A Toolkit for Building Dependable and Extensible Home Networking Applications

Yi-Min Wang *Microsoft Research Redmond, WA*

Wilf Russell *Microsoft Research Redmond, WA*

Anish Arora *Ohio State University Columbus, OH*

Abstract

Dependability and extensibility are two of the key requirements to successful home networking. In this paper, we describe the design and implementation of a software development toolkit for building dependable and extensible home networking applications. A reliable Soft-State Store (SSS) is implemented as a shared infrastructure to simplify robust distributed programming against device and object failures. SSS supports multi-timescale refreshes and selectively uses persistence to accommodate the battery power and network bandwidth constraints in the home networking environment. A publish/subscribe event system allows any changes in the SSS to be propagated to interested subscribers, which then perform appropriate adaptive, corrective, alerting, or cleanup actions. An Attribute-Based Lookup Service (ABLS) and a Name-Based Lookup Service (NBLS), both implemented on top of the SSS for robustness, provide a level of indirection for supporting extensibility as well as allowing user-friendly, natural language-based access. We demonstrate the use of the toolkit for building a home networking system in an actual deployment. We describe two end-to-end remote home automation applications, present performance results, and report our experiences.

1. Introduction

The success of the Web has demonstrated the great power of being connected. As the world increasingly moves towards a fully connected one, home networking that connects smart household appliances together and connects them to the Internet becomes the natural next step. The simplest form of home networking has emerged to allow the sharing of files, printers, and Internet connections, and to enable networked PC games. The next wave of advanced home networking applications will include family communications, device automation, digital Audio/Video (A/V) distribution, remote maintenance of household appliances, etc.

In the *Aladdin* home networking project, we focus on building end-to-end user scenarios and using those to drive the design and implementation of the required system infrastructure. Based on our experience in implementing and deploying a home networking system in a house with real users, we have identified dependability, extensibility, user-friendly interface, and remote access capability as the four key requirements to useful and successful home networking. *Dependability* ensures that failures of hardware devices and software objects will be detected, appropriate recovery or cleanup actions will be performed, and homeowners will be alerted if necessary. *Extensibility* allows any new device to be plugged into any of the in-home networks (phoneline, powerline, wireless, etc.) and become available to all existing applications. *User-friendly interface* allows users to control appliances and retrieve information in a natural way. *Remote access capability* lets homeowners connect to their homes at any time, from any place, and on any device.

In this paper, we give an overview of the Aladdin system and software architectures for addressing the above issues. We then focus our attention on the system infrastructure provided by the toolkit that we have constructed to simplify the task of building dependable and extensible home networking applications. Finally, we describe two end-to-end remote home automation applications to demonstrate the power of the toolkit and to evaluate its performance.

Compared to traditional networked environments, the home networking environment is more heterogeneous and dynamic. It is heterogeneous because consumer devices manufactured by different vendors, connected to different networks, and running different protocols are likely to coexist. It is dynamic because these consumer devices tend to connect, disconnect, move, fail, etc. more often. We propose a *Soft-State Store* with *Publish/Subscribe eventing* and lookup services built on top of the store as a uniform framework for robust management of diverse devices, where *soft-state* is defined as *volatile or nonvolatile states that will expire if not refreshed within a pre-determined, but configurable, amount of time*. Both hardware devices and software objects periodically announce their existence and optionally their states to the soft-state store, which may consist of multiple individual stores distributed throughout the system. When a device/object fails or gets disconnected, its corresponding soft-states eventually time out. Such changes to the soft-state store generate events to all interested subscribers, which then perform appropriate actions to adapt to the changes.

The paper is organized as follows. Section 2 gives an overview of the system and software architectures of the Aladdin home networking system. Section 3 describes the design and implementation of the soft-state store, the event system, and the lookup services. Section 4 describes two remote home automation applications built with the toolkit. Section 5 reports our experiences from the actual usage of the system. Section 6 discusses related work, and Section 7 summarizes the paper.

2. Overview of the Aladdin System

2.1. Distributed System Architecture for Home Networking

Figure 1(a) illustrates an ideal home networking system where the house is wired for running Ethernet and most devices are smart, networked devices connected directly to the Ethernet and running device control software themselves. A *home gateway* machine sits between the home network and the external communication infrastructures including the Internet and telephony. *User Access Points (UAPs)* are wall-mounted or stand-alone flat-panel displays deployed throughout the house to allow convenient access to in-home information (calendars, etc.) as well as the Internet from anywhere in the house. UAPs also expose Web-based, natural language-based, and voice-based interfaces for remotely controlling household devices and for monitoring environmental factors through remote sensors. Network bridges are provided for bridging devices on other communication media such as the powerline, Radio Frequency (RF), InfraRed (IR), and A/V cables to the Ethernet backbone. Such devices do not directly connect to the Ethernet for various reasons including legacy, cost, security, market competition, etc.

Since smart devices are not yet generally available, the current Aladdin system accommodates existing devices by using multiple Windows 98 PCs and their peripherals to serve as both User Access Points and network bridges, as shown in Figure 1(b). (Six PCs are used in this deployment.) The PCs also act as device proxies by running device control software on behalf of the devices. The system is deployed in the first author's three-story house and used by the author on a daily basis.

The PCs are all connected by 1Mbps to 10Mbps Ethernet over the phoneline [H98]. Since the powerline in the house has serious signal attenuation problem that prevents the low-cost consumer powerline devices from reliably communicating between any two outlets, powerline control requests are first routed to the PC that is (electrically) closest to the target device and then bridged to the powerline. Similarly, battery-operated RF devices such as the motion sensors and door sensors are usually short-ranged to preserve battery power. RF signals transmitted by these sensors are received by PCs within their ranges and bridged to the phoneline network to reach other parts of the house. Although currently most Aladdin applications do not require the full processing power of the PCs, they will be useful in the future for running potentially computation-intensive smart-home software including location tracking, person identification, voice recognition, learning, etc. [S+98][M98][Es99]. In total, the current Aladdin system consists of about 60 devices.

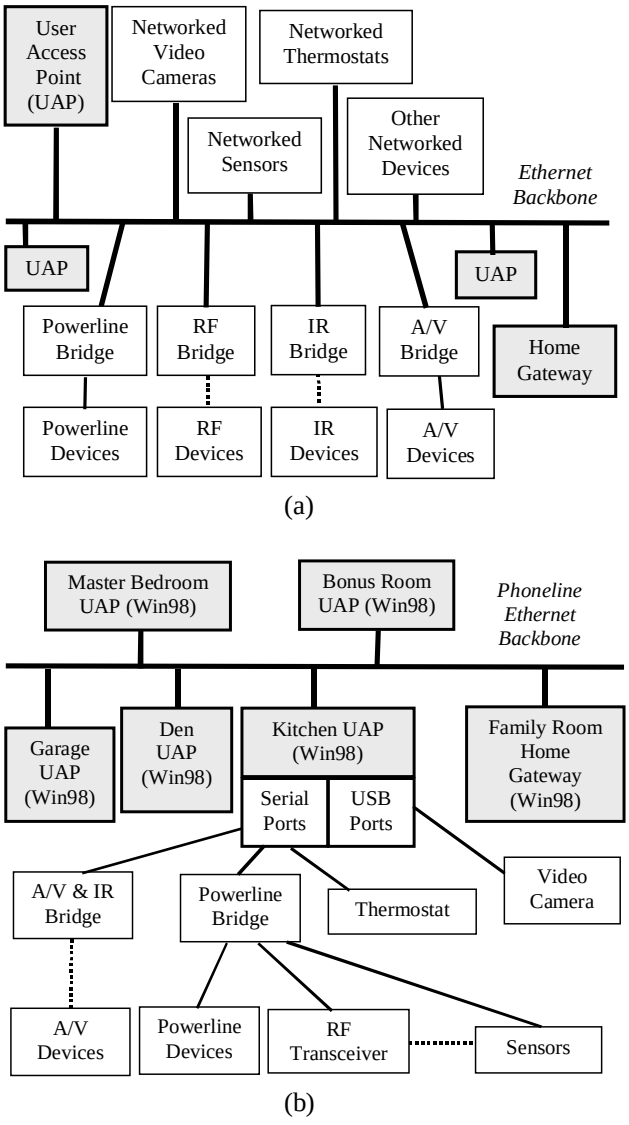

Figure 1. Distributed system architecture of the Aladdin home networking system. (a) Ideal future architecture; (b) Current architecture (only the peripherals of the

kitchen UAP are shown; all the other UAPs have similar setup).

2.2. Software Architecture

Figure 2 shows the overall software architecture of the Aladdin system, which provides an abstraction over the hardware devices, and connects them to external communication infrastructure. We give a brief overview of all three layers in this section, and will focus on the system infrastructure layer in the remainder of the paper.

1. System infrastructure: At the bottom layer, the system infrastructure consists of five components. The soft-state store manages the lifetime and replication of soft-state variables across individual data stores on each machine. It supports a Pub/Sub eventing mechanism that allows programs to subscribe to events related to changes in the store. The attribute-based lookup service maintains a database of all available devices and supports queries based on device attributes such as device type, physical location, etc. The name-based lookup service maintains a table of all running object instances and supports simple name-to-object-address mapping. The device announcement protocol describes how devices not connected to the main Ethernet announce their existence to the attribute-based lookup service. An Aladdin Device Adapter has been built to implement an instance of that protocol for powerline devices. Finally, the system management daemons are responsible for detecting the failures of PCs and devices, and initiating recovery actions. We do not cover this last infrastructure component in this paper.

2. Application layer: There are two types of home networking applications in the current Aladdin system. In the control-type scenario, the applications receive user requests as input, consult the lookup services to identify the devices and device objects that should be involved, and perform actions on them to satisfy the request. Device objects encapsulate device- and network-specific details and present interfaces (sets of method calls) as the programming abstraction for device control. Examples are camera objects for taking snapshots and recording video clips, garage door opener objects for operating the garage doors, etc.

In the sensing-type scenario, the applications monitor a list of environmental factors and take actions when any of the monitored events happens. Through the ABLS, the applications identify the appropriate events to subscribe to. Independently, device daemons act as proxies for sensors by monitoring sensor signals and updating appropriate soft-states to trigger events. For example, an alerting application running on every User Access Point subscribes to all events corresponding to critical sensors (water sensors, temperature sensors, safe-box sensors, etc.) and will sound alerts when any of the sensors fires.

Most commercial home networking products for intelligent sensing and control tend to be closed solutions limited to a single communication medium and are not extensible. Aladdin's approach of relying on the soft-state store, pub/sub eventing, and lookup services to tie together devices connected to different communication media provides a unique opportunity for constructing versatile applications.

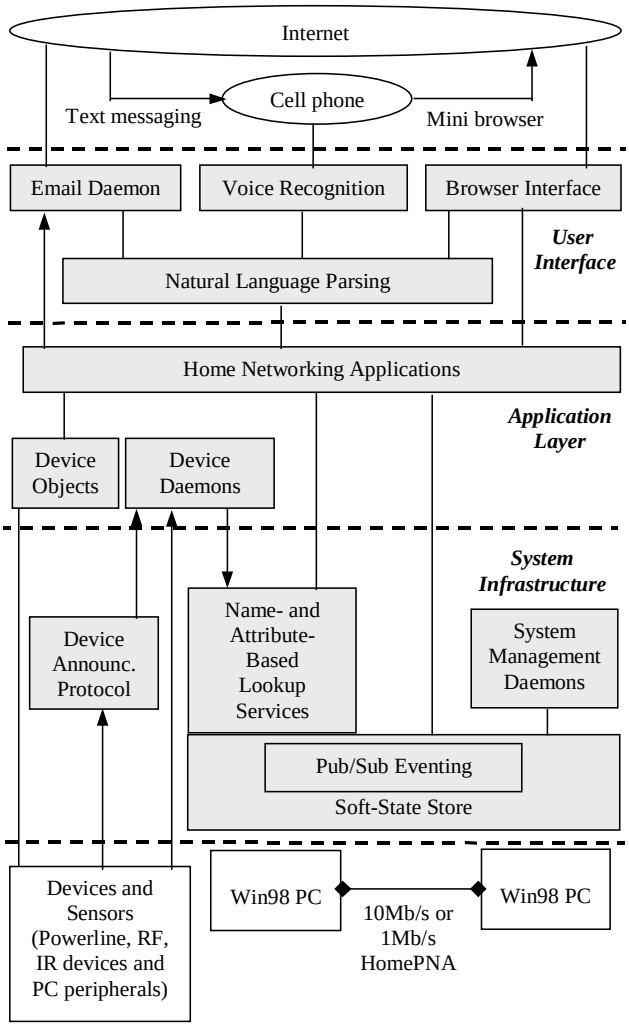

Figure 2. Software architecture of the Aladdin home networking system.

3. User interface: Unlike most other distributed systems, home networking systems are to be used by naïve computer users and so providing friendly user interface is especially important. The current Aladdin system supports three forms of user interface: a *browser interface* that allows the user to browse through all available devices or select devices based on attributes,

and to control devices through point-and-click; a *text-based natural language interface* based on a limited but customizable vocabulary; and a *voice-based interface* that employs speech recognition technology based on the same vocabulary. All three forms are available for in-home use. They are also being extended to support remote home automation when the user is away from home. When DSL or cable modem is available, the same browser interface can be used from remote locations. The text-based natural language interface has been extended to an email-based remote home automation interface. The user can send an email containing a control request to an account hosted by an Internet Service Provider (ISP). (Current deployment uses Microsoft Network, MSN.) The Aladdin email daemon periodically dials up the ISP, retrieves and parses the request, performs the actions, and sends a reply email that may optionally contain video clip(s) confirming the actions. In addition to the standard security mechanisms such as digital signatures and data encryption, the home control vocabulary can be customized to provide additional security. Through the text messaging support provided by cell phones, the email daemon can almost synchronously alert the users wherever they are when, for example, any of the critical sensors fires at home. Work on extending the voice-based interface to work reliably over telephony is still in progress.

3. Home Networking Toolkit

The Aladdin home networking toolkit currently consists of seventeen thousand lines of code packaged as several DLLs and EXEs. In this section, we describe the motivation and design of each component, followed by their specific implementations on Windows 98 and the Application Programming Interfaces (APIs). The soft-state store manages the timeouts and propagation of soft-state variables, and provides a generic publish/subscribe eventing mechanism that reflects any changes to the store. The name-based lookup service is implemented on top of the soft-state store, which actually stores and maintains the table of lookup service entries. In contrast, the attribute-based lookup service stores the entries in a database to support queries, while creating a soft-state variable for each entry to manage the timeouts and eventing. Both lookup services register an event callback conversion module with the soft-state store to convert generic events into their respective domain-specific events. Most applications interact with the system through higher-level lookup services APIs. But the low-level soft-state store APIs are also available.

3.1. Soft-State Store with Eventing

Consider the common requirements for the following scenarios: when a battery-operated garage-door sensor runs out of battery, the system should detect the failure and alert the user; also, it should discard the previous state of the sensor so that home networking applications do not perform erroneous actions based on stale data. When a device suddenly gets disconnected from the system and when an object gets terminated abruptly, their corresponding entries in the lookup services should eventually expire to allow the system to reclaim the space of those entries and to minimize the chance of client applications getting stale information and potentially malfunctioning. When an essential daemon process fails either due to machine crash or process termination, the system should be able to detect the failure and either reset the machine or restart the daemon.

To address the above issues, either the system needs to *ping* the sensor/device/object/daemon or the latter need to send periodic *heartbeats* or *refreshes* to the system. Traditionally, systems with homogeneous and relatively static parts are maintained in terms of *"hard"-states* that are updated upon demand by pinging the parts. By way of contrast, in the heterogeneous and dynamic environment of home networking, it is preferable to maintain systems in terms of *"soft"-states* that are updated periodically by refreshes from the parts.

There are several reasons for preferring soft-states to hard-states in Aladdin. First, to keep the dollar cost low, many consumer sensors are transmitters only and do not support pinging of their status. Second, since many network protocols, object models, and programming paradigms (distributed objects, wire protocols, etc.) are likely to coexist in home networking systems due to market competition, it will not be practical to require the system to ping all devices and objects with various protocols, models and paradigms. Third, the devices/objects being pinged may hang the pinging operations, thereby complicating system robustness. Finally, hard-states complicate the recovery tasks of daemons and protocols upon system crashes. By relying on the refreshes to reconstruct lost data, soft-states greatly simplify the recovery tasks. By the same token, they also simplify maintenance of information about devices that leave the system spontaneously and without announcements.

To simplify the development of home networking applications based on soft-states, we implemented a *Soft-State Store (SSS)* that serves as a shared infrastructure for managing and propagating soft-states across different applications and machines. A provider of a soft-state variable specifies the *refresh interval* and

the *threshold number of missing refreshes* before the variable is timed out. The SSS is responsible for maintaining the time-to-live timers for all providers.

An event system is built on top of the SSS to allow applications to subscribe to events related to data changes in the SSS. Subscribers of events can be notified, for example, of (1) the failure of a critical system daemon so as to perform fail-over recovery, (2) the addition of a new device so as to adapt to the new overall device availability, (3) the failure of a device so as to raise an alert to the users, and (4) the disappearance of a resource-consuming entity so as to perform cleanup actions. Subscriptions themselves are maintained in the SSS and periodically refreshed by the subscribers.

As will be demonstrated in the lookup service section, some of the refreshes in the home networking environment cannot be performed with high frequencies. Low-frequency refreshes ranging from a couple of hours to a day may be necessary to accommodate power constraints of battery-operated commodity devices and sensors, and the bandwidth limitations of some of the in-home networks. This is in contrast to the conventional case where the refresh rate is as high as every few seconds or minutes, as a result of which it is adequate to store the soft-states in volatile memory and, upon a system failure, to rely completely on the high-frequency refreshes for recovery.

To ensure acceptable data quality for soft-states whose refresh frequency is low, we introduce the notion of *persistent soft-states*. By maintaining these soft-states in persistent storage, the SSS can recover them after total system failures without waiting for a long time until the next refresh. For example, reliable remote operation of the garage door can be resumed immediately after system failure without waiting for the door sensors to refresh their states in up to 90 minutes. One must, however, pay special attention not to restore potentially stale, persisted soft-states. Each persisted soft-state variable is time-stamped with the *latest refresh time*, and *the number of missing refreshes* is also recorded. When the persistent file is used after a recovery to restore part of the soft-state store, the current time is compared with the time stamp of each variable and the time difference divided by the refresh interval is used to increase the number of missing refreshes. That is, any potential refreshes that may have happened during the system downtime are treated as missing refreshes. Any variable with the adjusted number of missing refreshes exceeding the threshold is not restored.

The Aladdin system is subject to a variety of faults that are weaker than total system failures but must be considered as they affect the freshness and availability of SSS data. At least one PC, typically the home gateway machine, is connected to an Uninterruptible Power Supply (UPS) so that short power outages do not cause total system failures. Upon a PC reboot, SSS maintains a leader that allows the rebooted PC to catch up with the SSS replicas: the leader streams its cache of soft-states to the rebooted PC over the phoneline Ethernet, thus minimizing the staleness of SSS data and the outage period of SSS service at the latter.

Implementation

The soft-state store implementation consists of a COM (Component Object Model [B98]) EXE server and a client-side COM DLL. It supports two interfaces. The ***ISoftStateStoreAdmin*** interface contains the following methods:

- *RegisterSoftStateTypes()* and *RemoveSoftStateTypes()* for defining and removing custom types and sub-types of soft-state variables, respectively; for example, the CRITICAL_SENSOR type is registered as a subtype of the SENSOR type.
- *ChangeTimeoutOnVar()* and *ChangeTimeoutOnSubscription()* for manipulating the metadata of soft-state variables and event subscriptions. Specifically, they allow changes to be made to the refresh intervals and the threshold number of missing refreshes.

The second interface, ***ISoftStateStore***, consists of:

- *RegisterSoftStateVars()* and *RemoveSoftStateVars()* for creating and deleting individual soft-state variables of particular types, respectively. *RegisterSoftStateVars()* also takes as input a flag indicating whether auto-refreshes are requested. If the flag is set to true, the client-side SSS DLL automatically sends periodic refreshes on the client's behalf. When the client application terminates without removing its variables, the DLL also dies and the variables will eventually time out.
- *SetValue()* and *GetValue()* for setting and retrieving the values of variables, respectively; *GetValue()* can optionally return the current number of missing refreshes and time to next planned refresh. Applications can use such information on potential data staleness to perform different actions, if desirable.
- *GetVarsOfType()* returns the names and values of all soft-state variables and optionally subtypes of a particular type.
- *SubscribeEvents()* and *UnsubscribeEvents()* for subscribing and unsubscribing events related to the changes of individual soft-state variables or all variables of certain types. The subscriptions

themselves are soft-states as well so that the system can remove stale subscriptions when the subscribers terminate without performing proper cleanup. The subscriptions, however, are only local and not replicated to other machines.

The callback address supplied as part of the input parameters to a *SubscribeEvents()* call is in the form of a unique NBLS name, to be described shortly, to allow late binding. When the SSS needs to fire an event to a subscriber, it resolves the name at that time through the NBLS to obtain the subscriber's up-to-date addresses. This late-binding model is essential for coping with failures, recoveries, and mobility of the subscribers, and is particularly important for subscriptions to rare events. If a subscriber uses the COM programming paradigm, it must support the *ISSSNotify* interface and announce the (marshaled) interface pointer to NBLS. ***ISSSNotify*** consists of four methods:

- *Added()* is invoked when a new soft-state variable is added to a subscribed type;
- *Changed()* and *Deleted()* are invoked when a variable or type is updated and deleted, respectively. A flag in the *Deleted()* call indicates if the deletion was the result of an explicit removal operation or expiration due to missing refreshes.
- *MetaUpdate()* notifies the subscriber that metadata associated with the variables or types have been changed. This is especially useful should the system need to change the timeout parameters when demand for a particular soft-state variable decreases, SSS load increases, or A/V applications demanding high bandwidth are being started.

3.2. Lookup Services

Lookup services are the key to extensibility. By providing one level of indirection, lookup services allow devices and objects to dynamically join the system and be available to client applications. In the home networking environment, we have found it useful to divide the lookup services into two layers. At the upper layer, the *Attribute-Based Lookup Service (ABLS)* maintains a database of available devices, sensors, installed software modules, etc. It supports queries based on a combination of attributes and returns a list of unique names (called *NBLS names*) identifying the matching items. For example, the query "device=curtain and floor=1" returns the names of all first-floor curtains. At the lower layer, the *Name-Based Lookup Service (NBLS)* maintains a table of available software objects. It takes a unique name as input and returns a list of "addresses" that can be used to contact the target object. For example, the unique name "living_room_curtain" may generate an NBLS response that contains two addresses for reaching the curtain object: one can be unmarshaled into a DCOM pointer for making synchronous calls, and another one can be unmarshaled into a handle to a message queue for making asynchronous calls. Both ABLS and NBLS rely on the soft-state store for robustness against sudden disappearances of devices and objects.

3.2.1. Attribute-Based Lookup Service (ABLS)

Attribute-based lookup service provides the foundation for user-friendly naming of devices. Instead of identifying each device by its low-level communication address, an Aladdin user identifies each device by its *physical location* in the house, which is a notion most familiar to the user and especially useful for remote automation; for example, "the lamp on the garage side of the kitchen" or "the VCR on the second floor". Since most of the devices in the current Aladdin system are ordinary, non-Ethernet devices, the main challenge is to devise a mechanism that would allow such devices to announce their attributes into the ABLS and to automatically include their physical location information in the announcement.

To address the above issue, we introduce the concept of an *Aladdin Device Adapter (ADA)*, which serves as the representative of an attaching device to participate in the Aladdin lookup service-based system. To make the presentation more concrete, we will describe the ADA in the specific context of the X10 powerline control protocol [S98]. The general concept is applicable to other communication media and protocols as well.

The X10 protocol allows for 256 unique addresses, each of which is specified by a *house code* (A through P) and a *unit code* (1 through 16). To allow a device to announce its physical location, we perform a one-time configuration task by *assigning a unique X10 address to every outlet that the user would like to control*, and *storing in ABLS the mapping of the address to the set of physical location attributes associated with the outlet*. For example, the address K3 may be assigned to an outlet on the garage side of the kitchen. To connect an ordinary device to an outlet, the device is plugged into an Aladdin Device Adapter and the Adapter is plugged into the outlet. Similar to the common X10 receiver modules, the Adapter has two dials for setting its X10 address, which in our scheme must be set to the address assigned to the outlet. The Adapter also has a third dial for selecting a *device code* among a pre-defined set of codes ("1" for lamps, "2" for fans, etc.) to indicate the type of the attaching device.

The Aladdin Device Adapter performs three tasks: *announce*, *revoke*, and *refresh*. When the device is switched on (and hence available for remote control),

the Adapter detects that through an AC current sensor and announces the device code and the X10 address over the powerline in the form of an extended X10 code. Device daemons running on a subset of PCs receive such an announcement and register with the ABLS (see Figure 2) the received device code and X10 address, which gets translated into the device's physical location information. The PCs also register themselves as the proxy controllers that can be contacted to instantiate appropriate device objects to control the device. When the device is broken or unplugged from the Adapter, the Adapter detects that through the current sensor and sends out another extended X10 code to indicate that the device has left the system. The receiving PCs then contact the ABLS to remove the device entry. The Adapter is also responsible for sending periodic announcements so that the proxy controllers can refresh the soft-state ABLS entry for the device. When the Adapter itself is unplugged from the outlet, the periodic refreshes stop and the device's ABLS entry will eventually be timed out to reflect the fact. Since powerline has limited bandwidth and devices do not join and leave very often, the refresh rates used by the Adapters are typically in the order of hours.

Sensors refresh their ABLS entries through a similar bridging process performed by the device daemon. Some existing low-cost consumer sensors already follow the soft-state model by sending periodic announcements approximately every 90 minutes. To participate in the ADA protocol, they need to be enhanced with a third dial for selecting the sensor type code. Since many consumer sensors operate on batteries to allow flexible installation, such low-frequency soft-state refreshes are particularly important for conserving battery power. Sensor refreshes in fact announce more than just the existence of a sensor; they also provide the current state of the sensor since most consumer sensors do not support polling of their states. Such state information also resides in ABLS and will expire at the same time the associated lookup service entry expires to prevent any client applications from accessing the stale state of a broken sensor.

Implementation

We took the wire protocol approach in the design of ABLS. Any smart device that plugs into the phoneline Ethernet can interact with the ABLS directly by generating messages conforming to the wire protocol. A COM API layer is defined on top of the wire protocol to simplify ABLS programming on PCs, currently the only Ethernet devices in the system.

The ABLS wire protocol specifies the XML tags for the request and response of two groups of operations: administrative operations and service operations. The first group includes operations involving the definition of the vocabulary for physical location attributes, the mapping of location names to sets of location attributes, the assignment of device type codes, the registration of ABLS data types for event subscriptions, etc. The second group consists of operations for announcing/refreshing and removing device entries, for submitting device queries in the form of attribute-value pairs, and for subscribing and unsubscribing events related to changes in ABLS.

ABLS requests are multicast to all ABLS daemon replicas running on a subset of PCs and listening on a well-known multicast address and port. All active daemons perform updates if necessary, but only the leader sends a response. The current Aladdin implementation uses the Jet database engine [HF95] to provide persistent storage of ABLS entries and to support SQL-like queries. The soft-state timers for the entries are maintained separately by the SSS.

To simplify ABLS programming, a client-side DLL provides a COM API layer that takes application-specific data as input and generates ABLS requests according to the wire protocol. The APIs are factored into two interfaces: ***IABLSAdmin*** and ***IABLSServices***. For example, *IABLSAdmin::SetLocaleInfo()* is typically used to specify the mapping between an X10 address and a set of physical location attributes; *IABLSAdmin::SetDeviceInfo()* assigns a device code to a particular device type; *IABLSServices::Update()* is used to indicate that a device of a particular device type is now controllable at a communication address. It also specifies the host name of the controlling PC.

IABLSServices::LookUp() takes as input a list of attribute-value pairs and returns the number of matching device entries and an *Enum* structure that can be enumerated to retrieve all matching entries. Each entry may contain multiple subentries, each of which is associated with a PC candidate that can be contacted to control the device and is identified by a unique *NBLS name*. For each matching device, the client application then chooses one of the NBLS names and submits it to the NBLS to locate the corresponding device object. To simplify the programming for the common case, an entry from the *Enum* supports a *GetAddresses()* method that implicitly performs the NBLS lookup using the NBLS name of the first subentry. In addition, *GetAddresses()* takes as input a flag indicating whether automatic activation is requested. If the flag is set and the object instance is not running, the ABLS will, based on the information supplied by *IABLSServices::Update(),* create the object instance on an appropriate machine and return the list of addresses to the caller.

3.2.2. Name-Based Lookup Service (NBLS)

The name-based lookup service essentially provides a lookup table that maps each unique name to the object instance identified by that name. The Aladdin NBLS, however, has several unique features that differentiate it from other name services and object locator services. First, it is built on top of the soft-state store and therefore robust against object failures and non-graceful termination. Client applications can take advantage of the publish/subscribe eventing mechanism to request to be notified when a target object is instantiated and announced to the NBLS, without having to periodically poll the service. Object instances that, for example, run on battery-operated devices may wish to perform refreshes at a low rate. Due to the persistence support of the SSS, such objects can remain available to clients through the lookup service immediately after the NBLS fails and recovers.

The second feature of the NBLS is its extensibility. Most device objects in the current Aladdin system have been built as DCOM objects because the object-oriented RPC model greatly simplified programming. However, several limitations of DCOM have been identified. For example, the synchronous RPC model does not work well in the presence of failures and in loosely coupled environments; the requirement of proxy/stub installation creates complexity, hinders application evolution, and creates versioning problems; the binary marshalling format and the private wire protocol do not work well in an open environment and make it hard to do introspection. To allow the Aladdin system to evolve as the distributed computing world moves towards loosely-coupled, asynchronous messaging, we built extensibility into NBLS, instead of making it a mere object locator service for DCOM. The heterogeneous nature of the home networking environment further calls for the need of an extensible lookup service. Future generations of smart devices may plug into different communication media and run different communication protocols. Even for devices based on the same communication protocol, market competitions from different vendors may eventually fragment the market and require devices running software with different object models or programming paradigms (distributed objects, wire protocols, etc.) to coexist. To maximize market acceptance, some devices may choose to support multiple protocols, models, and paradigms.

To provide extensibility, the Aladdin NBLS allows mapping a unique name to potentially multiple addresses. Each address starts with a prefix, identifying the protocol/model/paradigm, followed by an opaque string that encapsulates addressing information. Upon receiving a list of addresses as the response to an NBLS query, the client application can enumerate the list to identify the addresses that it can understand, choose the one that is most desirable, and unmarshal the address into a communication handle if necessary.

As an example, a DCOM object may choose to support an additional address for queued, asynchronous communication in order to accommodate non-RPC clients. The refresh interval for the synchronous address (i.e., a marshaled DCOM interface pointer) is typically in the range of tens of seconds to allow fast detection of failed objects. The interval for the queued address is usually longer (minutes to tens of minutes) to accommodate failures and recoveries. When the object's hosting machine fails and reboots, the object is destroyed and its synchronous address soon gets timed out. If a client wants to send a request to the object at that time and a queued call is acceptable, it can do so by using the remaining queued address. When the object restarts, it will check its message queue and process the request, much like a person walks into his/her office in the morning and checks voice mail or email.

Implementation

The NBLS wire protocol uses similar semantics to an earlier version of the Simple Service Discovery Protocol (SSDP) [G+99]. It specifies the XML tags for the request and response of device/object announcements, revocations, and lookups. Similar to the ABLS, a client-side DLL exposes a COM interface ***INBLSServices*** to simplify NBLS programming on PCs. For DCOM applications, a second client-side DLL provides additional helper APIs to further simplify programming: *CreateDCOMRefDisplayName()* marshals a DCOM interface pointer into an address string with the "*DCOMRef*" prefix; *BindToDisplayName()* takes such an address string as input and unmarshals it back to an interface pointer. On the server side, NBLS is tightly coupled with the SSS and relies on the SSS to store the lookup service entries as well as to manage timeouts.

4. Applications and Performance Results

We now describe two of the home networking applications that we have built using the toolkit: email-based remote automation and cell phone-based remote notification. We demonstrate how the various system infrastructure components are involved in these end-to-end application scenarios and how the toolkit facilitates the support for dependability and extensibility. Although home networking applications are usually not performance-sensitive, we measure the overhead of individual system components and present them in the context of end-to-end latencies.

- **Email-based remote control of garage door**

The user rushes out for a meeting and forgets to close the garage door. Sitting in the conference room, he/she sends a signed and encrypted email request to remotely close the garage door. After the email daemon receives, authenticates and parses the request, it locates the garage door opener object through ABLS and NBLS, and instructs the object to close the garage door. To ensure reliable operation, the garage door is equipped with three inexpensive, redundant sensors: a magnetic sensor that detects that the door is at least two inches open, and two horizontal/vertical sensors that detect that the door is at least 25% and 75% open, respectively. The object first reads the states of the sensors from the ABLS to verify that the door is indeed open, after which it sends a powerline control command to achieve the same effect of pushing the garage door opener button. The sensors periodically send out state refresh signals, which are received by the device daemon and converted to ABLS soft-state refreshes. If any of the sensors runs out of battery, its soft-state variable will eventually be timed out so that it does not cause the object to perform incorrect action. Also, a daemon subscribing to sensor variable deletion events will receive an event and display messages on the UAPs to notify the homeowner to change the batteries. To add additional confidence in this critical operation, the object locates the camera in the garage through the lookup services and instructs it to take two snapshots of the garage door, one before the action and one after. The two snapshots are sent back to the user as attachments in the reply email.

On the 550MHz home gateway machine, performance numbers for the various system infrastructure components involved in this scenario for each device operation are: 90ms for the language parser, 85ms for the lookup services (with both ABLS and NBLS running on the gateway machine), and 40ms for the *BindToDisplayName()* operation. The overhead is insignificant compared to the time for the actual device operations (several seconds to 15 seconds) and the time for email delivery.

- **Cell phone-based remote notification of emergency**

The device daemon queries the ABLS for all critical sensors and their corresponding powerline signals, and listens on the powerline, looking for matching signals. A water sensor lies in the crawl space. When it detects water, it sends out a powerline signal, which is translated into a soft-state update by the device daemon (see Section 3.2). Since the email daemon subscribes to the event of changes in any soft-state variable of the CRITICAL_SENSOR type, it receives an event callback from the SSS and sends an emergency email to the homeowner's cell phone email address. It also describes the sensor type and the physical location based on the ABLS information; for example, it says "Crawl space water sensor ON" on the cell phone screen. When a new water sensor is installed in the laundry room and registered in the ABLS, the ABLS change event notifies the device daemon to include the monitoring of signals from the new sensor. Since the soft-state variable associated with the new sensor is of the CRITICAL_SENSOR type, the event subscription of the email daemon does not require any update. This example is a simple demonstration of the extensibility of the Aladdin system.

The elapsed time between the detection of water and the ringing of the cell phone is usually 10 to 40 seconds. Occasionally, it may go up to several minutes, depending on the carrier's text messaging performance. If the gateway machine is not on line and so dialing up is necessary, that would take an additional 40 seconds.

5. Experience Report

The Aladdin system has been running in an actual deployment for several months, and used by the homeowner on a daily basis. For the most part, the system delivers satisfactory services of remote control and notification. However, we have observed problems at several layers, which must be solved before such home networking systems can become mainstream.

The lack of reliability of the X10 powerline control protocol is well known. X10 uses a single frequency to transmit signals and so is particularly sensitive to the signal attenuation problem due to the quality of the powerline wires, and the fluctuation in powerline transmission characteristics as devices are plugged into and removed from outlets throughout the house. Basically, the X10 powerline network is partitioned and the partitioning is dynamic. The soft-state-based distributed system solution provided by Aladdin greatly alleviate the problems by using the phoneline to reach powerline partitions and by allowing the lookup services to adapt to the changes in powerline partitioning. However, it is conceivable that some devices or sensors may become completely isolated from the rest of the powerline, rendering reliable control an impossible task. Next-generation powerline control protocols promise to overcome the above problems at a lower layer and present a reliable broadcast network abstraction over the powerline. Whether and when devices using such protocols can become low-cost consumer-ready devices may be a deciding factor on the broad acceptance of home networking.

Security is another major concern. We have observed in more than one occasion that a faulty

powerline computer interface may exhibit Byzantine behavior by sending random commands over the powerline [WRA+00]. This may create security problems if any of the commands happens to be addressed to a critical device. Even if the random commands only turn on and off less critical devices such as lamps, it creates an extremely unpleasant experience for the user and the diagnosis is not trivial. It remains a challenge for consumer electronics manufacturers to produce low-cost, yet dependable devices. It also requires a system management layer capable of detecting, diagnosing, and, if possible, recovering from anomalies created by faulty devices [WRA+00].

Power outage is an important and interesting failure mode. Since most of the devices in the current Aladdin system rely on electrical power either for operation or for signal transmission, prolonged power outages can potentially shut down the entire system. In the deployment, the gateway machine is connected to a 30-minute UPS and equipped with a power outage sensor. When a power outage occurs and lasts for longer than a few minutes, the email daemon sends out emergency notifications to alert the homeowner to the problem.

Since both remote home automation scenarios involve the use of emails, the Aladdin system is susceptible to email server unavailability and email delivery latency. For example, when the "ILOVEYOU" computer virus/worm and its variants plagued the email systems around the globe, the homeowner lost remote access to his home devices for many hours.

6. Related Work

The concept of soft-states has been widely used in network protocols and distributed systems. But it typically appears in the context of a specific protocol. Our contribution in Aladdin is to identify the importance of soft-states for building robust distributed applications in a heterogeneous and dynamic environment, and to build a soft-state store as a shared infrastructure. The APIs for interacting with the store have been designed to be sufficiently flexible to encompass the existing uses of soft-states in various areas.

In network protocols, soft-states are typically propagated by each node to one or more groups that the node belongs to. Examples include resource reservation protocols such as the receiver-initiated RSVP [ZDE+93] and the sender-initiated YESSIR [PS99], where the soft-states are path states and reservation states. In multicast protocols such as PIM [DEF+96] and SRM [FJM+95], the soft-states are group membership and topology updates. Other examples include periodic route advertisement in routing protocols [DEF+96], directory updates in DNS and in the MBONE Session Announcement Protocol [H96], and data summaries/statistics in transport protocols such as Real-time Transport Protocol [SCF+96] and Soft-State Transport Protocol [RM99]. More generally, we find that most self-stabilizing network protocols essentially use soft-states [AP95].

In distributed systems, soft-states are typically used where strong consistency is not required, but eventual consistency with high availability of information is sufficient. The use of soft-state control information has enabled fault tolerance and high availability in web servers [SLR98], distributed resource management [CRS98], and cluster-based network servers [FGC+97]. Heartbeats or "I am alive" messages have been widely used in detecting failures of remote nodes [HK93][V+98][RMH98]. DCOM pinging [BK98] is a soft-state-based, distributed garbage collection mechanism for server objects to reclaim reference counts associated with abnormally terminated clients.

Jini [E99] provides a set of specifications for services and conventions built atop Java Remote Method Invocation (RMI). Both Jini and Aladdin identified service discovery and lookup, leasing (or soft-states), and events as the most critical infrastructure components in a dynamic distributed system. They differ in the approach of dealing with heterogeneity. Jini focuses on the object-oriented API layer: all network entities must interact with the Jini infrastructure through Java interfaces by either hosting a local Java Virtual Machine or bridging through a Java communication proxy. In contrast, Aladdin specifies the wire protocols for interacting with the infrastructure. Any network entity that is capable of generating messages conforming to the wire protocols can participate in the Aladdin system as a first-class citizen without bridging through a communication proxy. Higher-level object-oriented APIs are provided only for convenience; they simplify the programming on PCs. In another aspect, both Jini and Aladdin allow arbitrary communication protocols between the clients and the services. The difference is that, in Jini, the lookup service entries are Java proxies that provide the interfaces seen by the clients. Any non-Java RMI communication protocols must also be hidden behind these interfaces. In Aladdin, the NBLS entries are opaque address strings. Aladdin is agnostic about the address encoding/decoding process and how a client makes use of an address. This design can facilitate the addition of new protocols and accommodate both RPC and non-RPC style communications. To simplify DCOM programming, Aladdin does provide helper functions for marshaling and unmarshaling DCOM interface pointers. Finally, the concepts of Aladdin Device Adapter and persistent soft-states, the

architecture for remote home automation, and the system management for providing reliable services are orthogonal to Jini and can be applied there as well.

Whether they use soft-states or hard-states, extant lookup services [BvST99, CZH99+, E99] mostly focus on the issue of scalability to millions of users and many groups of lookup services. Being focused on home networks, our lookup services do not emphasize hierarchy or federation. Providing a lightweight solution to support friendly naming of devices has been the key requirement. The NBLS wire protocol uses similar semantics to an early version of the Simple Service Discovery Protocol (SSDP) [G+99], which is part of the Universal Plug and Play (UPnP) initiative. In general, Aladdin provides higher level services for home networking and can build on top of UPnP. Since eventual consistency based on soft-state refreshes is sufficient for most home networking applications, we did not adopt active replication-based solutions [M96] that use totally ordered, reliable multicast to achieve virtual synchrony across replicated lookup services, thereby guaranteeing strong consistency.

Scalability of the soft-state networking protocols necessitates dealing with the message overhead introduced by the periodic refresh of information. This has led to several refinements: batching and compression of refreshes between nodes [WTZ99], adaptation of refresh frequency (either by negotiation between senders and receivers or independently by senders and receivers by monitoring control traffic) [SEF+97], use of acknowledgements for refreshes [RM99], and multistage decrease of frequency of refresh [PH97]. Our toolkit accommodates most of these refinements. It uses acknowledgements albeit only for low-frequency refreshes which benefit the most from acknowledgements in the presence of message loss.

Extant research projects on smart home environments are largely focusing on issues orthogonal to dependability and extensibility. Research results from those projects can be incorporated into the Aladdin environment to further enhance overall user experience. The Adaptive House project [M98] aims at developing a home control system that observes the lifestyle of the inhabitants and learns to anticipate their needs based on neural network techniques. The goal of the EasyLiving project [S+98] is to build an intelligent environment that maintains an awareness of its occupants through computer vision and facilitates unencumbered interactions among people and devices. The Aware Home project [Es99] focuses on analyzing and interpreting captured sensor streams from high-end multi-modal sensors to make the environment aware. The Smart Floor project [O00] has created a system for identifying people based on their footstep force profiles.

7. Summary

We have shown that the combination of soft-state store, publish/subscribe eventing, attribute-based lookup service, and name-based lookup service provides a powerful programming abstraction for building dependable and extensible home networking applications. To provide dependability, the soft-state store serves as a uniform framework for detecting the failures and unavailability of devices and objects, and for removing stale data to reclaim system resources as well as preventing misuse of such data. The publish/subscribe event system further allows higher-level adaptive, corrective, alerting, and cleanup actions to be performed in response to changes in the soft-state store. The late-binding feature provided by the lookup services makes the system more robust in the presence of failures, recoveries, and mobility. To provide extensibility, the attribute-based and name-based lookup services allow devices and objects to join dynamically and be available to future clients. Event subscriptions based on soft-state types, instead of individual variables, further allow triggering existing clients to include the new devices and objects. We have described two remote home automation applications and summarized our experiences from the actual usage of these applications.

Acknowledgement

We thank Kuansan Wang (Microsoft Research) for providing the language parser and speech recognizer.

References

[AP95] A. Arora and D. Poduska, “A Timing-based Schema for Stabilizing Information Exchange in Networks,” in *Proc. Int. Conf. on Computer Networks*, 1995.

[B98] D. Box, Essential COM, Addison Wesley, 1998.

[BvST99] G. Ballatijn, M. van Steen, A. Tannebaum, “Exploting location awareness for scalable location-independent object IDs,” in *Proc. Fifth Annual ASCI Conf.*, pp. 321-328, 1999.

[BK98] N. Brown and C. Kindel, Distributed Component Object Model Protocol -- DCOM/1.0, 1998.

[CRS98] K. Chandy, A. Rifkin, and E. Schooler, “Using announce-listen with global events to develop distributed control systems,” *Concurrency: Practice and Experience*, pp. 1021-1027, 1998.

[C88] D. D. Clark, “The design philosophy of the DARPA internet protocols, in *Proc. ACM SIGCOMM*, 1988.

[CM] CM11A Programming Specification, ftp://ftp.x10-beta.com/ftp/protocol.txt.

[CZH99+] S. Czerwinski, B. Zhao, T. Hodes, et al, "An architecture for secure service discovery service", in *Proc. ACM/IEEE MobiCom*, pp. 24-35, Aug. 1999.

[DEF+96] S. Deering, D. Estrin, D. Farinacci, et al, "PIM architecture for wide-area multicast routing," *IEEE/ACM Trans. on Networking*, Vol. 4, No. 2, pp. 153-162, Apr 1996.

[E99] W. K. Edwards, "Core Jini", Prentice-Hall Inc., 1999.

[Es99] I. Essa, "Ubiquitous Sening for Smart and Aware Environments: Technologies towards the building of an Aware Home," *Position paper for the DARPA/NSF/NIST Workshop on Smart Environments*, July 1999.

[FJM+95] S. Floyd, V. Jacobson, S. McCanne, et al., "A reliable multicast framework for light-weight sessions and application level framing," *Proc. SIGCOMM*, Sept. 1995.

[FGC+97] A. Fox, S. Gribble, Y. Chawathe, et al, "Cluster-based scalable network services," in *Proc. SOSP*, pp. 78-91, 1997.

[G+99] Y. Y. Goland et al., "Simple Service Discovery Protocol," *IETF Internet Draft*, http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt, Oct. 1999.

[H96] M. Handley, "SAP: Session Announcement Protocol," Internet Draft, Internet Engineering Task Force, Nov. 19, 1996

[H98] The Home Phoneline Networking Alliance, "Simple, High-Speed Ethernet Technology for the Home," http://www.homepna.org/docs/wp1.pdf, June 1998.

[HF95] D. Haught and J. Ferguson, "Jet Database Engine Programmer's Guide," Microsoft Press, 1995.

[HK93] Y. Huang and C. Kintala, "Software Implemented Fault Tolerance: Technologies and Experience," in *Proc. FTCS*, pp. 2-9, 1993.

[M96] S. Maffeis, "A Fault-Tolerant CORBA Name Server," in *Proc. SRDS*, pp.188-197, Oct. 1996.

[M98] M. C. Mozer, "The Neural Network House: An Environment that adapts to its inhabitants," in *Proc. AAAI Spring Symp. on Intelligent Environments*, pp. 110-114, 1998.

[O00] R. J. Orr and G. D. Abowd, "The Smart Floor: A Mechanism for Natural User Identification and Tracking," *Technical Report GIT-GVU-00-02*, Georgia Tech., Jan. 2000.

[PH97] P. Pan and H. Schulzrinne, "Staged refresh timers for RSVP," in *Proc. GLOBECOM*, Vol. 3, pp. 1909-1913, 1997.

[PS99] P. Pan and H. Schulzrinne, "YESSIR: a simple reservation mechanism for the Internet," *Computer Communication Review*, Vol. 29, No. 2, pp. 89-101, 1999.

[RM99] S. Raman and S. McCanne, "A model, analysis, and protocol framework for soft state-based communication," in *Proc. SIGCOMM*, pp. 15-25, 1999.

[S98] Silent Servant Home Control Inc., Automated Home Control, 1998.

[SCF+96] H. Schulzrinne, S. Cassner, R. Frederick, et al, "RTP: A transport protocol for real-time applications", *IETF Audio Visual Transport Working Group, RFC-1889*, Jan. 1996.

[S+98] S. Shafer, J. Krumm, B. Brumitt, B. Meyers, M. Czerwinski, and D. Robbins, "The New EasyLiving Project at Microsoft Research," *DARPA/NIST Workshop on Smart Spaces*, July 1998.

[SEF+97] P. Sharma, D. Estrin, S. Floyd, et al, "Scalable timers for soft state protocols," in *Proc. INFOCOM*, pp. 222-229, 1997.

[SLR98] A. Singhai, S.-B. Lim, and S. R. Radia, "The SunSCALR Framework for Internet Servers," in *Proc. FTCS*, pp.108-117, 1998.

[RMH98] R. van Renesse, Y. Minsky, and M. Hayden, "A Gossip-Based Failure Detection Service," in *Proc. Middleware*, 1998.

[V+98] W. Vogels et al., "The Design and Architecture of the Microsoft Cluster Service," in *Proc. FTCS*, pp. 422-431, 1998.

[WRA+00] Y. M. Wang, W. Russell, A. Arora, et al., "Towards Dependable Home Networking: An Experience Report," in *Proc. IEEE Int. Conf. on Dependable Systems and Networks (formerly FTCS)*, June 2000.

[WTZ99] L. Wang, A. Terzis, and L. Zhang, "New proposal for RSVP refreshes," in *Proc. 7th Int. Conf. on Network Protocols (ICNP'99)*, pp. 163-172, 1999.

[ZDE+93] L. Zhang, S. Deering, D. Estrin, et al, "RSVP: a new resource ReSerVation Protocol ," *IEEE Network*, Vol. 7, No. 5, pp. 8-18, Sept. 1993.