RFID: Opening Doors

Alexander Benn and Molly Piels

8 May 2008
Advisor: Erik Cheever

Abstract

RFID access systems offer a wealth of features and advantages over traditional key-based entry, such as low per-user cost, easy administration, and powerful access control and configuration. This report describes the design and implementation of a low-cost RFID access system installed in Hicks Hall at Swarthmore College. The system consists of two major components: a server running PHP and MySQL and a set of PIC microcontroller-based door controllers.

Table of Contents

1.0 Introduction

Presently, Swarthmore College issues keys, which are manufactured and tracked by a central office, to students, faculty, and staff in order to control access to secure areas. There are a number of drawbacks to this system. It generally takes several days for the central office, Key Central, to process requests from academic departments or individuals for keys. Moreover, if a user loses their key, they face a hefty fine and the entire building may have to be re-keyed. To reduce the cost and hassle of dealing with a lost key, most keys only open one or two doors, a system which forces many users to carry a large number of keys. This is especially a problem in Hicks, where students need after-hours access to lab equipment that is often spread throughout the building. In large part because of these difficulties, Swarthmore is moving toward implementing a single, centralized card system to supplant the existing key-based one. Swarthmore students will eventually be able to use their regular student ID cards, which will be equipped with radio frequency identification (RFID) technology, to open all the doors they have permission to open. At present, however, there is very little space on campus that is accessible by means other than keys.

The College's decision to use RFID, as opposed to other competing technology, to control access to buildings was not a forgone conclusion. Several of the technologies that compete with RFID are used in other entry systems and are already in use in other applications at Swarthmore. Magnetic stripe cards have been in use at the College and at other schools for years, as have barcodes. The school's decision to go with RFID was based on the considerable drawbacks of both of these. Magnetic stripes experience significant wear and tear. Even if the stripe itself does not wear out, the printing on the other side of the card often does. This printing can often include security measures; in Swarthmore's case, it is a photo of the cardholder. Barcodes have the benefit of not requiring physical contact between card and reader, but they are read by lasers, and it is not practical to mount a laser on a door and keep it on continuously. RFID systems also avoid the problems caused by physical contact between card and reader, but they do so in a way that is far more practical for a permanent installation and continuous operation than a barcode scanner would be.

An interaction between a generic RFID system and one of its users (either a person or an item the system is designed to track) takes place between two components: the reader and the tag. Readers are mounted at stationary locations where identification is necessary, and tags are carried by users or mounted on the objects the system tracks. Each tag is encoded with a unique identification number. When a tag enters the proximity of a reader, the reader reads the information encoded on the tag and transmits it to a control unit. No physical contact between the tag and the reader is necessary: the devices exchange information via radio-frequency electromagnetic waves. The tags Swarthmore uses are passive, which means that they do not have on-board power. They use the energy contained in the wave emitted by the reader to power a small integrated circuit. The circuit modulates the carrier wave at two distinct frequencies and reflects it back to the reader. The reader then interprets the modulation as a series of ones and zeroes. The tag's lack of a power supply limits both the amount of data that it can hold and the distance at which it can be read. Active tags, or tags with their own power supply, are more common in

applications where the tag has to hold a lot of data or is placed at a significant distance from the reader.

Swarthmore does not yet have a fully integrated entry system, but the school has begun embedding passive tags in student IDs. All members of class years 2011 and younger already have these new student ID cards and Public Safety will provide a new card with an RFID tag in it to older students upon request and demonstration of need. There are also two disjoint systems in place that are administered by different groups. The Computer Science department runs a system that controls seven doors in the Science Center. Information and Technology Services (ITS) administers a system that controls seventeen doors in Beardsley and one in Trotter. These two systems were both very costly and would not necessarily be well-suited to the needs of the Engineering department. The ideal system for the department would:

- cost relatively little: commercial systems cost in the tens of thousands of dollars.
- maintain privacy: the department has no need to keep track of who enters a room when.
- be easy to implement: the system would need to use the technology already in place at Swarthmore.
- be relatively secure: it should be as hard to wrongfully gain access using the RFID system as it is to pick a lock.
- be modular: it ought to be easy and inexpensive to add more doors to the system.
- be easily administered by department personnel: system maintenance and data entry must be able to be performed by an individual with no specialized training.

This project is a RFID-based entry system for Hicks that meets these general criteria. The system currently controls access to the two interior doors to which students most often request keys (rooms 213 and 310), but it will be easy to expand to control any door in the building at a cost of around $500 per door. It allows administrators to set room access permissions by connecting to a web server running software developed for this purpose. Students will be able to use the RFID tags already embedded in student ID cards to open the doors they are allowed to open. The system works in parallel with already existing locks to ensure that other groups that need access to Hicks will not need to change their administrative procedures. When the College moves to a single centralized system, it will be able to use the hardware that has been installed in the building as a part of this project.

1.1 System architecture

All RFID entry systems consist of five main components: an RFID reader, a door controller, a door strike, a power supply, and a server. In commercial systems, a magnetic sensor that detects whether or not a door is open is also included. With that one exception, this system does not differ from commercial systems in overall architecture; the major differences lie in the function performed by each component. Figure 1 shows a block diagram of this system. When a user presents a card to the reader, the reader sends the ID number to the microcontroller and Ethernet controller, which reformat it and send it over Ethernet to the server. The server responds over the Ethernet connection, and the

microcontroller operates the door strike accordingly. The module on each door is powered by power over Ethernet (PoE) injected into the system by a switch, which is located in the basement of Hicks.

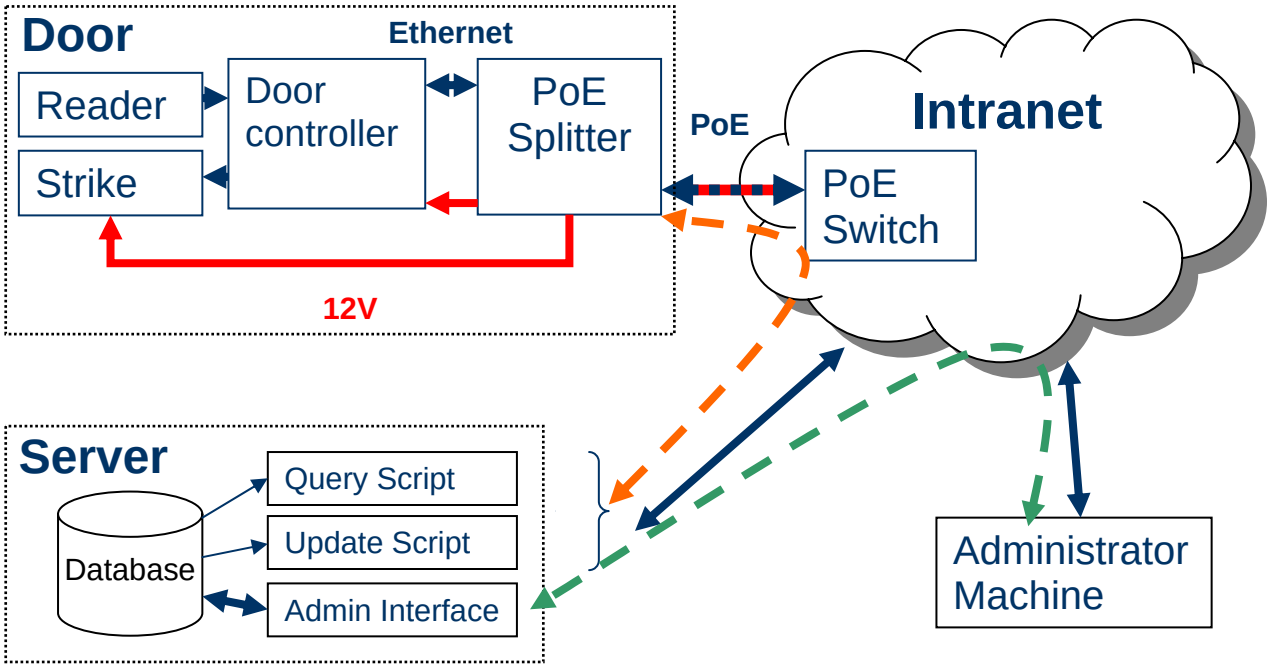

Figure 1. Block diagram of RFID entry system for Hicks.

In a commercial system, when a user presents a card to the reader, the reader sends the ID number to a controller. Commercially available controllers generally are capable of controlling more than one door; the controllers Swarthmore uses can usually control four each. The controller checks the ID number against a locally stored database of user IDs and permissions. It then operates the door accordingly and creates a log of the transaction. When administrators change door access permissions, the server updates these local databases. Administrators can also access the logs stored on the controllers from a workstation specifically dedicated to this purpose that sends requests to the controllers via the centralized server.

There are three main differences between this system and commercial ones. At the hardware level, this system relies on the wiring already present in a building to exchange information. At Swarthmore, this is a significant cost-cutting measure because of the preponderance of gray stone as a building material: it cost about $8000 to wire the Science Center for the Computer Science department system.[1] The second difference is that this system relies on a centralized server, rather than a local door controller, to respond to requests. To do this, commercial controllers must be about as sophisticated as personal computers. By decreasing the functionality required of the controller, this system can use a much less expensive processor (specifically, a microcontroller). This also cuts costs: a commercial four-door controller costs about $1,500 whereas this system can control four

[1] Siemens bid. The analogous value for the ITS system is not comparable because Siemens only replaced an existing system and the vast majority of the necessary wiring was already in place.

doors for around $350. The third difference is that this system's server software and administrative interface run on top of existing machines whereas commercial ones require their own dedicated server and workstation, the combined cost of which is around $4,000.

1.2 Component overviews

There are six key components in this system. A list of part numbers and market value can be found in Appendix A. Their function within the system as a whole is briefly described here and addressed in greater detail in subsequent chapters.

Reader

The reader reads the proximity tag and sends the ID number over two wires in clock-and-data format to the microcontroller.

Microcontroller

The microcontroller acts as a relay between the reader and the server. It is meant to use a serial EEPROM with a local hard copy of user codes and permissions in case the controller is unable to communicate with the server, but this is not yet fully implemented. The microcontroller interfaces with Ethernet via a small Ethernet controller chip on the same printed circuit board.

Ethernet Controller

The Ethernet controller sends the request to open the door from the microcontroller to the server and holds the server's response until the microcontroller instructs it to do otherwise.

Strike

The strike mechanically allows the door to open (or does not). It is installed in the door frame, and thus does not interfere with the lock already in place. It is fail-secure, which means that if it loses power, it will remain locked.

Power Supply

The door controller communicates with the server via Ethernet. The same Ethernet cable can be used to power the system using PoE, so only one connection between the door wiring and the outside world is necessary. The system uses a power splitter to extract power from an Ethernet cable and supply it to the strike, controller, and reader.

Server

The server manages the database of names and allowed IDs and handles requests from the door controllers. The server also includes a web-based management interface accessible to Engineering department staff. The web interface consists of a set of PHP server scripts, which interface with a local SQL database such as MySQL.

2.0 PCB design and construction

The printed circuit boards for this project are based on Microchip's PicDemNet board. Full Multisim and Ultiboard schematics of the board actually printed as well as an improved second version of the board can be found in Appendix B. Unnecessary features, such as the LCD, serial port, and RS232 jack have been omitted from the PicDemNet demo board design and power-related and door control circuitry has been added. The PIC cannot be programmed while it is on the board. The PIC has to be removed from its socket, placed on a board that does have a jack for the in-circuit debugger, programmed, and replaced in order to update the code. Given that the PCB is to be mounted securely in a box attached to the door and wired to the strike and the reader through a hole in the wall, it is much easier to remove the chip than to remove the entire board.

The board is powered by Power over Ethernet (PoE). A D-Link splitter located inside the box separates the Ethernet signal from the power, both of which are then connected (separately) to the board. The power comes in to the board at 12 VDC and immediately goes through a diode bridge and a linear voltage regulator. This may be unnecessary because the PCB is not powered in the exact same way as the PicDemNet board. The PicDemNet board is usually powered by an AC to DC converter that plugs in to the AC power in a building. Since these devices are imperfect, the demo board's power supply usually fluctuates with the AC signal. The diode bridge and linear voltage regulator in the demo board's design are meant to rectify this. PoE, on the other hand, is a DC supply more akin to a battery, and the D-Link module performs some voltage regulation function as well. At the time the board was designed, the quality of the power supply was unknown, so the features were retained, but a second-stage board would test the linearity of the power supplied by the D-Link to see if the diode bridge and 12 V linear regulator are redundant. A large (220 μF) capacitor between this supply and ground prevents fluctuations in the power supply from affecting circuit operation. Both the strike and the reader operate on the 12 VDC. The PIC and Ethernet controller run on 5 VDC. A linear voltage regulator is used to step down the voltage to that level. The linear voltage regulators both generate significant amounts of heat; care should be taken to avoid contact between the regulators and wires coming in to the box.

The PIC operates the strike by toggling the voltage on the gate of an NMOS transistor (see Figure 2). The transistor is between one of the door strike leads (the two are interchangeable) and ground. The other lead is at a constant 12 V. When the door ought to remain closed, the PIC asserts zero voltage on the gate of the transistor, effectively preventing current from flowing through the strike, which floats at a constant 12 V. When the PIC asserts 5 V on the gate, it allows some current to flow through the circuit. Voltage drops across the strike, which lowers the gate-to-source voltage on the transistor enough to allow current to flow freely through the strike. This releases the strike from its locked position and allows the door to open.

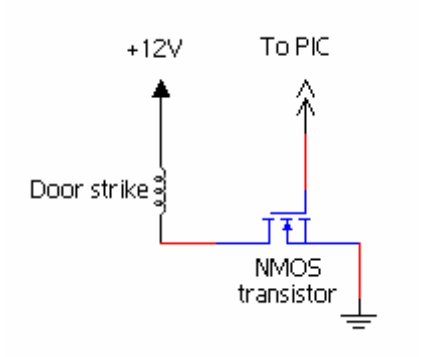

Figure 2. Circuit to operate strike.

The board connects to the door strike and reader through an 8-hole socket shown in Figure 3. In the original design (the one currently installed), an important connection between the Ethernet controller and its power supply was omitted. To rectify this, extra wires were soldered from the controller to pins five and eight and then to 5V and ground. A 100 nF capacitor should be inserted connecting these two pins. In the second generation design provided in Appendix B, pin five should be attached to the green LED line from the reader (orange) and pin eight should be attached to the beeper line from the reader (yellow). At present, these two connections are made by two separate female-to-female connectors attached to wires soldered directly to the board. The shield ground wire (black) from the reader is attached to the ground pin on the power jack or to the ground pin on the enclosure. The remaining three wires (violet, blue, and brown) should be wrapped separately with electrical tape.

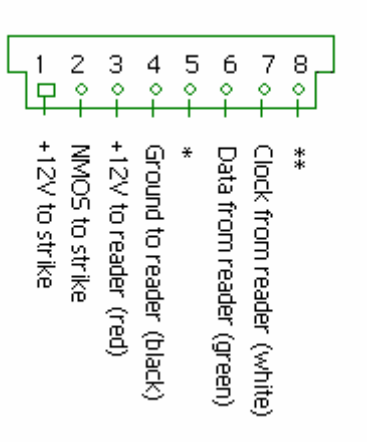

Figure 3. Socket that connects PCB to door strike and reader.

This version of the PCB suffers from connectivity issues such that (approximately) only every third attempt to transmit messages from the board succeeds. The microcontrollers currently in place in the boards have been programmed with this in mind and allow for a much greater number of retries (twenty) than is optimal. As a result, the amount of time between a card swipe and a server response is sometimes noticeable. It is important to emphasize that this issue is unique to this printing of the PCBs; it takes about 0.06 seconds for the demo board to complete a transaction with the server.

3.0 Programming the Microcontroller

The microcontroller is the heart of the door circuitry. It can be programmed on Microchip's PicDemNet board using their in-circuit debugger (ICD) 2 and then placed in the appropriate socket on the PCB. Programming a new microcontroller in order to add another door to the system can be done by changing the source code in a few key places. Altering the code to add functionality is substantially more complicated.

3.1 The basics: programming new controllers and working in MPLAB IDE

The code for this project is written to be compiled by Microchip's C18 compiler and MPLINK linker, both of which are available for free online. C18 follows most rules of standard ANSI C syntax. Microchip's TCP/IP stack is modular: the code for lower level functions is in a number of files that can be included in the project as needed. The purpose of this method of organization is to reduce overall program memory size. The code is meant to be compiled from within Microchip's IDE, MPLAB, which is well-suited to projects with multiple header and source files. MPLAB's primary components are its main window and two child windows: the workspace window and the output window (see Figure 4).

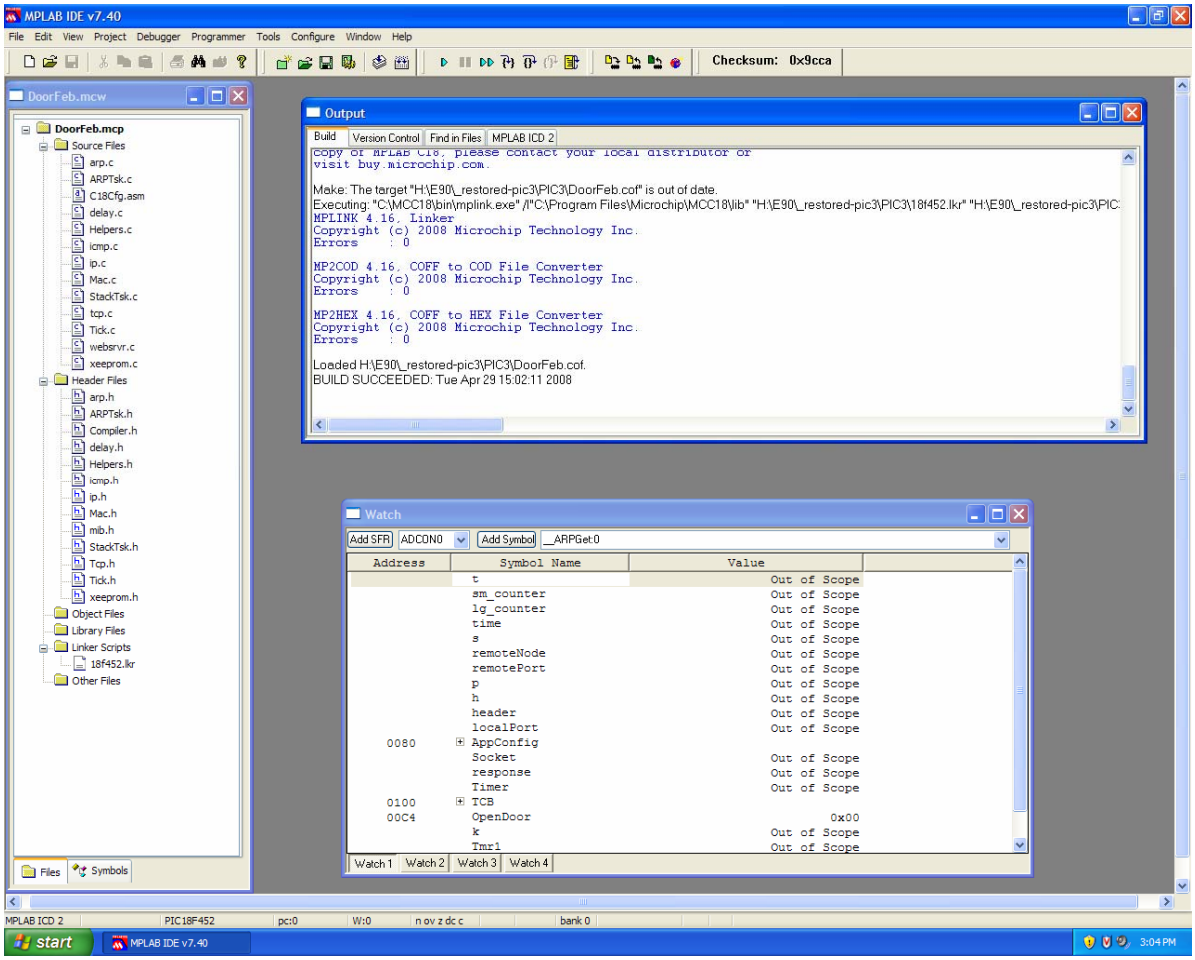

Figure 4. The MPLAB IDE main window, with workspace, output, and watch child windows shown.

The workspace window shows the files to be included in the project. The compiler will produce object (*.o) files for each of these, regardless of any relationships that exist between them. The linker resolves any cross-references between files and produces *.coff and *.hex files to be downloaded to the microcontroller. It will produce errors or warnings if a file is included twice or if an object is referred to but not defined.

Files can be added to the project either by right clicking on the project (*.mcp) folder in the workspace window and selecting "add file to project" in the workspace window or going to Project>Add Files in the main window. They can be removed in an analogous fashion. Double-clicking on a file opens it in a child window. File names are not case-sensitive.

The output window has four tabs: the "Build" tab shows the status of the compiler and linker; the "Version Control" tab is meant to be used with version control software; the "Find in Files" tab shows the result of the last project-wide search for a term; and the "MPLAB ICD 2" (or other debugger) tab shows the status of the debugger being used.

Both a project and a workspace file have been created as a part of this project; opening the project file will automatically load the workspace into MPLAB IDE. However, these files are not always back-compatible, so a new project may have to be made in order to program a door microcontroller. The following files should be included in the project:

```
C18cfg.asm
18f4520.lkr
arp.c
arp.h
ARPTsk.c
ARPTsk.h
delay.c
delay.h
Helpers.c
Helpers.h
icmp.c
icmp.h
ip.c
ip.h
Mac.c
Mac.h
StackTsk.c
StackTsk.h
tcp.c
Tcp.h
Tick.c
Tick.h
websrvr.c
xeeprom.c
xeeprom.h
```

Optional files that need to be included if (and only if) the code is compiled in debug mode are:

```
xlcd.c
xlcd.h
```

The header file `StackTsk.h`, in addition to serving as a header file for `StackTsk.c`, is used to define the mode of operation of the code; all modules that can be enabled or disabled, most notably debug mode, are enabled there. The main application code can be found in `websrvr.c`.

To compile correctly, a number of settings must be changed from the default. First, the project needs to be assigned the correct microcontroller. This project is designed to run on either the PIC 18F452 or 18F4520, though with some modification of `C18cfg.asm`, other 40-pin PIC 18s could also be used. To change microcontrollers, go to Configure>Select Device and select the appropriate microcontroller in the dialog box that pops up (see Figure 5).

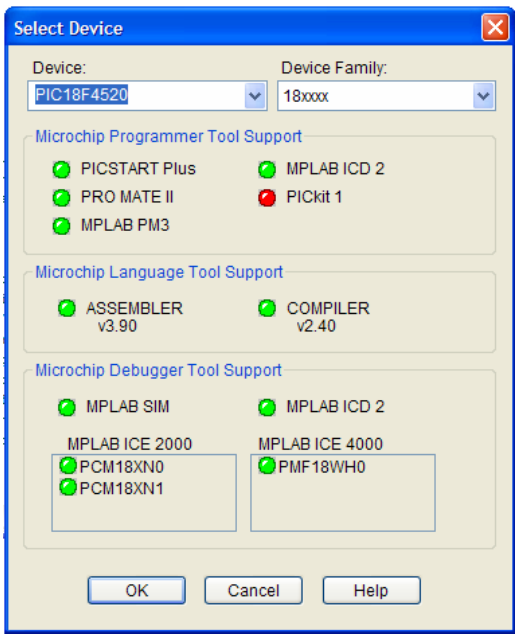

Figure 5. Select devise dialogue box.

After changing the device, the memory settings for the compiler have to be changed. Before compiling, in MPLAB IDE, go to Project>Build Options>Project, select the MPLAB C18 tab, category "General," and change the default storage class to "Overlay." This is the equivalent of using the command-line compiler options:

-sco -mL -Ou- -Ot- -Ob- -Op- -Or- -Od- -Opa-.

The code should also be compiled using the large code and large data models, which can be changed in the same tab as the default storage class, under the category "Memory Model." If the code is compiled using the wrong memory model or default storage class, the linker will be unable to resolve an arbitrary object; it will produce one of the two following error messages:

```
Error - section '.*.o' can not fit the section. Section '.*.o'
length=0x000001bd

Error - could not find definition of symbol 'X' in *.o"
```

where '*' is a source file and X is the name of a function or a variable. The compiler generates four warnings about suspicious pointer conversions in `websrvr.c`, but these can be ignored.

Once the code has been compiled, it can be downloaded to the PIC. To do this, select Programmer>MPLAB ICD 2 to program or Debugger>MPLAB ICD 2 to debug. An error message saying that it is no longer permissible to define both a debugger and a programmer simultaneously may pop up; click "OK." Then click on either Programmer>Program or Debugger>Program. If programming, the PIC will be ready; in debug mode, to start the program click on the blue triangle in the upper right-hand corner of the IDE main window.

If adding a new door to the system, the following lines should be changed before compiling and programming:

- At line 144 in `websrvr.c`, change the line:

```
ROM BYTE q2[] = \
     "&doorname=testdoor HTTP/1.1\r\n";
```

  to reflect the door name specified by the server.
- At line 278 in `StackTsk.h`, modify the IP address of the microcontroller (`MY_DEFAULT_IP_ADDR`).
- At line 293 in `StackTsk.h`, modify the MAC address of the microcontroller (`MY_DEFAULT_MAC`).

Other than obtaining a static IP address from ITS (see p. 14), this is the only work that needs to be done on the microcontroller in order to add a door to the system.

If the server's IP or MAC addresses change, the fields `SERVER_IP_ADDR` and `SERVER_MAC` can be changed in `StackTsk.h` beginning at line 343.

If changing the function of the code, MPLAB IDE offers a number of other tools that can be very useful. The watch and memory gauge windows, which can be opened by clicking View>Watch or View>Memory Gauge, are both helpful. Given the number of files associated with this project, the "Find in Files" command is also of use; it can be found under Edit or by pressing CTRL+SHIFT+F. A description of main program functions follows.

3.2 Main application

The main application coordinates all operations that the door controller performs (see Figure 6). It uses a single control byte, `OpenDoor`, to identify and prioritize tasks. In its idle state, the controller delays, responds to network addressing requests, and updates the clock. Its top priority is to respond when a card is swiped. Both a card swipe and an overflow on the on-board timer will generate an interrupt, and the program will immediately jump into the interrupt service routine (there is only one ISR). The routine prioritizes the card interrupt over the timer. The card swipe interrupt changes the value of the control byte so that the main routine will send the ID number to the server. The send function, when called, takes top priority within main. If it works (i.e. if the server returns an unambiguous response), the main loop opens the door, waits five seconds (to allow the user to open it), resets all flags, and then returns to its normal routine. If the send function does not work, it changes the control byte to tell main to check the serial EEPROM. The main while loop then checks the EEPROM, which either does or does not set the door open flag. Once daily, main calls `Download2EEPROM()`, which synchronizes the door controller clock with the server clock and downloads the list of ID numbers and access codes from it.

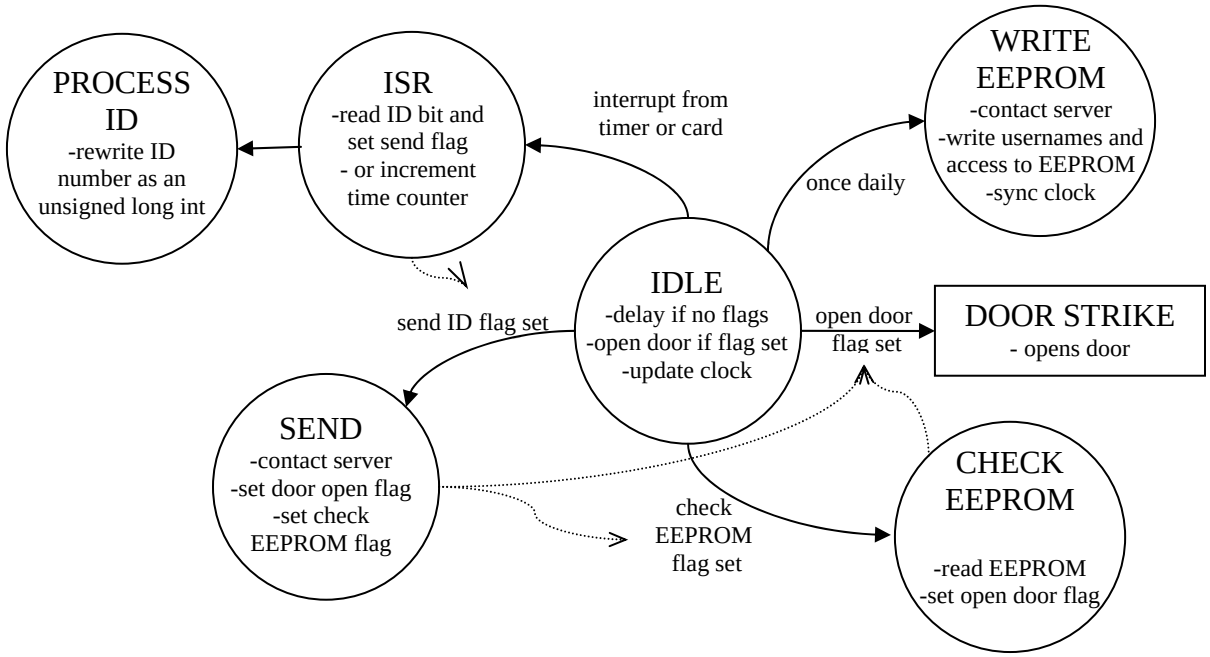

Figure 6. Main function map.

3.3 Clock and timing out

The PIC uses internal device clock generated interrupts on Timer 1 to keep the time. When the Timer 1 counter overflows, the ISR increments the program tick counter, which is stored in `tick.c`. When other stack modules call the function `TickGet()`, they read this value. The clock is used for two main purposes: timing operations and determining, using the contents of the serial EEPROM, whether or not a user should be allowed access to the room at a particular time.

All attempts to send data to or read data from the server can time out, as can all other while loops. At the beginning of these segments of code, the tick count is recorded and stored in the value `Timer`. All lower-level stack modules (those provided by Microchip) have functions that test whether or not a particular task is ready to be performed. While that task is not ready, the program checks to see if the current tick count exceeds the saved tick count by a set amount. These sections of code look like:

```
if(TickGet()-Timer>1.5*TICK_SECOND)
      {
      /* Give up */
      }
```

A `TICK_SECOND` is slightly less than two seconds long, so this function would time out after about three seconds. This is an approximate value, but sufficient for this

application. For tasks that involve contacting the server, the number of retries is also counted. As discussed above, this is limited to twenty because of mechanical issues. The value can be easily changed by modifying the line

```
#define NUM_RETRIES 20
```

at the top of `websrvr.c`.

The time kept by the main application is far less approximate because the constant `TICK_SECOND` is multiplied by a floating point number that has been experimentally determined to be accurate within a few seconds in 24 hours. Clock skew may increase as the device ages, and when a user swipes their card, the clock pauses briefly. To ensure that the clock is always within a few seconds of the actual time, the door controller synchronizes its clock with the server's clock at 3 am every 24 hours when it updates its list of ID numbers and permissions.

3.4 Getting an ID number from the reader

Swarthmore uses HID brand readers and RFID cards. HID readers can be configured to output data in clock and data or Wiegand format; the cards can also encoded in either clock and data or Wiegand format, but a clock and data reader can read a Wiegand card. Swarthmore uses Wiegand-encoded cards, and the door controllers output the card number in clock and data format. The readers are powered by 12 VDC and have a read range of about three inches, though the range decreases as a card ages.

3.4.1 Security risks inherent in the use of RFID technology

With the US Federal Government's recent move to add embedded RFID tags to passports, the amount of research into methods of duplicating another person's RFID number has grown. It is now well-established that with commercially available equipment costing around $200, it is possible to read a card as it is being carried by another person and copy its information well enough to fool a reader. Most institutions that have chosen to implement RFID systems have done so only after weighing the increased risk of key duplication against the many benefits of those systems. There are some commercially available card sleeves and wallets that prevent a card from being read for high-risk applications, but it is not likely that they will ever be used at Swarthmore.

3.5 The door controller and the network

The door controller works primarily as a relay between the card reader and a central server. It communicates with the server using established Ethernet protocols over the building's network.

3.5.1 Addressing

Every node (machine) on a network contains a Media Access Control (MAC) address and an Internet Protocol (IP) address. The MAC address is unique to the device, though on computers it can be changed easily. On Swarthmore's network, the IP address is assigned

from a pool of addresses through a process called Dynamic Host Configuration (DHCP). Thus the MAC address of a particular machine will not change unless an administrator changes it, but its IP address could change every time it is turned on. Because this system uses the IP address of the requesting node as a security measure, it is important to ensure that the IP address is static (i.e. not assigned through DHCP gleaning but rather hard-coded on to the device). To avoid collisions, the network needs to be able to identify the door controllers through their MAC addresses and not attempt to assign them new IP addresses; it also needs to remove door controllers' static IP addresses from the DHCP pool and not assign them to other machines. Before installing a door, it is important to consult ITS to obtain a static IP address and remove the device from the DHCP gleaning process. The IP and MAC addresses of the door controller and server are defined in `StackTsk.h`. The doors used in this project have the MAC and IP addresses shown below:

Table 1: Doors and their associated addresses

Door	MAC	IP
Testdoor	00:04:a3:00:06:39	130.58.84.138
Hicks310	00:04:a3:00:06:10	130.58.84.139
Hicks213	00:04:a3:00:06:11	130.58.84.140

Both addresses are loaded into program memory in the function `InitAppConfig()` in `websrvr.c`. The server, like the door controller, should have a constant IP address; if it changes, the door controller will no longer be able to form a connection with it and will have to be reprogrammed.

Every networked machine has a number of ports associated with it. Port 80 is reserved for incoming HTTP requests, and is thus it is the server port to which the door controller requests to connect. Ports above 1024 have no pre-assigned meaning; the door controller initially attempts to connect to the server from port 1026, but will increment the port number with each retry.

3.5.2 Anatomy of an Ethernet Packet

Ethernet packets consist of at least 64 bytes of data arranged in nested frames. These proceed from the lowest level (Ethernet) to the highest (HTTP) (see Figure 7). With the exception of HTTP, each frame consists of a header, a data field that contains higher level protocols, and a checksum.

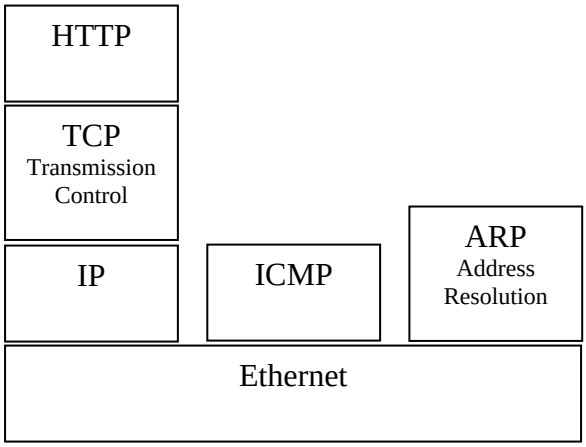

Figure 7. TCP/IP protocol stack (only the protocols used by the RFID entry system are shown).

The door microcontroller uses Ethernet, IP, and TCP to connect to and disconnect from the server. It uses HTTP to transmit its query and receive the server's response. If it is the first time the server and microcontroller are communicating, ARP is used to help the server locate the microcontroller. ICMP functionality is included but not strictly necessary. However, it may be desirable to allow the system administrator to determine whether or not the door microcontroller is connected to the network by pinging it, and pings are ICMP messages.

3.6 Sending the ID number to the server: the TCP state machine

The main TCP state machine consists of four stages: connect, send, receive, and disconnect. Connecting and disconnecting are done using stack layers up through TCP and sending and receiving include HTTP in addition. If any stage takes too long, the door controller stops trying to get information from the server and reads the serial EEPROM instead.

3.6.1 Connecting to the server

Before the door controller transmits the ID number to the server, it establishes a connection using a TCP handshake. The TCP handshake consists of three steps (see Figure 8). First, the door controller sends a synchronize packet (SYN) requesting that the server open a connection with it and with a start sequence number. Then, the server responds with an acknowledgement (ACK) and its own start sequence number. Finally, the door controller acknowledges that it has received the server's SYN+ACK packet and begins transmitting data.

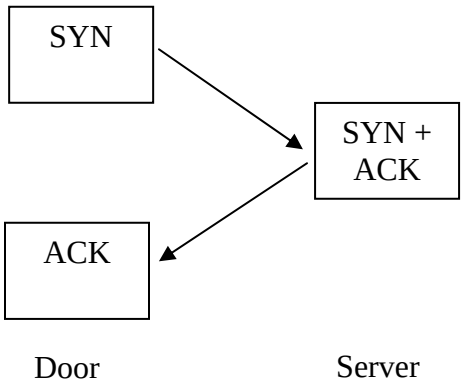

Figure 8. The TCP handshake.

Upon receiving the initial SYN packet, the server attempts to locate the door controller using Address Resolution Protocol (ARP) requests. In order for the handshake to proceed, the server must resolve the IP address of the door controller. The line:

```
StackTask();
```

in the main TCP state machine in `websrvr.c` handles incoming ARP packets and completes the TCP handshake.

The door controller initially attempts to connect to the server from its own port 1026. If this connection is denied or times out (after one second), it moves to port 1027, and so on, until a connection is established or the attempt to contact the server times out.

3.6.2 Sending the ID number to the server

The door controller sends the ID number to the server in the form of an HTTP GET request. The Microchip stack does not implement HTTP client functionality; The TCP state machine puts both the header and the query are into the transmit buffer, then flushes that buffer. A HTTP header consists of an action (GET), protocol specification (HTTP 1.1), host (the server), and a series of options. The HTTP header is as follows:

```
GET /php-rfid/query.php?rfid=xxxxxx&doorname=testdoor HTTP/1.1\r\n
Host: 130.58.84.55\r\n
User-Agent: Door microcontroller\r\n
Accept: text/html,text/plain\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n";
Keep-Alive: 300\r\n
Connection: keep-alive\r\n
http:\\130.58.84.138\r\n
\r\n\r\n
```

This string is stored in ROM and defined in `websrvr.c`. To change door or server IP addresses, door name, HTTP version, or agent name, this text must be modified. The remaining fields specify the kinds of data the door controller is configured to receive. The ID number is converted from an integer to a string in the main TCP state machine.

3.6.3 Receiving and processing the server's response

After sending the ID number, the door controller waits until the server's response appears in the Ethernet controller's receive buffer. The response consists of an HTTP header followed by a single line of text, either "true," "false," or "ERR:"

```
HTTP/1.1 200 OK
Date: Fri, 25 Apr 2008 02:10:06 GMT
Server: Apache/2.2.3 (Debian) PHP/4.4.4-8+etch4
X-Powered-By: PHP/4.4.4-8+etch4
Content-Length: 4
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
true
```

When the controller receives the response, it discards the HTTP header and scans the remaining five bytes in the receive buffer for all three of these possible messages. It then sets the open door flag accordingly. If it receives an error, it re-transmits the request once. If the server returns an error again, it returns to main and reads the serial EEPROM to see whether or not to open the door.

3.7 Checking and writing EEPROM contents

The serial EEPROM attached to the door controller is used as a backup system in case the server does not respond to the controller. Each user takes up 92 bytes of memory; 8 for their ID number and 84 that correspond to each two hour segment of the day for an entire week (so that each bit corresponds to a 15-minute segment). The `CheckEEPROM` function scans the serial memory for the appropriate ID number, then scrolls to the appropriate bit, reads it, and returns the value to main. At present, the download routine is not functional; the problem has to do with space limitations on the Ethernet controller's receive buffer.

3.8 Board configuration: power outages and startup routine

The board and web applications are initialized in two functions that are called on startup, `InitializeBoard()`, and `InitAppConfig()`. The board initialization sets the status of all used input and output pins and enables the appropriate interrupts. The application configuration loads the door controller and server MAC and IP addresses into program memory. As part of the startup routine, the door controller transmits a query about the ID number 123456 to the server. Given Swarthmore's facility code, it is not possible for there to be a user with this number in the database, but for security `OpenDoor` is set to keep the door closed immediately after the number is sent.

3.9 Debug techniques

There are a number of techniques that are useful for debugging the code, should changes become necessary. If the suspected problem has to do with server-controller communication, a packet sniffer can be used. If the suspected problem is on the controller side, `StackTsk.h` defines a debug mode that can be used to read the Ethernet controller buffers into program memory, and store all data coming from an ID card, among other things.

3.9.1 Packet sniffers

The best of these is Wireshark. It caches all incoming packets, can filter them, displays them in real time, and has a nice graphical interface. To start Wireshark (after installing it), type "sudo wireshark" at the command line in Linux. Go to Configure>Options, click the box "Update packet list in real time," then "start." A window displaying the number of packets sniffed, organized by type, will pop up. Killing this window will stop the capture. The filter:

```
(ip.addr==130.58.84.138)||(eth.addr==00:19:B9:4B:FC:8C)||
        (eth.addr==00:04:A3:00:06:39)
```

where the IP address is the address of the controller and the Ethernet addresses are its MAC address and the MAC address of the server will catch all packets coming from or going to the door controller. A correct exchange between server and door is shown in Figure 9. The complete transcript of a similar exchange can be found in Appendix D.

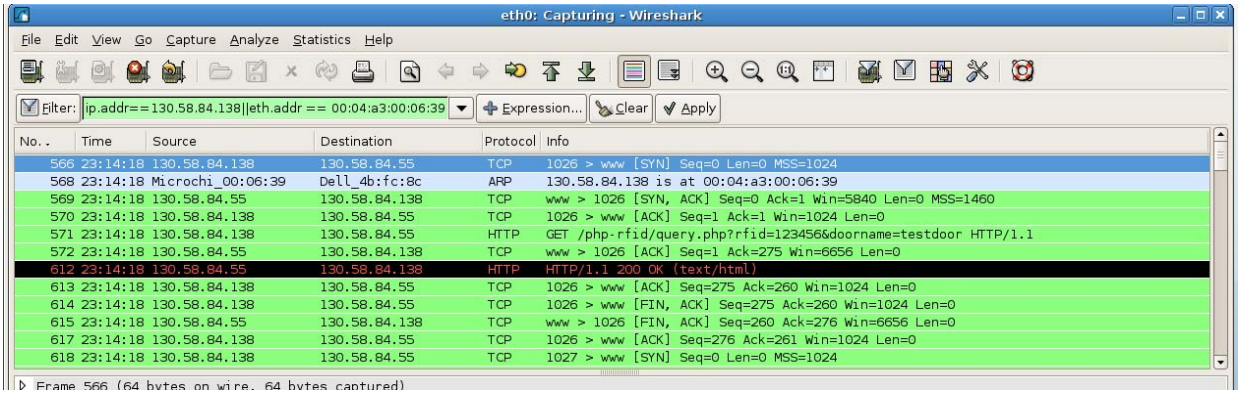

Figure 9. Correct conversation between door controller and server as captured by Wireshark.

3.9.2 Debug mode

Uncommenting the line

```
#define STACK_DEBUG_MODE
```

in `StackTsk.h` will enable the LCD.

3.9.3 The PicDemNet board

If the PicDemNet board is being used to debug, the files `XLCD.h` and `XLCD.c` can be included in the project and used to print to the LCD at any point in the code; this can be useful as the PIC18F452 only supports one breakpoint and the PIC18F4520 only supports two.

4.0 The Server

The other main component of the system is the server software. The server processes incoming door requests, and also provides the web-based administration tools.

4.1 Installation

This software package requires a server machine which meets the following system requirements:

- Pentium III, 500 MHz or faster
- an installed and updated Linux distribution, such as Debian or Red Hat
- Apache 1.3 or 2.1
- PHP 4.4 or later (tested on 4.4.4-8)
- MySQL 5.0 (tested on 5.0.32)
- Web browser, such as Firefox

To install, copy the script files to a directory that Apache can see, such as /var/www/php-rfid/. Currently, we do not have an installer script, but this will be included with the final version of the software. This installer script will configure the MySQL database and tables and set up a default Administration Machine and Administration User. It is also important to configure the global variables in common.php, such as MySQL username and password.

4.2 Administration: An Overview

Once an operational installation of the software package has been configured, the administrator can log in to the web administration interface. Until you an Administration Machine has been configured, the administrator will need to log in from the server itself (by typing http://localhost/php-rfid [for example] into Firefox). The default username is user and the default password is pass. Both of these must be changed to increase the security of the administration interface.

Once the login process is complete, a list of available panels appears at the left. The sections below discussing Administration Users and Administration Machines will outline both how to both change the administration password and how to add an additional Administration Machine.

4.3 Elements of the Administration Interface

There are six key types of objects in the administration package. They are Users,

Groups, Doors, Access Configurations, Administration Users and Administration Machines. Their behavior and mechanism of configuration is laid out in the sections below.

4.3.1 Users

A *User* describes an entity for which access may be specified for a Door. A User has a specific set of associated fields:

- ID (unique number)
- Last name
- First name
- List of doors this User may enter
- Facility code
- Card code
- Email address

One or more users may be created using the Add Users panel. Figure 10 shows this panel; the Add Users button is at the bottom of the page.

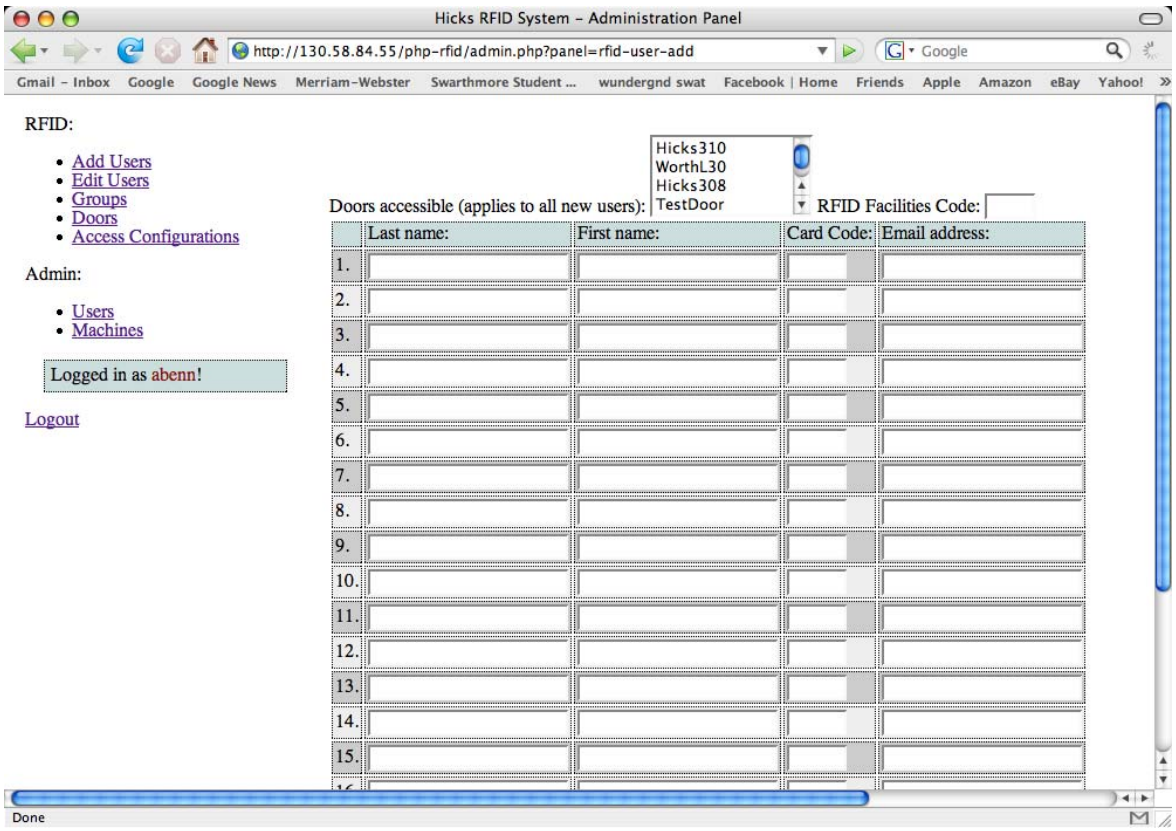

Figure 10: The Add Users Panel.

When creating a user, any field may be omitted *except* the Card code. If a Card code is not specified, it will default to a value of -1 and the User will not be able to enter any doors. The Facility code depends on the set of RFID cards distributed to students, and the Card code is unique to each card. See Figure 11 for an example of the card code on a Swarthmore ID card. The ID is given to the User by the system and cannot be changed later.

Figure 11: A Swarthmore ID card. Card code is boxed.

The RFID Users panel displays a list of users currently entered into the system and provides mechanisms to modify and delete users. This panel can be selected by clicking "Edit Users" on the left. Users are alphabetized by last name; to search for a name, type it in the Search box at the top. If there are more than 30 users in the system, the list may run to multiple pages, which can be accessed by clicking the numbers at the bottom of the current page. Note that this list is cached by the PHP script to speed lookup time; if it seems that the listing is outdated, navigate to another panel and back to get the freshest listing.

Certain operations in the Edit Users panel can be applied to multiple users using the checkboxes along the left side of the user listing table. Select one or more checkboxes and click one of the buttons at the bottom of the page to delete or modify a set of users, or to add these users to a Group.

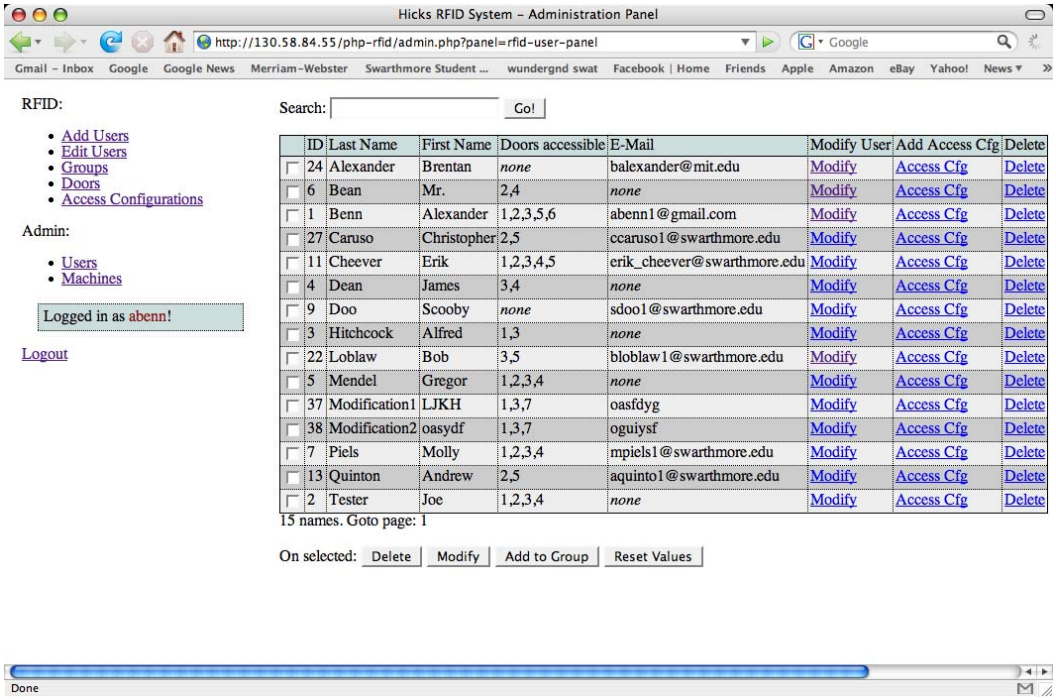

Figure 12: RFID User Panel.

Note that the list of doors each User may access is stored as a comma-separated list of door IDs; when modifying an individual user, it is important to maintain this format to guarantee functionality of the scripts.

4.3.2 Groups

The RFID system includes a powerful mechanism for controlling access on a set of Users. This mechanism is called a *Group.* A Group entry contains:

- Group ID
- Description
- List of Users

A User can be a member of zero, one, or more groups. An Access Configuration, described shortly, can be applied to an entire Group. To create a new Group, it is recommended that the administrator use the RFID User panel or a User Search. If it is for some reason more convenient to manually add a new Group or modify an existing Group, it is important to conform to the following rules for entering or modifying the list of UIDs. The list must start with a space, end with a space, and contain exactly one space between individual UIDs. This format guarantees that internal algorithms within the scripts can correctly parse these lists. A possible extension of the project would be to provide an interface that enforces these rules.

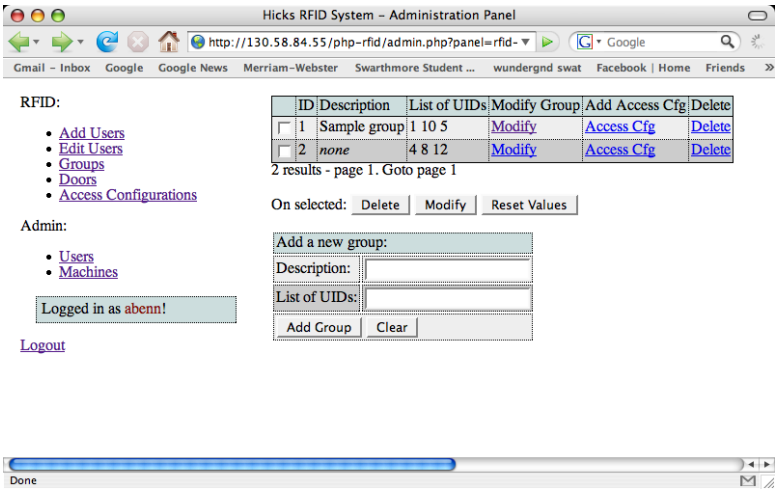

Figure 13: RFID Group Panel.

4.3.3 Doors

A *Door* is a point-of-entry controlled by the system. Each Door has an associated set of fields:

- ID
- Door Name
- Location
- Internal/External
- IP Address
- Default Configuration Setting

All fields are optional except Internal/External. ID is specified by the system and cannot be changed. The Door Name should be chosen carefully: this is the string sent by the door controller to the server to verify its identity, and probably shouldn't be too long or contain any spaces. Location is only used for administrators to remember where the Door is, as is Internal/External. The IP Address field must match the IP address of the door controller. Finally, the Default Configuration Setting is a string representing the default allow/deny rules to be applied to this door. It must be formatted in a specific way; this formatting is specified in the Access Configuration section below.

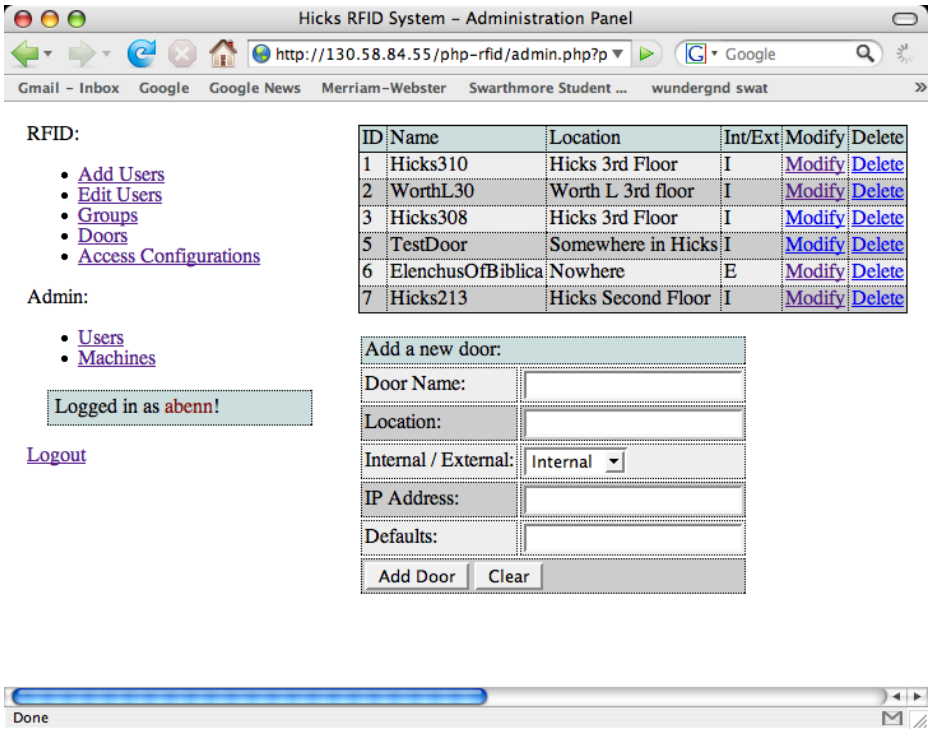

Figure 14: RFID Door Panel.

Doors are managed in the Door panel, shown in Figure 14. Here, an administrator can modify and delete existing Doors, or add a new Door. Keep in mind that deleting a Door may have unintended consequences: since a User's list of accessible doors simply lists Door's ID numbers, if a Door is deleted and another Door is later added that has the same ID, a User may be allowed into that new Door even if they were not explicitly given access to that Door. A future project may be to automatically update all Users' accessible door lists to reflect deletions of Doors.

4.3.4 Access Configurations

Basic door access control can be executed using the List of Doors field for each user, or by setting up a Default Configuration Setting on each door, but for more precise and powerful control, Access Configurations can be used. An *Access Configuration (AC)* is a rule for precisely controlling access to a door on a per-door, per-group, or per-user basis. An AC specifies a period or set of periods of time during which a subset of users are granted or denied access to a Door or Doors. There are several fields in an AC record:

- Access ID
- UID – User ID
- GID – Group ID
- Door
- A/D – Allow / Deny
- Times

Any or all of UID, GID, and Door may be omitted; a blank field specifies that that AC will apply to all possible values of that field. For example, an AC can specify GID and Door, but not UID, which will cause it to be applied to all Users in the specified Group who are trying to open a specific Door.

ACs are managed through the Access Configuration panel, shown in Figure 15. The topmost element of this panel is a map of the allowed times for the currently selected combination of Door, Group and User. To select a specific User, click "Grab ID" and select "Access Cfg" for the desired user, or type the user's ID in the UID box. Next, if the AC is intended to apply to access on a single door, choose it from the Door drop-down menu. Next, choose a day or days for this configuration to apply on, a start time, an end time, and whether the User is allowed or denied within the time range.

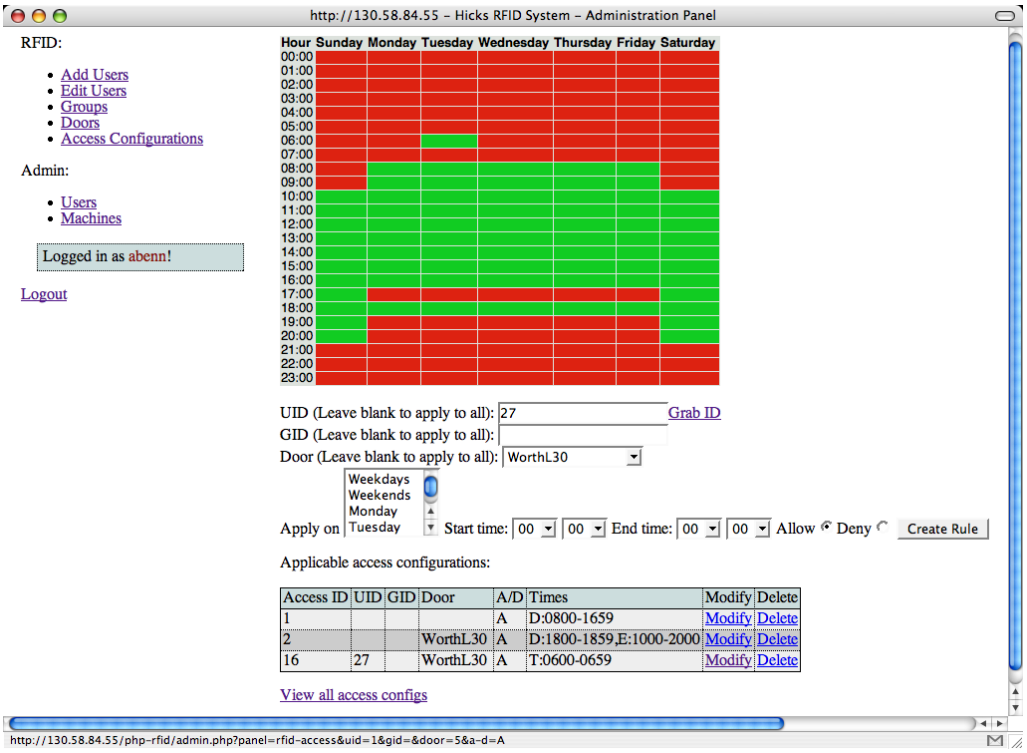

Figure 15: Access Control panel with example field values.

ACs are applied hierarchically: more specific ACs override the effect of more general ones. For example, an AC that specifies GID 10 and Door 1 will override an AC that only specifies Door 1 for all members of GID 10. Similarly, an AC that specifies both GID 10 and Door 1 will override an AC that only specifies GID 10. Deny ACs secondarily override Allow ACs, so if there are two ACs with the same UID, GID, and Door, but one specifies Allow times and the other specifies Deny times, the Deny AC will apply during times that fall within the time ranges of both ACs.

The complexity of the Times field warrants some discussion. When creating an AC, this field will be initialized to a single time range for a specified list of days of the week. Later, however, additional time ranges can be added to an AC, so a single AC could, for example, cover Tuesdays from 1-4 PM and Thursdays from 2-6PM. A single time entry contains one or more day characters, a colon, a start time, a hyphen, and an end time. The Times field may contain one or more time entries, separated by commas. Our example Times field would then look like "T:1300-1600,R:1400-1800". Times of day are in four-digit 24 hour

military time, 0000 to 2359. Note that the internal representation of this field is managed by the administration scripts; the user need only worry about this field when reading the Times field or when manually modifying the database fields themselves.

4.3.5 Administration Users

In order to access the administration interface, an administrator must log in with a username and password. Administration Users (AUs) are used to manage this log-in information. Each AU account stores a username, password, and email address. The password cannot be retrieved once it is set; the password must either be reset or manually changed, or a new account must be created.

AUs are managed through the Admin Users panel, which is selected by clicking Users under the Admin heading on the left side of the web interface. AUs can be added, edited, or deleted here. To add a user, type a username and password into the form labeled "Add new user." This form is shown in Figure 16. Users' usernames and passwords can be updated by selecting Modify next to the corresponding username.

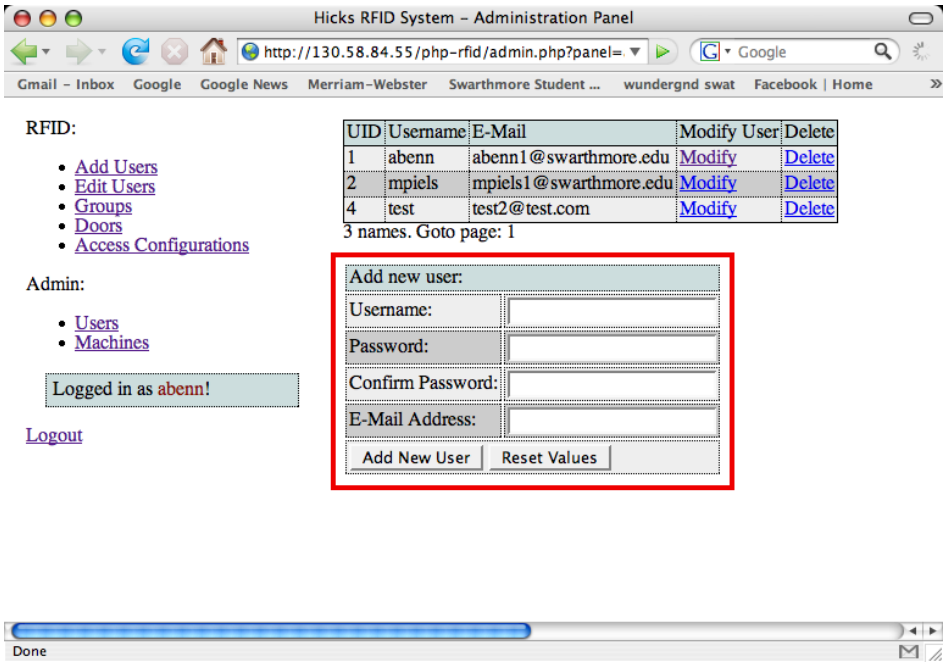

Figure 16: Adding an AU

4.3.6 Administration Machines

As an additional layer of security, administrators can only connect to the web administration interface from computers on the list of designated Administration Machines (AMs). Each Machine has a Name, Location, and IP address. A request to log into the web interface is rejected if it does not come from an IP of an AM. It is vitally important to keep the list of AMs up-to-date in order to ease web administration. Failure to do so may require physical access to the server machine to make changes, or manual manipulation of the MySQL database structures.

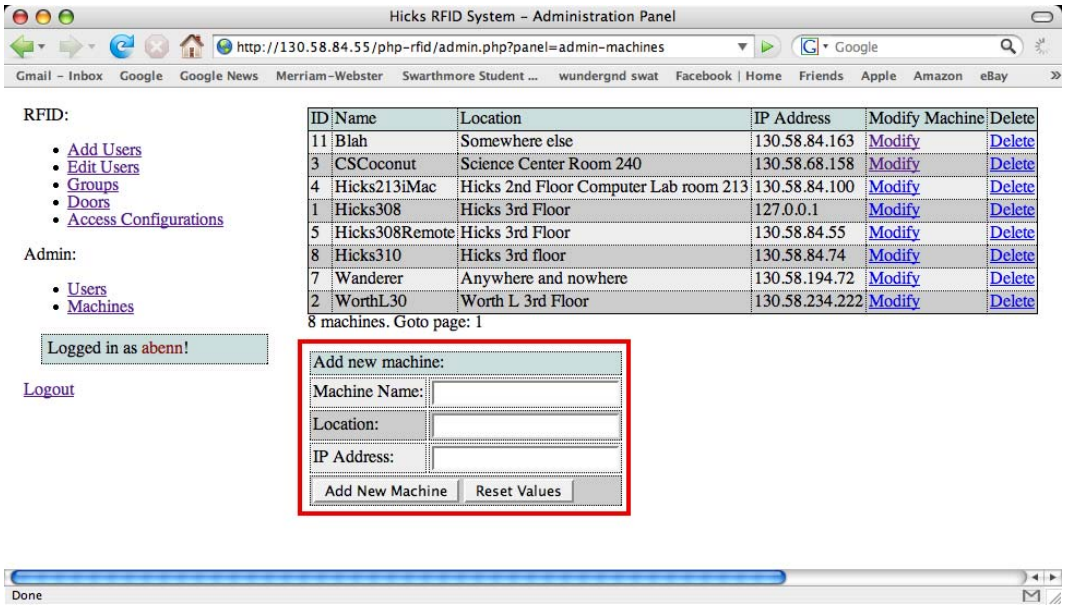

Figure 17: Adding an AM

AMs may be configured from the Machines panel under the Admin heading of the web interface, shown in Figure 17. Adding an AM with a new IP address allows an administrator to connect to the server using a machine with that IP.

4.4 Server Implementation Details

As time goes on it may become necessary to modify the existing implementation to provide bugfixes, add new features, etc. Below are the details of the administration panel, the remote query interface, and the underlying database.

4.4.1 Database

All of the configuration information is stored on the server machine in a MySQL database. The default database and login information is stored in global variables declared in the file common.php. The tables in this database and their fields are listed in Table 2.

Table 2: List of MySQL Database Tables and Their Fields

Table name	What goes here	Fields
rfid_access_configs	ACs	*aid*, uid, gid, door_id, allow_deny, times
rfid_admin_machine	AMs	*id*, name, location, ip_addr
rfid_admin_user	AUs	*uid*, username, password, email
rfid_door	Doors	*id*, name, ip_addr, location, int_ext, defaults
rfid_user	Users	*id*, u_lastname, u_firstname, doors_accessible, tag, email
rfid_group	Groups	*id*, uid_list, desc

Italic fields are primary keys and should never be modified or manually set when inserting new entries into a table. Underlined fields are indexed for fast lookup. Be aware of the size limits on these fields when writing code that modifies the MySQL data structures.

There are a few fields for which formatting is important to ensure correct functioning of the server software. The 'times' field in rfid_access_configs must be formatted as described in the AC section above. The 'defaults' field in rfid_door must also follow this format. In rfid_user, 'doors_accessible' must be a list of door ids, separated by commas, with no other characters. In rfid_group, 'uid_list' needs to be a list of User ids separated by spaces. The first uid must have a space before it, and the last uid must have a space after it. This has to do with the way uids are searched to determine which Groups a specific User is in.

4.4.2 Administration Panel

The administration panel is entirely coded in PHP. There are four primary script files that the client interacts directly with, and three common files that are used by the primary scripts.

Primary script files:

- admin_login.php – log-in prompt window
- admin_verify.php – user credential verification: username, password, Admin Machine
- admin.php – administration panels and entry modification prompts
- admin_update.php – effect changes on the database

Common files:

- common.php – stores global constants, such as MySQL database, and a few common functions
- admin_common.php – stores functions used by admin.php, admin_login.php and admin_update.php
- query_common.php – stores functions needed by both query.php (described in next section) and admin.php

For more details of the inner workings of these scripts, look to the comments in each script file. Beware, though: admin_update.php weighs in at 1500+ lines, and admin.php is over 2000 lines long!

4.4.3 Remote Query Interface

When a user swipes his or her card, the controller reads the card and sends a query request with the card information to the server. The card's tag consists of an 8-bit facilities code and a 16-bit card code. This is carried out by making an HTTP request to the script query.php. HTTP is carried by TCP/IP connections, which provide packet delivery guarantees, unlike UDP/IP, an alternative we examined whose main virtue is its simplicity.

There are two request types that can be made to query.php. Since RFID tags are broken down into facilities code and card code, the controller can send a request with both

components separate, or can combine them into a single integer and send this integer instead. Both methods must send the parameters using HTTP GET. Method 1: facilities code is sent in the variable rfid-fac, and card code is sent in the variable rfid-card. Method 2: the combined code is sent in the variable rfid. Since the facilities code is 8 bits and the card code is 16 bits, the combined code is constructed by bit shifting the facilities code left 16 bits and bitwise ORing the result with the card code. If Method 1 is used, both rfid-fac and rfid-card must be set.

The result of this request is either the text "true", "false", or "ERR=" followed by an error code number and description. Error codes that may be encountered:

- 1 – script error: include file could not be found at runtime
- 3 – The requesting IP does not match the IP address of any Doors in the Door database
- 4 – The requesting controller's IP is in the database, but it supplied the incorrect door name
- 5 – error in GET parameter: one or more GET parameters could not be found or were malformed

Other error numbers may be added as the software is revised.

For additional details on the implementation of the query scripts, read the comments in the files query.php and query_common.php. It is worth noting that query requests are never logged or otherwise stored; this was a design choice made to guarantee privacy of individuals using the system.

4.4.4 Update Script

The update script provides a mechanism for the controller to update its time, local access tables, and server IP address. Since the update involves sending a large amount of information, and the PIC has very limited memory resources, the update script breaks this information into small chunks before sending. No message is longer than approximately 100 bytes, not including the header. Since the PIC must simultaneously juggle performing the TCP transaction and writing the lookup table to EEPROM, it is vital to keep the update script's messages small.

An update transaction involves the following steps. First, the controller requests the script update.php, passing its door name as a GET variable. Next, the server responds with a message containing a unique session id, or SID. Then the controller requests the first user's access table, passing the SID with each request to help the server track each update request. The server sends each user's information in small chunks. When no more users remain, the server sends the current time and the IP address that the controller should use for future server requests. The time is in seconds since the beginning of the week.

Further details can be found in the comments of update.php.

5.0 Conclusion

This project sought to provide an affordable and feature-complete alternative to expensive commercial RFID entry systems. The result has achieved all design goals, and is already in use in Hicks 310. The completion and successful demonstration of the system should serve as a launching point to promote campus-wide use of RFID. All code will be made publicly available as well, potentially providing countless others with the tools they need to set up their own inexpensive RFID-based systems.

6.0 Acknowledgements

We would like to thank Professor Erik Cheever both for advising this project and, as department chair, for clearly defining department's goals for this system. Mary Hasbrouck was instrumental to this project: she provided all the information we have regarding the Computer Science and ITS administered systems and the RFID cards that Swarthmore uses. She also prepared a guide to RFID entry system-related issues that was instrumental in the planning phase of this project. Tom Cochrane helped us understand hardware-related concerns and ordered the door strikes on our behalf. Doug Judy and Grant Smith performed all installation on short notice and without very clear direction from us. Finally, Edmond Jaoudi put up with our incessant and probably obnoxious requests for parts, most notably the Ethernet controller, which was difficult to order in the small quantity desired.

7.0 References

HID, Application Note 002 by B. Postma, last updated 2004.

Informal documents relating to the Science Center and ITS systems prepared by Mary Hasbrouck.

Microchip, Application Note 833: the Microchip TCP/IP Stack v. 2.15 by Nilesh Rajbarti, 2002.

Microchip, MPLAB C18 C Compiler Libraries.

Microchip, MPLAB C18 C Compiler User's Guide.

Microchip, PIC18F 2420/2520/4420/4520 datasheet.

Microchip, PICDEM.net Embedded Internet/Ethernet Demonstration Board User's Guide.

RealTek, RTL8019AS RealTek Full Duplex Ethernet Controller with Plug and Play (PNP) Function datasheet.

Siemens, proposal for Dupont Science Center Access System, 22 March 2004.

Siemens, proposal for ITS Checkpoint System Replacement System, 8 June 2004.