An Architecture and Implementation of Secure Device Communication in Oxygen

by

Todd Jason Mills

B.S. Computer Engineering

University of Illinois at Urbana-Champaign, May 1999

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2001

[June 2001]

Author.....

Department of Electrical Engineering and Computer Science

May 17, 2001

Certified by.....

Srinivas Devadas

Professor of Electrical Engineering and Computer Science

Thesis Supervisor

Accepted by.....

Arthur C. Smith

Chairman, Department Committee on Graduate Students

An Architecture and Implementation of Secure Device Communication in Oxygen

by

Todd Jason Mills

Submitted to the Department of Electrical Engineering and Computer Science
on May 17, 2001 in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes the design and implementation of a system for securely controlling and communicating with devices. The system requires little processing overhead on the device while allowing secure control of the device through a proxy. The proxy is a software representative for the device that runs on a fast computer and so is capable of implementing sophisticated cryptographic algorithms to authorize control of the device. With the proxy doing the difficult work, the device must only implement simple algorithms. The devices communicate using a RF protocol that allows mobility and the ability to easily add new devices to the environment. In addition, the devices communicate over the Internet so they can be controlled from virtually anywhere. The system design incorporates well with resource discovery services that facilitate device automation. This thesis describes one such application.

Thesis Supervisor: Srinivas Devadas
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my supervisor, Srini Devadas for his support and advice in completing this thesis. Also, I'm grateful for my partners in crime, Ali Tariq and Matt Burnside. They provided useful comments and helped brainstorm many of the ideas presented in this thesis. I also thank John Ankcorn whose encouragement and support came at just the right time. He kept me focused and always had useful advice in our many discussions. Also, I thank Bodhi Priyantha who helped me incorporate his Cricket location system into my design and Anant Agarwal for his guidance.

I am thankful for all the people that made life at LCS interesting. Ed Suh with whom I ate "junky" lunches, my officemate Dan Engels, Daisy Paul, Mark Huang, Prabhat Jain, George Hadjiyiannis, and Sandeep Chatterjee.

Finally, a big thank you goes to my family. Their constant love, encouragement, and support allowed me to live up to my potential and make it through these many years of education.

Contents

List of Figures

List of Tables

1 Introduction

Many devices that currently exist are difficult to control from remote locations, cannot interact with each other, and require the user to be within the physical proximity of the device to use it. Examples of these devices are radios, televisions, telephones, thermostats, and lights. If these devices were connected so that they could communicate with each other, we could automate many tasks. We would no longer need to do simple things such as turning down the radio volume when someone is on the phone as the devices could do this automatically.

Most simple devices today already contain enough processing power to automate some simple tasks, but by letting the devices communicate with each other we could further enhance their automation. For example, most coffee makers have a small processor that lets the user program in the time for when they want the coffee to be made. This feature is useful but it could be made more useful if the coffee maker could communicate with the user's alarm clock. Then the coffee maker could automatically make coffee a few minutes before the user's alarm clock goes off. The user does not need to worry about setting two alarms, one on the coffee maker and one on the alarm clock, but just needs to set the alarm clock for when he/she wants to wake up. Another example is using this automation to conserve power. A system using the infrared sensors from a security system could automatically turn off the lights if no one is in the room and could turn down the thermostat if no one is in the house. These are just some simple examples but once all devices are connected to each other, an infinite number of possibilities will exist to automate and control them.

Connecting the devices is one problem in automating device usage but another more significant problem is the data the user accesses or enters into the system should stay private and secure. We use the term security to mean only authorized users are allowed to control the devices and the communication is kept private. Ideally, we want the devices to be controlled from anywhere, which we can do by allowing them to communicate over a network implementing the Internet Protocol (IP). As all the different IP networks are connected to one another forming the global Internet, the devices will be able to receive communication from anywhere. Then a user can control and access their devices from any computer with Internet access whether that computer is right next to the device or thousands of miles away. This also means that anyone with Internet access can potentially communicate with the devices, which gives a huge opportunity for accidental and malicious attempts to control the devices. Therefore,

we need security to provide a way for the devices to authenticate communication it receives so that it only responds to the appropriate users.

We have designed and implemented a system that allows devices to communicate securely and only accept commands from authorized users. In addition, our system was designed to be appropriate for very simple devices where we do not want to significantly increase the cost of the device. Our system requires a small amount of processing power on every device that allows it to execute simple cryptographic algorithms. Every device is then associated with a purely software component called a proxy that communicates with the device using the simple cryptographic algorithms. The proxy runs on a computer with a significant amount of computational power and controls what communication is sent to the device by authenticating users using cryptographic algorithms. These algorithms may require large amounts of computation such as public-key cryptography that cannot be easily incorporated on the device. The proxy can then advertise the services of the device in a resource discovery system and act as a middleman for communication to the device.

1.1 Goals

Our system was designed as part of Project Oxygen at the MIT Laboratory for Computer Science to help automate our everyday lives and accomplish more by doing less [13]. To do this our system needed to be easy to use which implied the system must be flexible and convenient. For flexibility, we wanted connectivity between the devices. For convenience, we wanted mobility. While connectivity is important, it brings up issues of robustness that can be solved through the proper use of security.

The five goals of our system are security, connectivity, mobility, easy of use, and appropriateness. We discuss each of these goals below.

1.1.1 Security

Security is the main goal of this architecture and is a key feature missing in most other systems. For our system, we want end-to-end security so that we have security protocols and algorithms from the user of the device all the way to the device itself. In addition, we also want the user to have the ability to authenticate the device he or she is communicating with and for the device to authenticate the user. The devices should also be able to encrypt and decrypt data if confidentiality is desired. With security, we will also get robustness, as the devices will only accept authenticated communication so any accidental as well as malicious communication will be ignored.

Our system achieves our security goal by having a small amount of processing power on the devices that can execute some simple cryptographic protocols, which allow it to communicate securely with its proxy. The proxy is responsible for authenticating the different users and determining which users are allowed to communicate with the device.

1.1.2 Connectivity

Since we are building a system that allows devices to communicate one of our goals must be connectivity. Our system hooks up all devices to the Internet through a gateway allowing them to communicate with any computer on the Internet. In addition, our devices do not need any manual configuration to enable this connectivity.

1.1.3 Mobility

As a goal of our implementation, we wanted the devices to be mobile. Mobility makes our devices more flexible as they are not constrained to a stationary location but can be moved around to different locations. We designed our devices to communicate using a radio frequency (RF) protocol so that they do not require physical wires to communicate which allows our devices to maintain connectivity no matter where they are located. Of course, there are stationary devices that do not require mobility but adding the capability to be mobile does not affect their function.

1.1.4 Ease of Use

As with all systems ease of use is an important goal. Our system is designed to not overburden the user with security mechanisms that are too difficult to use or make it difficult to add new devices. Since our system uses RF to communicate, the user also does not need to worry about hooking up cables for communication. Our system also does not require a central point of administration for all devices. Each user is responsible for the devices they own and they can be set up independently of any other devices.

1.1.5 Appropriateness

Our final goal is an appropriate solution. We want a solution that fits well with the intended devices. Many of the devices are very simple and so our system only requires a minimal amount of computation on each device. This computation is used to implement simple cryptographic algorithms that provide us with security. We do not overburden the devices by requiring large amounts of computational power to implement complex security algorithms.

1.2 Related Work

There are many existing technologies that connect devices and allow them to be controlled to automate tasks. Many of these technologies, however, do not focus on the security of the devices or they assume that the devices have the ability to implement complex security algorithms.

1.2.1 X-10

X-10 is a simple protocol that allows the control of devices over 110-volt house wiring [27]. It works best for appliances that can be just turned on or off such as lights or coffee makers but there are also other devices that have been modified to work with X-10 such as thermostats and window drapes. The X-10 protocol allows it to address up to 256 different devices and send control messages to these devices. When installing X-10 enabled devices, the user must manually configure each one with a unique address. These devices can then be controlled using a remote control RF transmitter or with special hardware, and they can be controlled by software programs on a computer.

The drawbacks of X-10 are that it requires manual configuration and it is not secure. Every device needs to be manually configured so that it gets a unique address. In addition, the user must keep track of which address they assigned to which device so that they are able to control the device they want. X-10 is also not secure, allowing other people on the same 110-volt power feed to have unauthorized control of devices. X-10's lack of security also makes it less robust and more susceptible to accidental control of devices.

1.2.2 Jini

Another technology is Sun's Jini, which provides a resource discovery system for linking together clients and services [11]. Jini contains a lookup service where service providers register themselves and clients use to find out what services are available. To access the services, clients download proxy software that each service provider puts in the lookup service. The clients then use the proxy to interact with the service.

Jini's main advantage is that it allows clients and servers to use any protocol they like to communicate. Since the client downloads a proxy that the server provides, the server can use any protocol it likes to communicate with its proxy. This allows services to use a protocol best suited to the service they are providing.

Jini has two main drawbacks; firstly, it requires the devices to have a Java Virtual Machine (JVM), and secondly, it is not very secure. Since Jini is based on the Java programming language, every device that wants to work with Jini needs a JVM. A JVM is expensive in terms of processing power and memory requirements, which is not ideal for use with resource-constrained devices. The current Jini security model does not have good authentication capabilities although there has been work on fixing this by combining Jini with UIUC SESAME [1] and by using the Simple Public Key Infrastructure (SPKI) [7][6]. Even though both of these systems help fix the security of Jini, they both use expensive public-key cryptographic operations. Work combining Jini with SPKI certificates has shown severe performance penalties. Although the code was not optimized, it took 983 ms to execute the first remote method call and 300 ms for subsequent calls. These times correspond to execution on a 750 MHz AMD Athlon CPU which just shows how computationally expensive some of these public-key cryptographic algorithms are. For simple devices, there is no way they will be able to execute such algorithms without large unacceptable delays.

1.2.3 Resurrecting Duckling

The Resurrecting Duckling is a security model for ad-hoc wireless networks [22][21]. In this model when devices begin their lives, they wait to be imprinted. A master imprints a device by being the first one to communicate with it. After this imprinting, a device only listens to its master. In this way the master is like the mother duck and the devices it controls are its ducklings. During the imprinting, the master is put in physical contact with the device and they share a secret key that is then used for symmetric-key authentication and encryption. A master can also delegate the control of a device to other devices so that it does not become a communication bottleneck. The master can also force devices to die and then resurrect so that they can then be imprinted by a new master.

We have extended the Resurrecting Duckling model for our system. In our system, we have a master for each device called a proxy. The proxy allows our system to be easily integrated with resource discovery systems as the proxy can act as an interface for the device to these systems. In addition, our proxies have the ability to authenticate users and authorize certain users to control the device.

1.3 Contributions and Roadmap

This thesis presents the design, implementation, and evaluation of a system designed to allow devices to communicate securely. Our system consists of devices that can be controlled

securely by proxies that are software guardians for the devices. Our system architecture resulted from attempting to satisfy several goals including security, connectivity, mobility, ease of use, and appropriateness.

In Chapter 2 we discuss the architecture of the system and the reasons behind having a proxy for each device. We also discuss two different security models, the direct and indirect models. This chapter ends with a discussion on how devices and their corresponding proxies are initialized.

Chapter 3 discusses our implementation of the proposed architecture. We begin this chapter by describing the communication protocol we chose to use and how using RF communication allows the devices to be mobile. Then we discuss the actual security algorithms that we have implemented on the devices and how they work. This chapter then goes on to explain the actual hardware that we built to control devices and has a discussion on our choice of electronic components. Next, the chapter discusses the design of the software that we implemented for the system. The chapter ends with an evaluation of the design in terms of how much memory and processing power is required by a device to implement our security algorithms and communication protocol.

In Chapter 4 we discuss how our system can be integrated with other systems that provide device-to-device communication and allow scripts to be run that control the devices. In this chapter, we also discuss an application we built using our implementation of the secure device communication system described in this thesis.

Finally, Chapter 5 concludes the thesis.

2 System Architecture

The main goal that largely influenced the architecture is security. When we talk about security, we are interested in three goals, authentication, authorization, and privacy. By authentication, we mean that the party receiving communication can verify the sender of the message. Authorization means the system can determine who is allowed to do certain operations. Privacy means that the communication between the two parties should be kept confidential and eavesdroppers should not have the ability to figure out what is being sent. To fulfill these requirements we need a system with end-to-end security all the way from the user controlling the device to the device itself. In addition, we want a solution that is appropriate for the devices we are controlling. If we want to control a light bulb then we do not want to add expensive processing requirements just so that it can be secure. Ideally, we only want to add a cheap, simple micro-controller that has the minimum amount of processing power to enable the light bulb to be controlled securely.

Security is difficult to handle, however, when a device may have multiple users wanting to control it. With multiple users, a device needs an access control list (ACL) that tells the device who has permission to do what operations. The problem with an ACL is that it requires memory to keep this information and the device may be severely limited in the amount of memory it has available. One solution to this problem would be to give each device a secret key and only users that know this secret key can control the device. All the device then needs to remember is this one secret key. This solution may work but it has definite drawbacks. One drawback is that it is difficult to change the access permissions for individual users. If we give out the secret key to five different people but later on we want to revoke someone's access to the device, we must change the secret key in the device and then give the new secret key only to the users we still want to have access. Managing who can and cannot access the device becomes a nightmare. Another drawback is that the secret key can be easily duplicated letting others have control of the device. This would not be as much of an issue if each person had his or her own secret key since sharing your key would essentially let others impersonate you. In this case sharing the key only lets someone else control the device and does not let him or her impersonate you so there are no consequences for sharing the key.

Even if the devices have enough memory to store an ACL, we still need some method to authenticate users. One option would be to use symmetric-keys where each user that wants to access a device shares a secret key with the device. The problem is that this system is difficult to

manage and does not scale well. A much better solution would be to use public-key cryptography. In public-key cryptography, each user has a private key that only they know and a public key that everyone knows. With this system, the device can authenticate users by having them sign a message with their private key and then verify that the user is authentic by checking the message with their public key. Using this system the devices need an ACL with the public keys of users that are allowed access and the user only needs to remember one key, their private key.

While public-key cryptography seems like an ideal solution in practice, it requires significant computational power. A common public-key cryptographic algorithm, RSA with 1024 bit keys, takes 43 ms to sign and 0.6 ms to verify on a 200 MHz Intel Pentium Pro [25]. This is using a 32-bit processor. Most of our devices will have small 8-bit micro-controllers running at a few megahertz so it will take quite a while to do these computations on such a processor.

Since most common devices that we want to control have little or no processing capabilities, we need a system that does not require each device to have an expensive amount of processing power. We do not want to significantly increase the cost of a device. We would also like to have the benefits of using an ACL and public-key cryptography.

To deal with these problems our architecture creates a separate software component for each device called a proxy. The device's proxy controls access to the device while the device itself consists of some software plus any physical hardware that is being controlled. The proxy does not need to run on the same piece of hardware as the device; they only need to be able to communicate securely with each other. For example, the proxy and device could be separated by large distances and communicate over the Internet. Having the device and the proxy separate is the most important part of our architecture.

Our architecture creates a proxy for each device so that the device can be kept as simple as possible. Since the proxy is a purely software component it can run on any computer as long as it can somehow communicate with the device. Many proxies can also run on the same computer, which allows us to amortize the cost of the proxies since they may require a significant amount of processing power and memory to control access to the device. Since the processing and memory capabilities of the device are kept as simple as possible, we keep any additional cost for the device very low.

2.1 Devices

We are focusing on devices that have a small amount of processing power and memory. Although devices with larger amounts of processing power could also be used, we are focusing on the common denominator that all devices possess. This means the devices are most likely controlled by a simple 8 or 16 bit micro-controller running at a few megahertz. The devices typically take small control commands and output simple state. An example device that fits this model is a radio as it has simple input such as turning it on and off, adjusting the station, and adjusting the volume. The state it outputs is also simple consisting of just the current station and volume settings.

Devices also need some way of communicating with their proxies. Proxies that run on the device already have the ability to communicate but for proxies that are separated from the devices, we need a method for them to communicate. A device and proxy could communicate using wireless communication such as radio frequency (RF) or infrared or it could use a wired solution like Ethernet. They could also use a combination of these methods.

The device and proxy also need some sort of reliable protocol built on top of the communication medium. For devices with enough computational power, they could just implement standard Internet protocols like TCP/IP. For other devices, we may want to keep the protocol very simple due to the limited availability of processing power. An effective protocol would be one where the receiver acknowledges every packet of data received and the sender keeps on sending the packet until it gets an acknowledgement. While this protocol may have a high latency, it does not matter because with these simple devices we are not sending enormous amounts of data to the device. We are just sending simple commands.

The devices may also contain a component that allows them to determine their location. Devices that know their location can provide this to a directory service so that services based on location can be provided.

2.2 Proxies

The proxy is a software component that runs on a network-visible computer. The proxy's primary purpose is to provide access control for a device. The proxy can also handle other functions such as running scripts on behalf of the device and communicating with a directory service.

The proxy can implement computationally-expensive security algorithms since it runs on a computer that has significantly more processing capabilities than the device. The proxy can

also keep large access control lists that would not fit in the device's memory. The proxy uses these mechanisms to act as a guardian for the device. The proxy authenticates users and only allows those with valid permissions to control the device. The proxy can use any type of mechanism to authenticate users such as password authentication with SRP [26] or by using public-key cryptography with certificates such as SPKI.

The proxy runs on a computer the owner of the device trusts. Many proxies can run on a computer and be grouped together in proxy farms. Proxy farms allow the user to easily set permissions for all devices as all proxies within a proxy farm can have the same access permissions. The user can easily change the access permission for many devices by just making one global change to the proxy farm.

2.3 Security Models

The proxy and device share a secret key. This secret key allows them to communicate using symmetric-key authentication and encryption. Symmetric-key operations take much less processing power than public-key so the device can do this computation with a small micro-controller. Since the proxy and device can communicate securely there are different methods to allow a user to communicate with a device, indirect, direct, and a combination of both.

2.3.1 Indirect

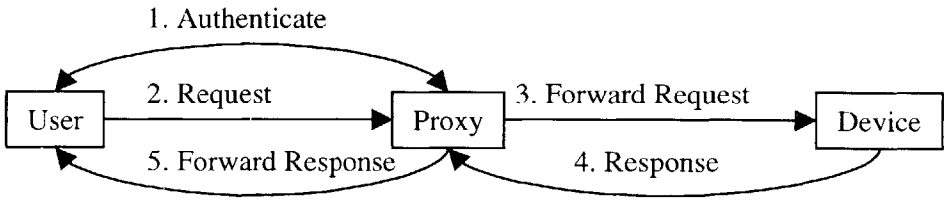

Figure 2-1: Communication flow in the indirect security model

In the indirect security model all communication passes through the proxy. The proxy authenticates the user and then routes any further communication from the user to the device. The flow of communication is shown in Figure 2-1 with each step described below.

1. The proxy authenticates the user and at the same time, the user authenticates that he or she is talking to the correct proxy. They also set up a secure communication channel.
2. The user sends their request to the proxy.
3. The proxy checks its ACL to make sure the user is allowed to do the operation specified in the request. The proxy forwards the request to the proxy if the user is

allowed to do the operation. If the user is not allowed to do the operation the proxy would send back an error letting the user know.

4. The device responds to the request and sends the response back to the proxy.
5. The proxy takes the response from the device and forwards it on to the user.

This model works well for several different scenarios. The first is when the proxy does some processing on behalf of the device. In this case, the proxy needs to receive all communication so that it can use this information to do the processing for the device. For example if we have a thermometer as our device that has many requests for the temperature then the proxy can help speed up the responses by caching the temperature from the thermometer. Since the thermometer and its proxy may communicate over a slow link it makes sense for the proxy to receive all requests so that it can send back responses faster than having every request go to the thermometer.

This model also works well when the device needs to communicate with many different users. In this case, our device may have little processing power and so if the device needs to communicate with multiple users at once it may not have the resources to do this. However, using the indirect model, the proxy can buffer requests for the device and handle them one at a time. This way the device only needs to communicate with one other entity, the proxy, and does not need to know or remember all the different users it is executing requests for.

The indirect security model does not work well when there is a large latency in routing the communication through the proxy. In this case, the direct security model, discussed next, will work better.

2.3.2 Direct

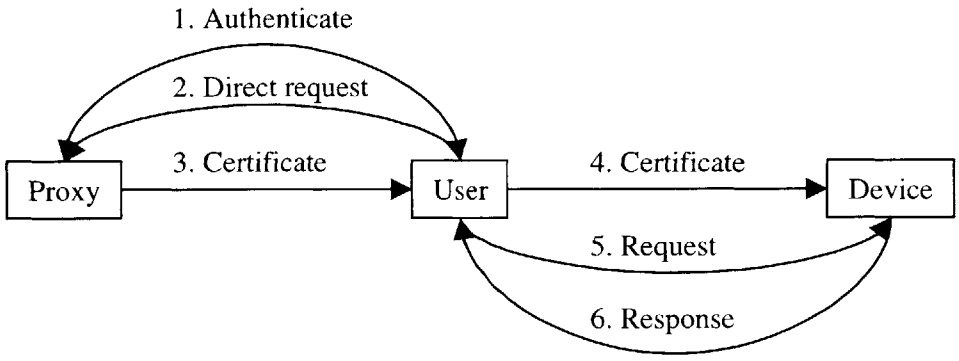

Figure 2-2: Communication flow in the direct security model

The direct security model allows the user to communicate directly with the device. The user then does not need to rely so much on the proxy as it does in the indirect model. This model works by giving both the device and the user a secret key that they can use for symmetric-key authentication and encryption.

Figure 2-2 illustrates the communication flow in this model with each step described below.

1. The proxy authenticates the user and at the same time, the user authenticates that he or she is talking to the correct proxy. They also set up a secure communication channel.
2. The user requests from the proxy that it can have direct communication with the device.
3. The proxy checks to see if the user should be allowed to have direct access to the device and then sends a certificate to the user. The certificate contains a new randomly generated secret key called K_U and a time when that key expires.
4. The user sends the certificate to the device informing the device the user is allowed to control it.
5. The user encrypts and authenticates its request for the device with the secret key K_U.
6. The device authenticates and decrypts the request with K_U and then sends back a response authenticated and encrypted with K_U.

The certificate the proxy sends to the user contains a secret key, K_U, and an expiration time, T_{EXP}, that tells when this key expires. The time could be measured in terms of the number of requests a user can make or an absolute time of when the certificate expires or perhaps a combination of both. The expiration time is a way of making the user ask the proxy again for permission to access the device when the time has expired. It makes revoking access to a device much easier than having a revocation mechanism. The proxy can revoke a user's right to access a device by just not granting them another certificate when the current one expires.

The certificate is encrypted and authenticated using the master key that the device and proxy share. This way the user can send the certificate on to the device and the device knows the certificate is valid and generated by the proxy. This also keeps K_U secret so that an eavesdropper cannot figure it out. Since the user cannot figure out what is in the certificate the proxy tells the user the contents of the certificate over the secure communication channel.

A proxy may also embed in the certificate what actions the user is allowed to perform. A proxy may not want to let the user have the same type of control it has over the device but just a subset of the controls. The proxy most likely will not want to allow the user to generate new certificates for the device or change the device's master key. Also depending on the user that wants access to the device the proxy could selectively choose what parts of the device they can control. For example, on a radio the proxy may let some users change the station while others can change both the volume and the station.

The direct security model works best when it is much faster to talk with the device directly than to have the communication routed through the proxy. This method removes the indirection of having to always communicate with the device through the proxy. If the proxy is slow or does not have a good connection to the network, this model will still allow the user to communicate with the device without significant delays.

This model also works well when the proxy and the device have no way of communicating with each other. In this model, we have separate communication channels, one between the user and the proxy and one between the user and the device. The proxy never needs to communicate directly with the device.

This model will not work very well, however, if the proxy needs to do processing on behalf of the device since the proxy will no longer receives all the communication. Also if the device needs to communicate with many users this model may not work well because the device may not have enough memory to keep all of the symmetric-keys needed to communicate with all the users.

2.3.3 Combination

The two security models, direct and indirect, can also be combined together for more versatility. This allows the user to decide the best way to communicate with the device and choose either option.

2.4 Initialization

New devices need to be initialized so that a proxy is assigned to them and so that they share a secret key with the proxy. This is done by physically touching the device to the computer that will run the proxy. When the device is touched to the computer, the proxy farm on that computer creates a new proxy for the new device. The new proxy gets the same access permissions as are defined for the proxy farm. The proxy then generates a random secret key that it shares with the device. The proxy also tells the device its address so the device knows how to

contact the proxy. This initialization is straightforward and easy for the user installing the device. The user does not need to set any switches on the device or do any manual configuration.

3 Implementation

We have built devices that are wireless, can determine their location, and can communicate securely. These devices follow the architecture described in Section 2. This section will describe the implementation of these devices including how they communicate, the types of security algorithms they use, and how they determine their location. We will also evaluate the performance of the devices in terms of how much memory and processing power is required to implement them.

3.1 Communication

We had two main alternatives to choose from for device communication, wired and wireless. Having wired communication means that each device needs to be physically wired to some communication medium. If all devices have wires connecting them, it can be quite difficult to manage and requires a lot of infrastructure to hook up all of the devices. Also having every device wired limits their mobility. Wireless communication on the other hand allows the devices to be mobile. It also does not require as much infrastructure as wired communication. For these reasons we chose to implement wireless communication for our devices. Of course, for some devices that are stationary, a wired solution may be more appropriate.

There are different types of wireless communication that we had to choose from. The first type is communicating by using light such as infrared or laser. This requires the devices communicating to be in line of sight with each other and have their transmitters and receivers oriented to face each other. The other type of wireless communication uses RF. The benefit of RF is that the devices do not need to be in the line of sight of each other since RF travels through walls and other obstacles. We chose to focus on RF communication because it allowed us greater versatility in that we did not need to worry about where the devices were located and if they had an unobstructed view for communication. RF communication also makes the system easier to use, which is one of our goals.

3.1.1 Overview

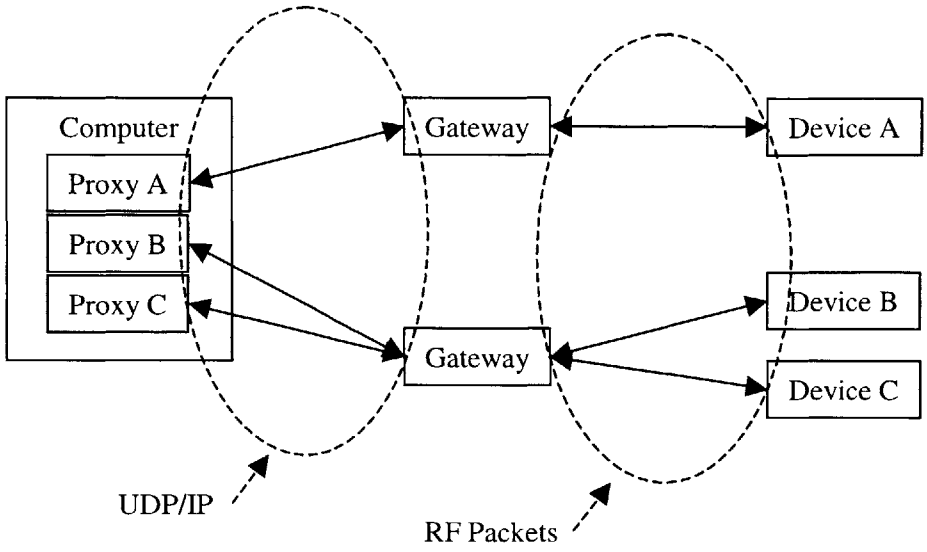

Figure 3-1: Communication Overview

The devices communicate using RF and the proxy needs to be able to communicate with the devices. This is done using a gateway that is able to listen to the RF communication from the device and translates the RF packets into UDP/IP packets that can be sent over the network to the proxy. Of course, the gateway also works in the opposite direction by converting UDP/IP packets from the proxy to RF packets and transmits them to the device. An overview of the communication is shown in Figure 3-1. This figure shows a computer running three proxies for three separate devices with device A using a different gateway for communicating than device B and C.

3.1.2 Packet Format

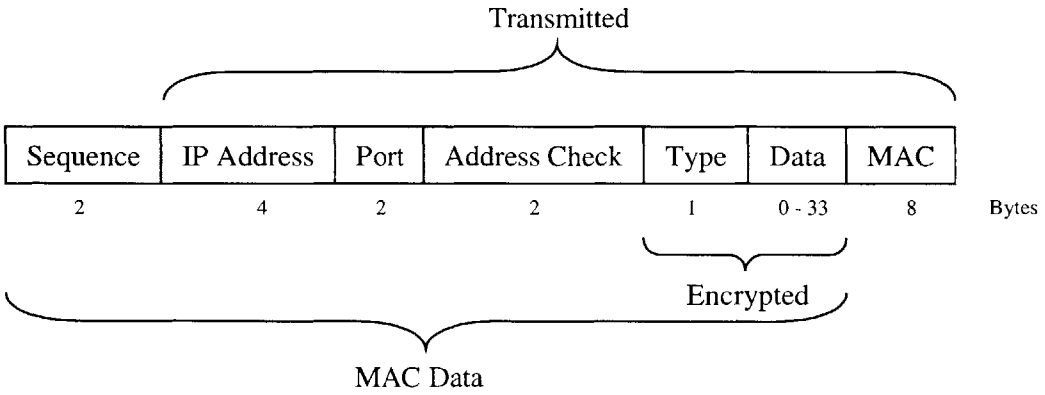

Figure 3-2: Packet Format

The proxy and device communicate using the packet format shown in Figure 3-2. Each packet contains a sequence number, an IP address, a port, a type, data, and a message authentication code (MAC).

Each packet is addressed with an IP address and a port number. The proxy's IP address and port correspond to the port it listens on for UDP packets from the device. The device's IP address is the same as its proxy and its port is assigned by the proxy. The device port and proxy port must be different so that the device and the proxy have unique addresses. The IP address and port of the packet also have two address check bytes that provide error detection over the address. This allows the gateway to determine if the address is correct before sending the packet to the proxy.

Every packet contains a type and may contain up to 33 bytes of data. The type tells what kind of packet it is such as a command or acknowledgement. The data sent is device dependent and is limited to 33 bytes to keep the packet from being too large. Both the packet type and data are encrypted for privacy.

All packets also have a sequence number that the proxy keeps incrementing every time it sends a new message. This allows the proxy and device to stay synchronized and prevents the same packet from being processed more than once. The sequence number in conjunction with the MAC also prevents replay attacks.

The last part of every packet is the MAC. The MAC authenticates the message so that the recipient of the message knows it came from a source authorized to communicate with it. The MAC is computed over the whole message after the type and data are encrypted. The MAC also does double duty as it also allows us to find errors in the packets. If a packet has an error then the MAC will be incorrect and the packet will be rejected, which is the behavior we want. This also eliminates the need for packets to have some type of error detection bytes like a cyclic redundancy check.

Another optimization in the packets is that even though they do have a sequence number, the sequence number is actually never sent. This keeps down on the amount of data that needs to be transmitted. The reason we do not need to transmit the sequence number is that the receiver already knows what sequence number to expect next. It expects one more than the last packet's sequence number. Since the packet's MAC is computed over the sequence number the receiver can verify that the packet is the one it is expecting by inserted its expected sequence number at the beginning of the data received and verifying the MAC. If the MAC is correct then the data corresponds to a packet with the expected sequence number.

We could have made other optimizations to the RF packets also. For instance, when transmitting a packet from the proxy to the device the address and port of the device do not actually need to be transmitted. They could be handled in the same way that the sequence number is handled. Each device would append its address and port to the packet received before computing to see if the MAC is correct. If the MAC is correct the device knows the packet was meant for it. This optimization could also be beneficial for security reasons since the device's address would not be transmitted it would be much more difficult for an eavesdropper to figure out which device is receiving a packet.

Technically, we also do not need to send the address check bytes in packets from the proxy to the device since the device can use the MAC to check for errors. In addition, the address check bytes may not be needed in communication from the device to the proxy. The gateway uses these bytes to check for any errors in the address before sending the packet on to the proxy specified in this address. If we remove the address check bytes a gateway may send a packet to the wrong destination but this should have little consequence.

There are three reasons we did not just implement the Internet protocol over the RF channel: it would make the RF packets larger, we would need an IP address for each device, and it would be difficult for the devices to be mobile. We elaborate below.

If we make the RF packets larger then the probability of an error occurring during transmission becomes higher. This will decrease the throughput of the RF channel. An IPv4 header takes up 24 bytes [16], which would significantly increase the size of our packets.

Assigning each device an IP address would also be difficult. Either the devices would need to implement a protocol to get an IP address dynamically or it would need to be set by the user. Having the user set the IP address would make the devices difficult to use since it would require manual configuration to get them working. On the other hand dynamically acquiring an IP address may work but this means there needs to be more code in the device to handle this protocol.

The main reason we do not implement the Internet protocol on the devices is that it would be difficult for the devices to be mobile. When the devices move they may move from one gateway to another. These gateways may be on different subnets so the device cannot keep on using the same IP address. This makes it difficult for the proxy to keep in contact with the device if the device keeps on changing its IP address. We would need another protocol for the device to inform its proxy of its new IP address and the device would need to somehow obtain a new IP address. This would make the device's code much more complex and for this reason we did not want to use the Internet protocol to communicate over the RF channel.

The main problem with this packet format is that the device always transmits the address of the proxy. Potentially an eavesdropper could listen for this and deduce where specific devices are located since they always send out this unique identifier. The device could solve this problem by having the address of a few fake proxies and randomly choosing to send the packet between its real proxy and these fake proxies. The fake proxies would then forward the packets on to the real proxy. Another option is to use a system similar to remailer networks [5]. Since remailer networks hide the sender's identity of an email, we could use a similar system to hide the identity of the device sending the packets.

3.1.3 RF Encoding

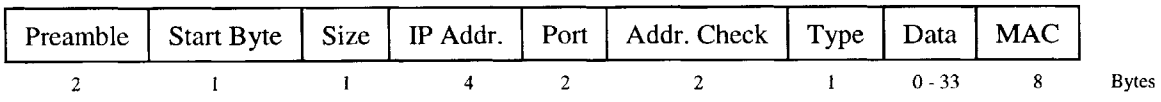

Figure 3-3: RF Encoding

The actual RF transmission appends a header on to the packets that allows the receiver of the RF communication to easily find the beginning and size of the packets. Figure 3-3 shows the format for the RF transmission. First, the sender transmits a preamble; this is a sequence of 1's and 0's used to initialize the RF data slicer with a good threshold. Next, a start byte is sent which is a unique sequence that signifies the beginning of a packet so that the receiver knows when to start sampling for a whole packet. Then the size of the packet is sent and finally the packet is sent.

All of the data transmitted is encoded to maintain DC balance for the RF data slicer. Every four bits of data is converted to a six bit encoding that has an equal number of 1's and 0's. This guarantees the transmission to never have more than four 0's or four 1's in a row. Another alternative to this type of encoding is Manchester encoding which converts a 1 to a 1-0 bit sequence and a 0 to a 0-1 bit sequence which guarantees to never have more than two 0's or two 1's in a row. Manchester encoding doubles the transmission size of the packet. We chose the 4 to 6 bit encoding because it does not need to transmit as many bits as Manchester encoding while still maintaining good DC balance.

3.1.4 Gateway

The gateway translates from UDP/IP packets to RF packets and vice-versa. The gateway has connections to an Ethernet network and to the RF network. It constantly listens on both networks, translates, and forwards packets from one network to the other.

The gateway translates a RF packet to a UDP/IP packet by first looking at the address check bytes of the RF packet. It verifies that these bytes are correct which verifies that the IP address and port of the RF packet are error free. Since the gateway does not have access to the key that produced the MAC at the end of each RF packet, it cannot use the MAC to check for errors. The gateway only knows that the IP address and port are error free but the rest of the packet may have errors. The gateway copies the whole RF packet into the data section of the UDP/IP packet and uses the IP address and port as the destination of the packet.

The gateway translates a UDP/IP packet to a RF packet by just looking at the data in the UDP/IP packet. It assumes that the data in the packet is already formatted as a RF packet and that it can just send it.

The gateway has some intelligence to try to only keep out accidental or malicious communication from flooding the RF network. Since the speed of the RF network is an order of magnitude slower than the Ethernet network, it would be very easy to flood the RF network with packets, which could cause a denial of service. The gateway has a simple mechanism to try to prevent this. Whenever the gateway receives a RF packet, it keeps the IP address and port of the packet in a table and will only forward UDP/IP packets from addresses in the table. The gateway removes addresses from the table if it has not received a RF packet with that address for more than a minute.

3.1.5 RF Protocol

The current communication protocol has the proxy initiating all communication with the device. This keeps the device's code simple, as it only has to respond to messages sent to it. This protocol also helps keep the RF channel from being over utilized with too much communication. If the devices initiate communication then it could be possible to have many transmission collisions since many devices may try to communicate at the same time. Since the RF channel has a small bandwidth then having transmission collisions could severely degrade performance and cause unacceptable delays in communication.

With the proxies initiating all communication, the gateway can act as a guardian over the RF channel. When the gateway receives a message from a proxy, it then transmits it over the RF channel. Since the device will always respond back to a message it receives the gateway waits for about 50 ms for a response from the device. During this time, the gateway will not transmit any messages over the RF channel to prevent a transmission collision between the gateway and the device.

The proxy and the device also need to communicate reliably. The proxy keeps on transmitting the same packet until it gets a response from the device. Since every packet has a sequence number, the device sends back a packet with the same sequence number to acknowledge that it has received the packet. Once the device acknowledges a packet it then starts looking for packets with the next sequence number. If the device receives a packet with a sequence number one less than its current expected sequence number then it just sends the last packet it sent. This way the device will only process each unique packet once.

Devices can also be mobile so this presents a problem in that we still want to maintain communication with the device. The problem is that the device could move out of communication range of one gateway and then into the range of another gateway. The proxy will not know about this new gateway and so will be unable to contact the device. Therefore, there needs to be a mechanism for the device to tell the proxy about the new gateway. Whenever the device has not received a new packet from the proxy for about ten seconds it starts "screaming". When a device is screaming it repeatedly transmits the last packet it sent every few seconds. These packets will be routed through the new gateway to the proxy, which allows the proxy to determine the address of the new gateway.

Devices may need to be optimized as to when they start screaming. For devices that are stationary, they will typically communicate using the same gateway. These devices can have a longer delay of not hearing from their proxy before they start screaming. This way the devices do not need communication from their proxy very often so this limits the amount of packets that need to be sent over the RF channel. For devices that are very mobile, we may want a short delay before they start screaming so that the proxy can quickly find the new gateway and continue communicating with the device.

Another optimization would be for the proxy to keep a cache of the most recent gateways it has used to communicate with the device. Then when the proxy tries to communicate with the device but does not get a response it can try reaching the device through the other gateways in its cache.

3.2 Security

The proxy and device communicate through a secure channel that encrypts and authenticates all the messages. The HMAC-MD5 [12][19] algorithm is used for authentication and the RC5 [20] algorithm is used for encryption. Both of these algorithms use symmetric keys with the proxy and the device sharing a 128-bit key. This implementation only supports the

indirect security model where all communication to the device comes from its proxy. The direct security model could be added on if needed.

3.2.1 Authentication

HMAC is a hashed message authentication code that uses a cryptographic hash function to produce a MAC that can validate the integrity of a message. HMAC works with any cryptographic hash function such as MD5 or SHA-1 [14]. We chose to use MD5 because it requires less computation than SHA-1 and the code is freely available.

HMAC works with a cryptographic hash function, denoted by H, a secret key, denoted by K, and the message data, denoted by D. HMAC is essentially computing H(K, D) for the MAC but the actual computation is a little more complex. Since HMAC uses the secret key then only someone that knows that key can create the MAC or verify that the MAC is correct.

HMAC with the MD5 hash function produces a 16 byte MAC. We take these 16 bytes and append the most significant 8 bytes on to the end of each message. We do this to limit the amount of data that must be transmitted with each packet. From a security perspective this has the advantage of less information for an attacker but the disadvantage that an attacker has to guess fewer bits. We feel this is a good tradeoff since if we include all 16 MAC bytes in every packet then even more of each packet will be devoted to authentication instead of useful data.

3.2.2 Encryption

We encrypt the data using the RC5 encryption algorithm. We chose RC5 because of its simplicity and performance. RC5 does not require tables to speed up processing, as it is mainly XOR operations and rotate operations.

Our RC5 implementation is based on the OpenSSL [15] code. RC5 is a block cipher, which means it usually works on eight byte blocks of data at a time but by implementing it using output feedback (OFB) mode we essentially make it into a stream cipher. This allows us to encrypt an arbitrary number of bytes without having to worry about blocks of data. Also by using OFB mode we only need the encryption routine of RC5, we do not need the decryption routine.

OFB mode works by taking an initial vector and a key and generating an encryption pad. The encryption pad is then XOR'ed with the data to produce the cipher text. Since $X \oplus Y \oplus Y = X$ we can decrypt the cipher text by producing the same encryption pad and XOR'ing it with the cipher text. Since this only requires the RC5 encryption routines to generate the encryption pad we do not need separate encrypt and decrypt routines and can just use the same code for both operations.

For our implementation we use 16 rounds for RC5 and the same 128-bit key we use for authentication.

3.3 Location

We decided to use the Cricket location system [18][17] to find the location of devices. Our primary reasons for choosing this system to determine location are its user privacy, decentralized control, low cost, and easy deployment. It also works indoors which is where most of our devices are most likely to be located. Cricket's user privacy means that only the device can figure out its current location. It's up to the device to decide if it wants to let others know its current location as the locations are not stored in a centralized database. Cricket is also easy for the user to deploy because of its decentralized nature and easy of use.

Other location systems were considered such as the Active Badge system [24] and the Bat system [9][10]. Both of these lacked the user privacy that we found essential for our system since they store the location in a central database. The RADAR system [2] may work well since it does allow for user privacy but it is not as easy to setup and deploy.

Cricket works by having beacons placed on the ceilings of rooms. These beacons periodically broadcast out location information such as "Room 226" that can be listened for by Cricket listeners. The beacons broadcast location information using RF while at the same time sending an ultrasound pulse. A listener listens for the RF message and once it receives the first part of the message it times how long until it receives the ultrasound pulse. The listener uses this timing information to figure out how far away that beacon is and can compare it with the distance to other beacons and knows that its location is the location broadcast by the closest beacon.

Cricket does have some problems in that it is easily confused by other sources of ultrasound. Many common sounds such as jangling keys or coins produce ultrasound that can confuse the Cricket listener to think it is somewhere that it is not. The Cricket listener can avoid this problem by sampling for a longer period before figuring out its location to filter out any errors. The drawback is that there is a latency of about 2 to 6 seconds to determine the current location. However, for may applications, this is not a major concern.

3.4 Board Design

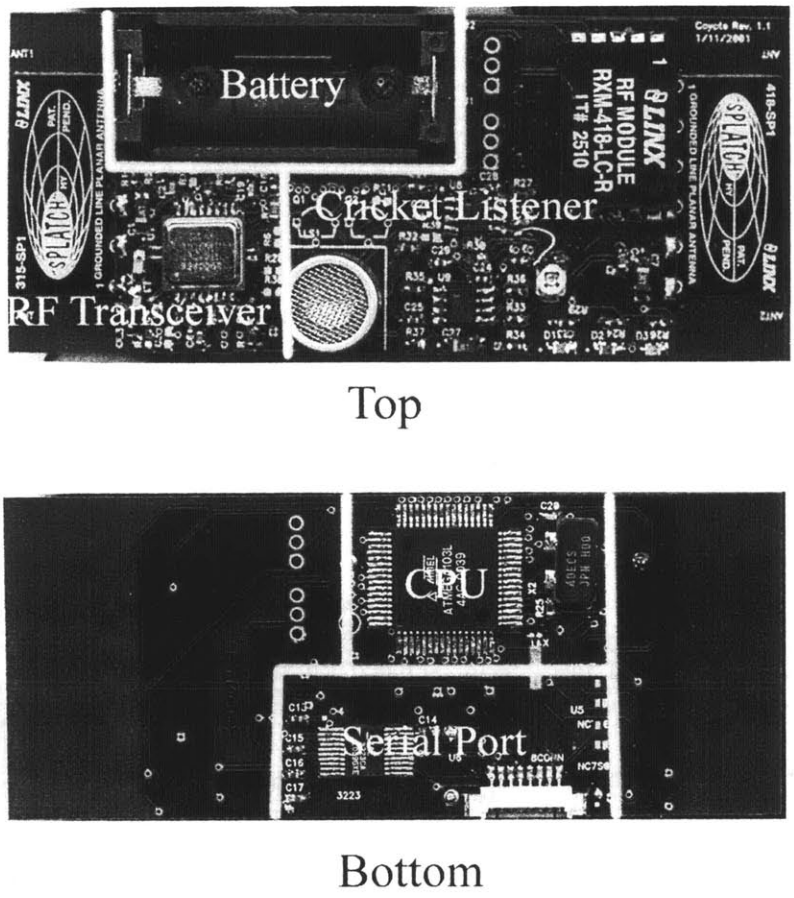

Figure 3-4: Picture of the circuit board

For our implementation, we built a circuit board that acts as the brains of a device. It contains the components to communicate over RF, find its location, implement the security algorithms, and interface with devices. The circuit board can also act as a gateway or a Cricket beacon depending on what components are installed and what software is used. A picture of the board is shown in Figure 3-4 that highlights the major components of the design which are a battery, RF transceiver, Cricket listener, CPU, and serial port. The board size is 43x102 mm.

The battery used is a 3-volt lithium battery that has a nominal capacity of 1,200 mAh. We chose this battery primarily because it stores quite a bit of power for its size. Since this battery is still fairly large relative to the size of the circuit board a coin-type battery could be used to make the board smaller.

The serial port allows the device to communicate with other computers or to control devices that also have a serial port. The serial port is used by gateways to send and receive RF packets to a computer.

The Cricket listener allows the device to determine its location. This component is not needed on all devices; only devices that need to know their location. It consists of an RF receiver to receive the location information from the Cricket beacons as well as an ultrasound receiver to listen for the ultrasound pulse.

The board can also be configured as a Cricket beacon. In this configuration, the RF transceiver component of the board is not needed. The Cricket listener section is replaced with different components to function as a Cricket beacon, which include putting on a RF transmitter to send location information and using an ultrasound transmitter to send the ultrasound pulse.

3.4.1 CPU Selection

Many different CPUs could be used as the brains of our device. In choosing one, we mainly wanted a CPU that would be representative of the processing power the simplest device would have. The CPU we chose to use is the Atmel ATMega103L which is an 8-bit CPU that uses the Atmel AVR instruction set and operates at 3-volts. It has 128KB of flash memory, 2KB of RAM, and 512 bytes of EEPROM. We ran the CPU at 4MHz. The CPU's flash memory is actually quite large and may not represent what most simple devices have but it is useful for the development of software. This CPU is extremely easy to use, as it only requires a crystal and power to operate. All of the memory is internal to the device, which keeps the area the chip needs small and compact. The CPU can also be programmed via a simple cable plugged into the parallel port of a computer.

Table 3-1 shows the ATMega103L processor compared to some other processors. While the ATMega103L may not be the best selection in terms of power, it provided an easy to use development environment with a C compiler. To reduce power the Microchip PIC16F877 processor can be used since it has similar processing capabilities as the ATMega103L that we used in our design. The only drawback is the PIC16F877 has a much smaller amount of both flash and data memory.

	Atmel ATMega103L	Microchip PIC16F877	Texas Instruments MPS430F139
Maximum Frequency (MHz)	4	20	8
Operating Voltage (V)	2.7 to 3.6	2.0 to 5.5	1.8 to 3.6
Datapath Width (bits)	8	8	16
Flash Memory	128 KB	8K x 14 bits	60 KB
Data Memory	4 KB	368 bytes	2 KB
Current (mA) at 4 MHz, 3 Volts	5.5	0.6	1.7
IO pins	48	33	49

Table 3-1: Comparison of different micro-controllers

3.4.2 RF Selection

The device needed some RF component that allowed it to communicate with its proxy via the gateway. An ideal RF component is one that is robust, cheap, consumes little power, and has enough bandwidth for our devices. Quite a few options can be used for the RF component of our design with different tradeoffs.

First, a few major different modulation schemes are used by most RF components. The first is on-off keying (OOK) that is very simple and only sends a binary value, on or off. In OOK the transmitter is either on and modulating the output or it is turned off. Another option similar to OOK is amplitude shift keying (ASK) that allows the amplitude of the transmitting signal to vary over time. A third option is frequency shift keying (FSK) where the frequency of the transmitting signal is varied in time to transmit different values. Typically RF components using OOK and ASK are cheaper to manufacture than those using FSK while robustness increases going from OOK to ASK to FSK [2].

Most typical applications using OOK are cheap controllers like garage door openers, car remote keyless entry, etc. Most OOK solutions provided are a one-chip solution that require few external components and so are easy to use.

New RF communication standards are being designed such as Bluetooth that is designed as a short range communication framework to allow many different devices such as PDAs, mobile PCs, phones, etc. to interact in a user-friendly and efficient manner [8]. Bluetooth RF transceivers operate around 2.4 GHz, have a data rate up to 1 Mbps, and do frequency hopping of 1600 times per second. Frequency hopping makes the communication more robust to interference by constantly changing the frequency it is transmitting at. The Bluetooth

specification also specifies a baseband and link layer. The baseband layer does things like timing, framing of packets, and flow control while the link layer manages connection states. For our devices, we are using a many-to-one communication scheme so only the RF transceiver part of the Bluetooth specification is relevant to us.

Table 3-2 gives an overview of a few RF components. Linx Technologies has separate transmit (TXM-LC) and receive (RXM-LC) chips that have a low data rate but also take a small amount of current. They also only require two external components, an antenna and a bypass capacitor. The RF Monolithics part, TR-3001, has a higher data rate but requires more external components and takes more current to transmit. The Texas Instruments TR6900 uses FSK but takes a lot of current and requires many external components. Finally, the Conexant CX72303 is a Bluetooth RF transceiver chip that has been recently announced and takes a moderate amount of current but gives a high throughput.

We chose to use the RF Monolithics TR-3001 for the device to gateway communication. The reasons are because it had a reasonable amount of bandwidth, does not take much current, and does not need very many external components. The Linx chip's small bandwidth of 4.8 kbps was not adequate because if we have many devices we could easily exhaust the available bandwidth. The Texas Instruments part took too much current to be useful in our battery operated device. Finally, while the Bluetooth part from Conexant has high data rates our devices do not need this large of a data rate. Our devices are designed to be simple and so 1 Mbps would be overkill, however, both the Texas Instruments and Conexant parts do have the advantage that they use more robust modulation schemes. This property may be more important since noise could easily disrupt the simple OOK modulation we are using.

For the Cricket listener we chose to use the Linx Technologies RFM-418-LC since the beacons use the corresponding transmitter. The Cricket listener also operates at 418 MHz while the device to gateway communication operates at 315 MHz so that we do not have any interference between them.

	Linx Technologies TXM-LC	Linx Technologies RXM-LC	RF Monolithics TR-3001	Texas Instruments TR6900	Conexant CX72303
Modulation Type	OOK	OOK	OOK	FSK	GFSK
Frequency (MHz)	315, 418, 433	315, 418, 433	315	850 to 950	2,400
Transmit Current (mA)	3.0 (typical)	N/A	12 (maximum)	21 to 37 (typical)	14
Receive Current (mA)	N/A	6.0 (typical)	4.5 (maximum)	26 (typical)	14
Operating Voltage (V)	3.0	3.0	3.0	3.0	1.8
Throughput	4.8 kbps	4.8 kbps	19.2 kbps	>100 kbps	1 Mbps
Receiver Sensitivity (dBm)	N/A	-95 (typical)	-95 (typical)	Unknown	-84
Output Power (dBm)	0 (typical)	N/A	+1 (typical)	-8 to +4.5 (typical)	-10 to 2
External Component Count	2	2	~20	~50	Unknown

Table 3-2: Comparison of different RF chips

3.5 Software Design

We have written software for the Atmel ATMega103L for devices and gateways and wrote software that allows the proxy to communicate with the device.

3.5.1 Device

The device software that runs on the ATMega103L is written mostly in C with some assembly optimizations. It consists of RF transmit and receive routines, a packet processing routine, a location processing routine, and the security algorithms for RC5 and HMAC-MD5.

There are two RF routines, one that receives and transmits data to the proxy and another that receives data from the Cricket beacons. They are both driven by interrupts that occur at periodic intervals. This allows the device to communicate with its proxy while at the same time receiving messages from the Cricket beacons.

The RF routine to communicate with the proxy uses an interrupt that occurs 19,200 times per second. This frequency corresponds to the maximum rate at which the RF chip

communicates. Every time the interrupt occurs, the RF routine samples the output of the RF chip and stores this value. The routine then checks to see if it has received the preamble for a packet. If it has seen a preamble then it begins storing all the bits of the packet into a buffer to be processed later. As the routine is receiving the bits, it also does the conversion from 6 bits to 4 bits. In doing so the RF routine can also check for errors since if a 6-bit sequence does not correspond to a 4-bit sequence then there must be an error. If it receives an error, the RF routine then starts over and begins looking for another preamble. When the RF routine receives a packet, it stops receiving data and informs the packet processing routine that it has received a packet.

The packet processing routine for the proxy communication first verifies that the packet is authentic. If the packet is not authentic then it just throws it out and ignores it. If the sequence number of the packet is one less than the expected number then the routine resends the last packet sent. Otherwise, if the sequence number is the expected one then it decrypts the packet, processes the message, and sends a response.

The RF routine to listen to the Cricket beacons works in the same manner as the RF routine for proxy communication except that it operates at 4,800 times per second. When it receives the preamble of the packet it then starts a timer that is stopped once an ultrasound pulse is received. This allows the routine to determine the distance to the beacon. If the routine does not receive an ultrasound pulse before the whole packet has been received it throws out the packet because the beacon is either too far away or in a different room. Once a packet has been received, the location processing routine processes it.

The location processing routine for the Cricket beacon packets implements a very simple method to determine the closest beacon. Every time it receives a new location packet it compares the distance to that location with the distance to the location it already has stored. If the new distance is smaller it then stores that location as its location. When the proxy of a device asks for its current location it returns the location that was closest and then resets its current location to an invalid value so that it can start sampling again for its current location.

3.5.2 Gateway

The gateway uses the same RF routines that the device uses and differs in how it processes the received packets. When the gateway receives a packet it then transmits the packet over the serial port to a computer. If the gateway receives a packet from the serial port it then transmits the packet over RF. The code does not modify the packets but just directly passes them back and forth from RF to the serial port and vice-versa.

The gateway also contains a software component that runs on the computer that is implemented in Java. This component communicates over the serial port to send and receive the RF packets. It also takes the RF packets and sends them over the network as UDP/IP packets and vice-versa.

3.5.3 Proxy

The proxy is also written in Java and contains the communication routines that allow it to communicate with the device. It implements the simple retransmit protocol for device communication and also keeps track of which gateway it should send the packet to. The proxy is also where other code could be placed that does access control for the device and allows it to communicate securely with a user.

3.6 Evaluation

In this section we evaluate our devices in terms of what its memory and processing requirements are and also how well the RF communication works.

3.6.1 Memory Requirements

Component	Code Size (KB)	Data Size (bytes)
Device Functionality	2.0	191
RF code	1.1	153
HMAC-MD5	4.6	386
RC5	3.2	256
Miscellaneous	1.0	0
Total	11.9	986

Table 3-3: Code and data size on the Atmel processor

Table 3-3 gives a break down of the memory requirements for the different software components. The code size represents how much memory it takes up in the Flash memory and the data size represents how much memory is needed in RAM. The device functionality component includes the packet processing routine and the location processing routine. The RF code component includes the RF transmit and receive routines as well as the Cricket listener routines. The HMAC-MD5 and RC5 components consist of the code to implement them. Finally, the miscellaneous component is code that is common to all of the other components.

The device needs about 12 KB of code space and 1 KB of data space to run the code. The security algorithms, HMAC-MD5 and RC5, take up most of the code space. Both of these algorithms were optimized in assembly, which reduced their code size by more than half. More

optimizations could be done but this gives a general idea of how much memory is required. We feel that the code size we have attained can be easily incorporated into any device.

To further optimize the code size a better authentication algorithm could be used. We have used HMAC-MD5 for the authentication algorithm but it takes up 4.6 KB of code space. It may be better to investigate authentication algorithms that use an encryption routine instead of the hash function used in HMAC-MD5. This could significantly reduce code size as the RC5 code might be able to be reused to do both encryption and authentication. This could potentially reduce the code size to less than 8 KB.

3.6.2 Processing Requirements

Function	Time (milliseconds)	Clock Cycles
RC5 encrpyt/decrypt (x bytes)	(0.163 * x) + 0.552	(652 * x) + 2,208
HMAC-MD5 up to 56 bytes	11.48	45,920

Table 3-4: Performance of encryption and authentication code

The device's most demanding processing comes from the security algorithms. The time it takes for each of these algorithms to run is shown in Table 3-4. As can be seen the RC5 algorithm processing time varies linearly with the number of bytes being encrypted or decrypted. The HMAC-MD5 routine on the other hand takes a constant amount of time up to 56 bytes. This is because HMAC-MD5 is designed to work on blocks of data and so anything less than 56 bytes it treats as if it were 56 bytes long. Since the RF packets are never longer than 50 bytes we only care how long the HMAC-MD5 routine takes for packets less than or equal to 50 bytes.

To get an idea of how long the device takes to communicate we will evaluate how long it takes the device to receive a packet, process it, and then send a response. For this example we assume the device is receiving a packet that has 10 data bytes making the total packet size 27 bytes. The device communicates at 19.2 Kbps so it takes ((27 (packet size) + 4 (RF header)) * 12) / 19200 = 19.4 milliseconds to receive the packet. It then takes the device 11.5 ms to authenticate the packet and then 2.3 ms to decrypt it for a total of 33.2 ms. The device always sends a response back for every packet it receives. In this example we assume the device sends back another packet of the same size so the device must encrypt, authenticate, and then transmit the packet which will take another 33.2 ms. So, theoretically the device can handle 15 of these transactions per second. Without the security algorithms, the device could handle 25 transactions per second. We feel that for our simple devices being able to do 15 transactions per second should be sufficient.

Other optimizations could be done to improve the speed of the security algorithms. First, the size of the secret key used for encryption and authentication could be made smaller. This would speed up the RC5 algorithm. Second, the number of rounds that the RC5 algorithm uses could be reduced to improve the speed. Of course, both of these optimizations have an effect on the overall security of the system by potentially making it easier to break.

The best optimization would be to use an authentication algorithm that is not based on blocks of data. Since our packets are always smaller than 56 bytes, the HMAC-MD5 algorithm always costs us extra time because that is the minimum size of data it will work with.

3.6.3 RF Communication

Unfortunately, the RF communication was the weakest component of the system. On average, with the device and gateway three feet apart, the proxy needs to transmit a packet to the device three times before getting a response. This is due to errors in the RF communication channel. The RF chip we used, RFM TR3001, is too sensitive to noise this tends to introduce errors in the data being received. A better option would be to use an RF chip that uses a more robust modulation technique such as FSK that is more immune to interference.

The RF communication does limit the scalability of the system somewhat but this can be overcome. Since there is a limited amount of bandwidth, if one gateway needs to communicate frequently with many devices we may have a problem. If we assume all devices communicate with their proxies every second then theoretically we could support about 15 devices per gateway based on the performance times from the section 3.6.2. Since the RF does not perform well, in reality we could only support 4-5 devices. If this became a limiting factor in the scalability of the system, we could always make the RF signals weaker. That way the area a gateway covers would be smaller so we could have more gateways per area and thus more devices.

3.6.4 Power Consumption

While our board was not specifically designed for low power consumption, it is still important to know its power requirements. When the board has its RF transceiver in the receive mode it takes about 22 mA of current or 66 mW of power. At this rate in nominal conditions the battery will last 54 hours. When the board transmits it requires 29.5 mA of current or 88.5 mW of power. Most of the time the board is receiving data so the power level for transmitting is not reached very often. For devices that do not need to know their location, the Cricket listener can be removed to save power. The Cricket listener takes 10 mA of current or 30 mW of power so removing the listener reduces the boards power by almost half.

If we redesigned the board and optimized the code size, we could use the PIC16F877 for the processor and reduce power by 15 mW. Of course, there are other ways of reducing power. Since the RF chips consume the most power the communication protocol could be designed in such a way to allow these chips to be turned off for a period of time. In addition, the processor could sleep when it has nothing to do to conserver power.

4 Integration

The design of our system allows it to be easily integrated with other systems that provide resource discovery or that run scripts for automation. The proxy provides a convenient place for interfacing the devices to any system. For example, our architecture would allow us to incorporate the secure device communication into the Jini system. The proxy would be the service provider in the system and would be able to control the device securely.

Our system was also integrated with work by Matt Burnside who developed a server network [4] and work by Ali Tariq who developed an automation engine [23].

4.1 System

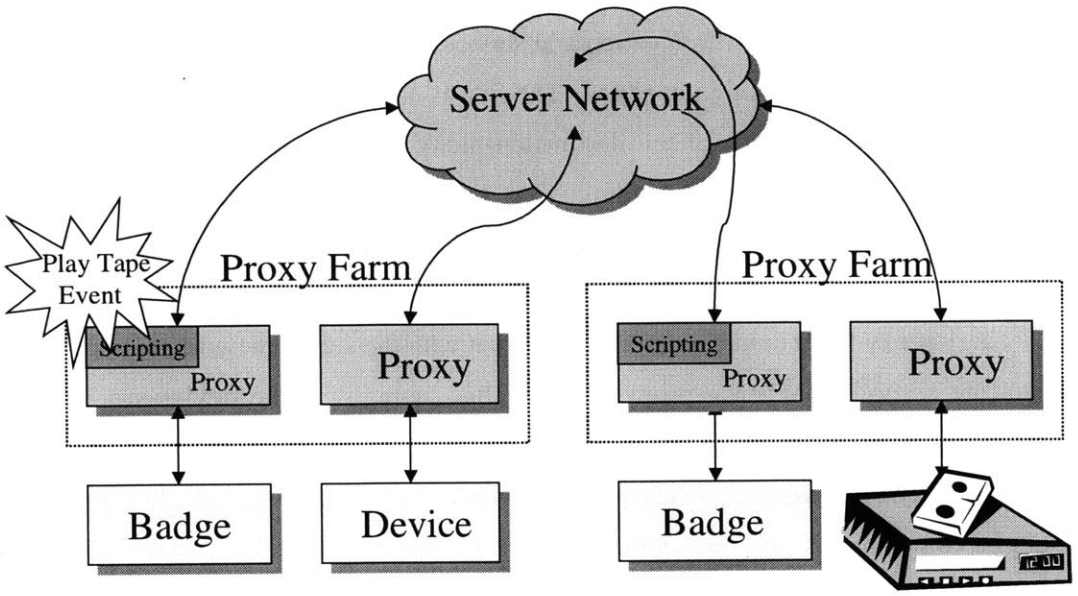

Figure 4-1: System overview: device communication, server network, and automation

The overall system is designed in a layered approach with our secure device communication being the bottom layer. On top of this layer is the server network that allows proxy-to-proxy communication and provides a directory of devices and their capabilities. The top layer is the automation engine that allows scripts to be run that control all of the devices.

The whole system is based on an event-messaging scheme. This scheme allows the devices and scripts to communicate in a dynamic nature where events can be routed to multiple recipients. For example, when a telephone rings it could send a "telephone call" event that would be routed by the server network to those proxies listening for the event. The proxies listening for

the event will have scripts that determine what type of action to take when they receive events. In this example a script that gets the "telephone call" event could then send a "turn down volume" event to all the speakers in the room. The speakers would be listening for this event and when they receive it, they would turn down their volume.

Figure 4-1 shows how all the layers of the system fit together. This figure also shows two types of specific devices, a VCR and a badge. The badge, worn by every user of the system, reports the user's current location to its proxy, which can then be used by the scripting engine. The scripting engine runs on every badge proxy and lets the user input his or her scripts and automate his or her environment. The server network provides the basic infrastructure allowing all the devices to communicate.

The server network consists of a distributed network of servers that help route events and a directory that keeps track of all the devices in the system. The directory allows the automation scripts to query for devices based or their name. A name is a list of attribute-value pairs, used by a device to describe itself, such as location and service type. The server network also provides two different methods of routing messages. The first is early-binding where the server network returns the IP address and port of the destination proxy so that communication is direct from proxy to proxy. The second is late-binding where the server network routes the event based on the name of a device. For example, in late-binding the server network would route an event with the name "location = room 4011" to all the devices in room 4011.

Every proxy has a component that registers it with the server network and one that lets it communicate securely. When a proxy comes online, it registers the device's name with the server network and finds the closest server that it will use for routing events and executing queries. A proxy also implements security protocols that allow it to authenticate itself to other proxies and communicate confidentially.

The scripting engine runs on every badge proxy so that each user has his or her own place for entering and executing scripts. The scripting engine presents a graphical user interface so that a user can send commands to devices or input simple scripts. The user can enter multiple scripts, each script has some condition that needs to be met and when it is, the script executes a response. The scripts also have a lifetime so the user can choose to have the script execute once or forever.

4.2 Example Applications

We developed an example application that allows music to follow a person around as he or she moves from room to room. In this application, we have a badge that the user wears. This badge is the same board we described in section 3.4 and has a corresponding proxy that

communicates with it securely. The proxy periodically asks the badge for its current location and reports this location to an automation script that is running on the proxy. When the automation script receives audio events, it queries the server network looking for the speakers that are in the same location as the user. The server network looks in its directory for speakers in that location and returns them to the script. The script then routes the audio events to a speaker in that location. Figure 4-2 gives an overview of how the different components in this application work.

One of the benefits of routing the audio through the proxy is that the person sending the audio does not actually know the location of the person receiving the audio. This allows the person receiving the audio to maintain some privacy by not having to give out their location for this system to work. The proxy could also route other types of information to the user's current location such as text messages, or video, while keeping the users actual location private. For other applications, the user could setup his or her badge's proxy to only give out his or her location to select people but keep it private from others.

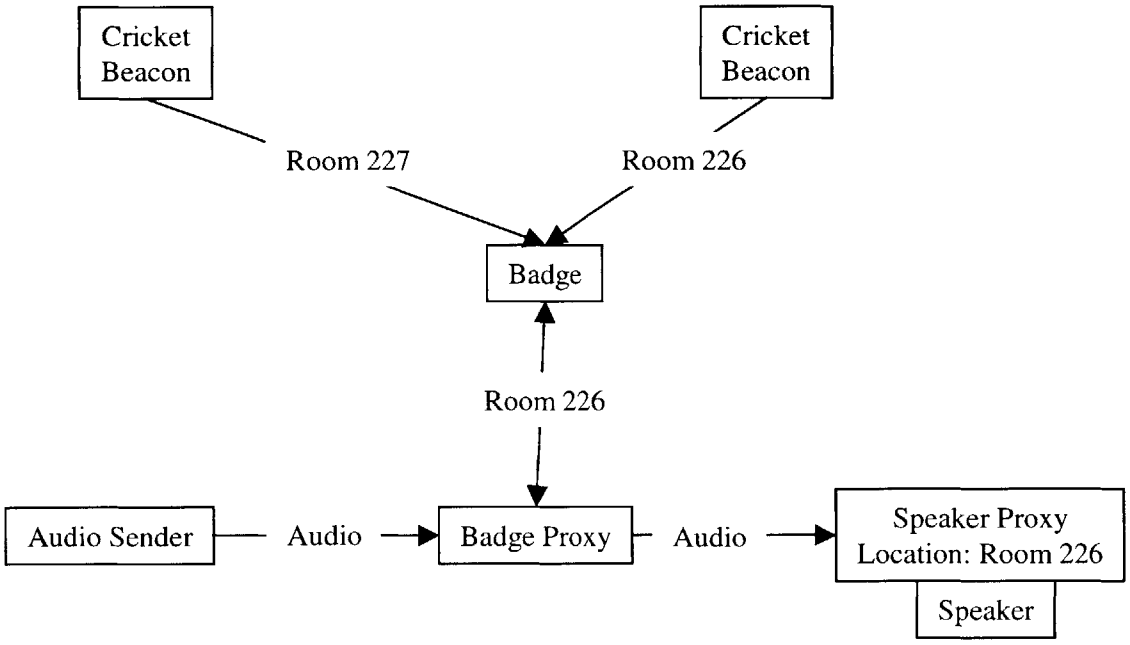

Figure 4-2: Audio example application

Our current system could also easily support other applications as well. For example many customizations could be made upon entering a new room such as turning on the lights, setting the thermostat, and opening the blinds. Another application would have the computer nearest the user automatically log him or her in and essentially customize the computer to his or her preferences. The system could also forward phone calls to the phone nearest the user. For

privacy, the system could be set up to only print your documents when you are located next to the printer.

5 Conclusion

This thesis has presented an architecture and implementation for secure device communication that works for simple devices. Our system accomplishes the following items:

- By using a proxy to handle access control, we keep the device's overhead for secure communication extremely small.
- The architecture has two different security models, indirect and direct, that allow flexibility in how the devices can be controlled.
- The proxy provides a convenient place for interfacing a device's services with a directory service and for running automation scripts to control the device.
- The devices communicate with gateways using RF. The gateways are distributed throughout a coverage area so the devices can be mobile.
- Through code optimizations, the devices require less than 12 KB of code and can execute the security algorithms in tens of milliseconds on 8-bit micro-controllers.
- We have kept the communication protocol simple and optimized the packet size by using a MAC to authenticate packets and to check for errors.

Bibliography

[1] J. Al-Muhtadi, M. Anand, M. Mickunas, and R. Campbell. Secure Smart Homes using Jini and UIUC SESAME. In *Computer Security Applications*. December 2000.

[2] John Anthes. OOK, ASK and FSK Modulation in the Presence of an Interfering Signal. http://www.rfm.com/corp/appdata/ook.pdf.

[3] P. Bahl and V. Padmanabhan. RADAR: An In-Building RF-based User Location and Tracking System. In *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[4] Matthew Burnside. Architecture and Implementation of a Secure Server Network for Project Oxygen. Master's thesis, Massachusetts Institute of Technology, Work in Progress.

[5] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), Feb 1981.

[6] Carl Ellison, Bill Frantz, Butler Lampson, Ron Rivest, Brian Thomas, and Tatu Ylonen. SPKI certificate theory. Internet Request for Comments RFC 2693, Septermber 1999.

[7] Pasi Eronen and Pekka Nikander. Decentralized Jini security. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2001)*, San Diego, California, February 2001.

[8] Jaap Haartsen, Mahamoud Naghshineh, Joh Inouye, Oalf J. Joeressen, and Warren Allen, Bluetooth: Vision, Goals, and Architecture. *Mobile Computing and Communications Review*, 2(4):38-45, October 1998.

[9] A. Harter, A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4(5):42-47, October 1997.

[10] A. Harter, A. Hopper, P. Steggles, A. Ward, and P. Webster. The Anatomy of a Context-Aware Application. In *Proc. ACM/IEEE MOBICOM*, Seattle, WA, August 1999.

[11] Jini™ Network Technology. http://www.sun.com/jini/.

[12] H. Krawczyk, M. Bellare, R. Canetti. HMAC: Keyed-Hashing for Message Authentication. Internet Request for Comments RFC 2104, February 1997.

[13] MIT Project Oxygen. http://oxygen.lcs.mit.edu/.

[14] NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.

[15] OpenSSL. The OpenSSL project. http://www.openssl.org, 2001.

[16] Jon Postel, editor. Internet Protocol. Internet Request for Comments RFC 791, September 1981.

[17] N. Priyantha. Providing Precise Indoor Location Information to Mobile Devices. Master's thesis, Massachusetts Institute of Technology, January 2001.

[18] N. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proc. ACM MOBICOM*. Boston, MA, August 2000.

[19] Ronald Rivest. The MD5 Message-Digest Algorithm. Internet Request for Comments RFC 1321, April 1992.

[20] Ronald Rivest. The RC5 Encryption Algorithm. *Dr. Dobbs Journal*. January 1995.

[21] Frank Stajano. The resurrecting duckling -- What next?. In *Proceedings of the 8th International Workshop on Security Protocols*, Lecture Notes in Computer Science. Springler-Verlag, Berlin, Germany, April 2000.

[22] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks. B. Christianson, B. Crispo, and M. Roe (Eds.). *Security Protocols, 7th International Workshop Proceedings*, Lecture Notes in Computer Science, 1999.

[23] Mohammad Ali Tariq. Architecture and Implementation of Automation and Scripting for Oxygen. Master's thesis, Massachusetts Institute of Technology, May 2001.

[24] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91-102, January 1992.

[25] M. Weiner. Performance Comparison of Public-key Cryptosystems. *RSA Laboratories' CryptoBytes*, 4(1), 1998.

[26] T.Wu, The Secure Remote Password Protocol, In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, San Diego, CA, March 1998, pp. 97-111.

[27] X-10.ORG – X-10 Technology and Resource Forum. http://www.x10.org.