it. The easier a communications system is to deploy, the more people can benefit from using it. In addition, given the rate of development of new networking technologies, a system that is difficult to deploy may become obsolete before it is widely available. The benefit of such a system is limited. For example, ISDN was not widely available in the United States until fourteen years after it was first specified [Sta94] because it required upgrading phone switches and wiring. By that time, faster technologies like cable modems and DSL were already emerging. In contrast, the World Wide Web was deployed widely in less than four years, in part because an individual can set up a web server without modifying the network infrastructure.

We have designed the Personal Proxy to be as easy to deploy as a web service. It is a single Java server that can be installed by an individual on any Java-compatible host with no special support from the underlying network infrastructure. The only requirement is that the Proxy be able to communicate with each device through which the mobile person expects to be reachable. In general, if a mobile person expects to communicate through a particular network (e.g., the telephone network or the Internet), he probably already has connections to those networks in his home or at work.

5 Related Work

Many existing solutions allow user mobility among devices in a network of a single type. Examples include the GSM and UMTS [UMT] cellular telephony systems and the Personal Mobile Telecommunications option of the Japanese cellular telephony system (PDC). All use smart cards to identify the user currently using a phone. Unlike MPA, these solutions provide user mobility within only one network type. MPA provides user mobility across all network types.

The Iceberg [JBK98] project has goals similar to those of MPA, but takes a different approach. Iceberg's functionality depends heavily on a preexisting network infrastructure involving a large number of nodes called Iceberg Access Points (IAPs). The Iceberg design calls for tracking information and user preference information to be distributed among IAPs throughout the network infrastructure. As a result, the IAPs can avoid the level of indirection and delay added by our Personal Proxy by setting up direct and possibly shorter communication paths between the sender and receiver. However, the Iceberg design also requires that IAPs be installed in each type of network supported. For PSTN (Public Switched Telephone Network) or cellular networks, this requires modifying switches or base stations. Many small service providers may have trouble convincing telephone companies to give them access to their base stations. Given the large number of international telephone standards and network operators, it may be difficult to deploy Iceberg widely. Furthermore, Iceberg requires the user to trust several entities (the IAPs) that are not under the user's (or even necessarily a trusted third party's) complete physical control.

The TOPS architecture [AGK+99] provides both host and user mobility for telephony over packet networks. It shares with MPA the notions of a person-level addressing scheme, translating online IDs into application-specific addresses, tracking the current location of users, and converting between incompatible formats. An important aspect of TOPS that we would like to incorporate into MPA in the future is its support for capability negotiation between the caller and the callee. A key difference between TOPS and MPA is that TOPS mainly targets telephony-like applications, where real-time voice and video are the predominant content types, whereas MPA targets all communications applications, synchronous or asynchronous. Also, TOPS pushes all filtering functionality into the Directory Service. MPA requires a Directory Service only to locate a Personal Proxy; all subsequent computation, including filtering and authentication is handled by the Personal Proxy instead. We believe this is the only way to allow directory servers to be fast and efficient, without curtailing the functionality delivered to the end user. Furthermore, TOPS exposes a person's point of attachment to others. MPA presents a black-box view to callers, allowing for better protection of the mobile person's location privacy. Another key difference is that TOPS requires all end-user applications to be rewritten. MPA can be fully operational using current applications and a Personal Proxy, which makes it much more readily deployable.

The SPIN project [LRQS97] at the Canadian National Research Council has designed a seamless messaging system whose goal is to intercept, filter, convert and deliver "multi-modal" messages including voice, fax, and email messages. Like MPA, the system performs tracking to determine the availability of the user and places emphasis on maintaining application-independent filtering rules. However the SPIN project does not try to preserve a user's location privacy. Instead this information is made globally available throughout the system. SPIN also assumes that for every data format, there will be a converter that transforms the format into a standard text format. The filtering rules are then applied to the standard format. While this eliminates the need for path planning, it introduces two problems. First, there are some data formats such as images that cannot be converted to the standard text format. Second, adding a new data format requires writing a new converter, because existing off-the-shelf converters are not likely to transform their input into the standard SPIN format. In MPA, we leverage off of existing converters and simply write wrappers around them to integrate them into the Conversion Planner.

Transcoding proxies have been researched extensively. In work by Fox et al. [FGBA96, FGCB97] the goal is to accommodate clients with limited resources and provide ways to make transcoding more scalable. Our transcoding approach does not try to improve on the results above; instead, we build smart transcoding paths, and allow the transcoding process to be coordinated with filtering and prioritization.

Many applications exist which allow the user to define rules to filter and categorize electronic mail based on metadata and keywords. This idea was explored early on in the Information Lens, a research system whose goal was to facilitate information sharing within organizations [MGT86]. Although the system allowed people to compose complex rules for messages written using semistructured templates, most people created fairly simple rules for tasks such as processing messages from distribution lists [MMC+89]. Since the Personal Proxy has no control over application-specific communication formats, it must rely on the structure provided by each application. However, the Proxy can perform conversions and route between different communication applications as well as filter and categorize, so it can provide users with a much richer set of tools for managing communications.

6 Conclusions

Ubiquitous network connectivity for devices does not automatically imply continuous reachability for people. Whereas existing mechanisms have addressed host mobility or the mobility of people within one network, few have allowed *people*, the ultimate and most important endpoints of communication, to roam freely, without being constrained to one location, one application, one device, or one network.

We propose the Mobile People Architecture (MPA) to maintain person-to-person reachability. MPA distinguishes itself from similar systems by its emphasis on protecting the user's privacy, being extensible to new network devices and applications, and being easy to deploy.

Our prototype person-level router, the Personal Proxy, currently interoperates with telephony, email, and ICQ (an instant messaging program). It filters out spam using a powerful rule-driven engine based on the mobile person's preferences. We show that the Personal Proxy restricts the trusted computing base to a single host under the physical control of the user. In addition, we show that given a library to interoperate with a new network device or application, the Personal Proxy can be extended in less than 200 lines of Java code. Finally, we show that a Personal Proxy can be deployed by an individual and used by all of that individual's correspondents without the need to modify the applications and devices they use.

The latest updates on the Mobile People Architecture can be found at http://mpa.stanford.edu/.

7 Acknowledgements

We would like to thank Ichiro Okajima of NTT DoCoMo for providing us with valuable questions and suggestions during our MPA discussions.

This research has been supported by a gift from NTT Mobile Communications Network, Inc. (NTT DoCoMo), a grant from the Keio Research Institute at SFC, Keio University and the Information-technology Promotion Agency in Japan, and a grant from the Okawa Foundation.

References

[AGK+99] N. Anerousis, R. Gopalakrishnan, C.R. Kalmanek, A.E. Kaplan, W.T. Marshall, P.P. Mishra, P.Z. Onufryk, K.K. Ramakrishnan, and C.J. Sreenan. TOPS: An Architecture for Telephony Over Packet Networks. *IEEE Journal of Selected Areas in Communications*, 17(1), January 1999.

[FGBA96] Armando Fox, Steven D. Gribble, Eric A. Brewer, and Elan Amir. Adapting to Network and Client Variability via On-Demand Dynamic Distillation. In *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 160–170, Cambridge, MA, October 1996.

[FGCB97] Armando Fox, Steven D. Gribble, Yatin Chawathe, and Eric A. Brewer. Cluster-Based Scalable Network Services. In *Proceedings of the Sixteenth Symposium on Operating Systems Principles*, pages 78–91, Saint Malo, France, October 1997.

[ICQ] ICQ, Inc. http://www.icq.com/.

[JBK98] A. D. Joseph, B. R. Badrinath, and R. H. Katz. The Case for Services over Cascaded Networks. In *Proceedings of WOMMOM '98*, October 1998.

[Lam73] Butler W. Lampson. A Note on the Confinement Problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[LRQS97] Ramiro Liscano, Impey Roger, Yu Qinxin, and Abu-Hakima Suhayya. Integrating Multi-Modal Messages across Heterogeneous Network. In *Proceedings of the IEEE International Conference on Communications*, June 1997.

[MGT86] T. W. Malone, K. R. Grant, and F. A. Turbak. The Information Lens: An Intelligent System for Information Sharing in Organizations. In *Proceedings of CHI '86*, 1986.

[MMC+89] W. E. Mackay, T. W. Malone, K. Crowston, R. Rao, D. Rosenblitt, and S. Card. How Do Experienced Information Lens Users Use Rules? In *Proceedings of CHI '89*, 1989.

[MRS+99] Petros Maniatis, Mema Roussopoulos, Ed Swierk, Kevin Lai, Guido Appenzeller, Xinhua Zhao, and Mary Baker. The Mobile People Architecture. In *ACM Mobile Computing and Communications Review*, July 1999. To appear.

[Sta94] William Stallings. *Data and Computer Communications*. Macmillan Publishing Company, 4th edition, 1994.

[UMT] The Universal Mobile Telecommunications System. http://www.umts-forum.org/.

[WKH97] M. Wahl, S. Kille, and T. Howes. Lightweight Directory Access Protocol (v3). RFC 2251, 1997.