Evaluating Rapide Using the FRODO Service Discovery Protocol

M.D. Speelziek

June 27, 2006

University of Twente

Faculty: *Department of Electrical Engineering, Mathematics and Computer Science*
Area: *Distributed and Embedded Systems*

Graduation Committee

Ir. J. Scholten
Ir. P.G. Jansen
V. Sundramoorthy

Evaluating Rapide Using the FRODO Service Discovery Protocol

Summary

The future will see the advent of home automation systems: all kinds of home appliances will be connected to a network that enables them to communicate with each other. In such a home automation system a mechanism is needed that integrates the individual devices into a single system. Service Discovery protocols provide this mechanism, but the existing protocols are not well suited for use in a home automation environment. Therefore a new Service Discovery protocol has been developed within the At Home Anywhere project: FRODO. This thesis presents the development of a model of FRODO. For this model an Executable Architectural Description Language, named Rapide will be used, only Rapide is not widely used and therefore little is known about the strengths and weaknesses of this language. That is why modelling FRODO will serve two purposes: creating a simulation environment that will be used to validate the protocol specification of FRODO, and evaluating Rapide. The created simulation environment will be used to simulate a number of scenarios that compare the characteristics of FRODO to those of current Service Discovery protocols.

Samenvatting

In de toekomst zal er steeds vaker gebruik gemaakt worden van Home Automation systemen: allerlei verschillende apparaten in huis worden met elkaar verbonden in een netwerk, zodat ze met elkaar kunnen communiceren. In zo'n netwerk is een mechanisme nodig om de apparaten met elkaar laten samenwerken. Service Discovery protocollen bieden zo'n mechanisme. De huidige Service Discovery protocollen zijn echter ontwikkeld om ingezet te worden in een kantooromgeving. Daarom is binnen het At Home Anywhere project een nieuw Service Discovery protocol ontwikkeld, speciaal bedoelt voor gebruik in Home Automation systemen: FRODO. Dit verslag beschrijft de ontwikkeling van een model van FRODO. Er wordt daarbij gebruik gemaakt van Rapide, een Executable Architectural Description Language. Rapide is echter een weinig gebruikte taal en er is daarom ook maar weinig bekend over zijn sterke en zwakke punten. Het modeleren van FRODO zal daarom gebruikt worden voor twee doeleinden: het creeren van een simulatie-omgeving waarmee de specificatie van FRODO gevalideerd zal worden, en het evalueren van Rapide. Verder zal de simulatie-omgeving gebruikt worden om een aantal scenario's te simuleren, om zo de karakteristieken van FRODO te vergelijken met bestaande Service Discovery protocollen.

Contents

Chapter 1

Introduction

1.1 Motivation

These days, TVs, Hi-Fi equipment, micro-waves, and washing machines all contain embedded computers. In the near future this list may be extended to include refrigerators, light switches, temperature sensors and many other devices. The idea is to connect all these devices to a single network, so that they can be controlled from any anywhere in the house, or even from anywhere in the world by using an Internet connection. The At Home Anywhere (@HA) research project [AHA03] tries to create such an environment. One of the goals of the project is to develop a real-time network protocol that supports the different classes of traffic the appliances generate: entertainment, control and information.

But a home automation system needs more than just a way for devices to communicate with each other. A mechanism is needed that enables one device on a network to discover the services provided by other devices on the same network. A Service Discovery Protocol (SDP) provides such a mechanism.

Several Service Discovery Protocols have already been developed, but none of these have been developed for use in a home automation environment: they don't allow the participation of resource-lean devices, like light switches or temperature sensors, or they need the assistance of an administrator to set up the system. Therefore a new SDP has been designed as part of the @Home Anywhere project. This new protocol is called FRODO and it has been optimized for use in home automation.

But, when creating a new network protocol you need a way to validate the design. One way of doing this, is to create an executable model that can be used to simulate different scenarios. At the National Institute of

Standards and Technology (NIST) models have been created of the existing service discovery protocols Jini and Universal Plug and Play (UPnP). The experiments [DME02a, DME02b] conducted on these models were used to compare the performance of Jini and UPnP. A similar model of FRODO can be used to validate the protocol specification. It will also make it possible to compare the performance of the protocol to that of existing protocols.

To model Jini and UPnP, NIST has used the language and toolset developed as part of the Stanford Rapide project [RAP98]. Rapide is an event-based executable architecture definition language that has been developed to model distributed time-sensitive system, which a network protocol basically is.

The problem with the Rapide language and toolset is that they are not widely used, and therefore little is known about its strengths and weaknesses. In order to find these strengths and weaknesses, you need more than the examples in the supplied documentation and tutorials: you need to use it on a non-trivial problem. Creating a model of FRODO would be such a problem.

This thesis will focus on creating a model of the FRODO service discovery protocol. This model will be used to repeat the experiments that NIST has conducted on their models of Jini and UPnP. The purpose of these experiments is twofold: (1) it will prove the correctness of the model and (2) it will reveal the properties and performance of FRODO. The conducted experiments will be discussed in chapters 7 and 8, while the model will be presented in chapter 6. Chapters 2 to 5 contain a summary of some important concepts used in service discovery protocols, an introduction to Rapide, a discussion of the models made by NIST and an introduction of FRODO respectively. Finally, the conclusions can be found in chapter 9.

1.2 Problem Statement

Create a model of the FRODO service discovery protocol using the Rapide language and toolset. Use this model to (1) Validate the design of FRODO and (2) Evaluate the use of Rapide as a modeling and simulation tool for network protocols.

Chapter 2

Service Discovery

This chapter will give an introduction to the terminology and concepts used in Service Discovery Protocols. NIST describes a Service Discovery Protocol as follows:

> Discovery protocols enable software components to find each other on a network and to determine if discovered components match their requirements. Further, discovery protocols include techniques to detect changes in component availability, and to maintain, within some time bounds, a consistent view of components in a network [DM01].

A number of existing protocols have been designed to meet these requirements: Jini [JINI03], UPnP [UPNP00], SLP [RFC2608], Salutation [SAL99], Bluetooth SDP [BLUE99].

2.1 Generic Terminology

The existing SDPs use their own names for the used components and concepts, although most of the components and concepts are common to all service discovery protocols. To help in comparing the protocols, NIST has developed a consistent terminology for the components of a SDP [DM01]. This terminology will also be used in this thesis.

Service Provider (SP) A software or hardware component providing some kind of service (e.g. a printer).

Service Description (SD) A record describing the identity, type and attributes of a *Service Provider* (e.g. the printer on floor three, that is capable of printing in color on A3 sized paper).

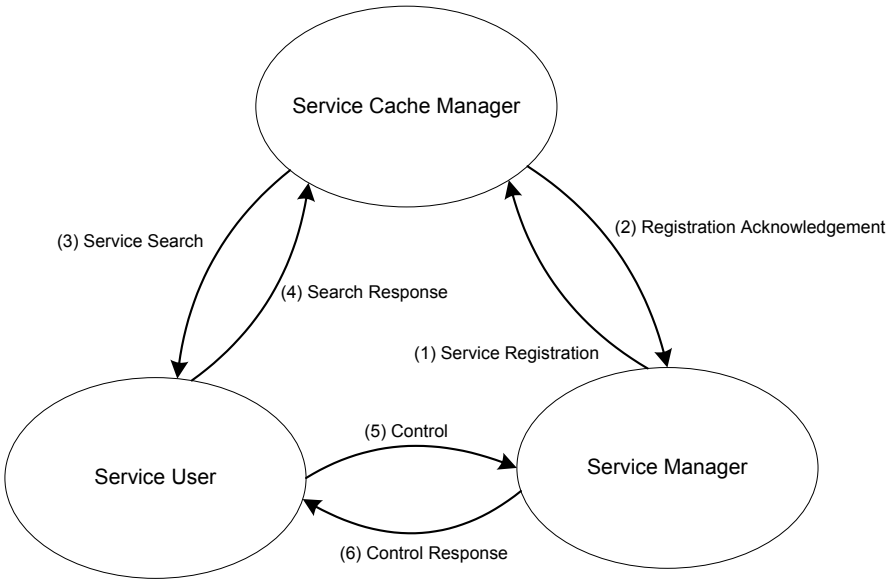


Figure 2.1: Relations between components.

Service Manager (SM) A component that a database of *Service Descriptions*. Most implementations will use one *SM* per device. That *SM* will contain *SDs* of all *SPs* located on the same device (e.g. for a printer it would probably contain one *SD*, but for a multifunction device it would contain *SDs* for the print, the scan, the copy and the fax functionality).

Service User (SU) A component that will use the protocol to search for a *Service Provider*.

Service Cache Manager (SCM) A component that maintains a central database of *Service Descriptions* of all *Service Providers* on a network. This component is not used in all service discovery protocols.

Figure 2.1 shows how the different components are related. After coming online, the SM will register its SDs with the SCM. When the SU needs to use a service, it sends a service search to the SCM. The SCM will respond to this search with the matching SDs. After that, the SU can communicate directly with a SM to control the SP.

A mapping between these generic names and the names used in the Jini and UPnP protocols can be found in table 2.1.

Generic Name	Jini	UPnP
Service Provider	Service	Device or Service
Service Description	Service Item	Device Description or Service Description
Service Manager	Service or Device Proxy	Root Device
Service User	Client	Control Point
Service Cache Manager	Lookup Service	not applicable

Table 2.1: Mapping to generic terminology.

2.2 Service Discovery Architectures

Most SDPs use one of two underlying architectures: two-party and three-party [DME02a].

A two-party architecture consists of two major components: SMs and SUs. When a SU wants to use a service it has to broadcast a message over the network with the required service attributes. Each SM that maintains a SD that matches the search specifications will respond to the search. A SM can also periodically announce the presence of its services. An example of a protocol using this architecture is UPnP.

A three-party architecture uses an additional component: the SCM. Instead of broadcasting a service search to all SMs, a SU will contact a SCM with its search request. The SCM will return the matching SDs from its database.

An advantage for a protocol using the three-party architecture: it requires fewer broadcasts than a similar protocol using the two-party architecture, because it does not need broadcast messages for each search. The disadvantage of the three-party architecture is that it introduces a single point of failure: the SCM. Most protocols using a SCM will therefore provide a fall-back mechanism to use when no SCM is available. An example of a protocol using the three-party architecture is Jini.

2.3 Service Discovery Concepts

A number of mechanisms commonly used in SDPs are polling, notification and leasing. Polling and notification are both used to propagate updated SDs to interested SUs and SCMs. The definition of polling and notification used by NIST:

In polling, a SU periodically queries the SCM or the SMs to

> obtain an up to date SD of a SP that was previously discovered. [DME02a]
>
> In notification, immediately after an update occurs, a SM sends events that announce a SD has changed. [DME02a]

A definition of leasing is given by the following:

> Leasing is a well-known design tool in building resilient distributed software. The parties in a lease relationship, as in real life, are the grantor of the lease (the lessor) and the holder of the lease (the lessee). In general, the lessor makes some resource available for a specified amount of time, and the lessee uses the resource to some advantage during this time. The duration can be determined in a number of ways and a lease may be renewed if this is agreeable to both parties. [GOL02]

One use of leasing in SDPs is to keep a consistent view of the available SPs in the network, by limiting the lifetime of cached SDs in SUs and SCMS.

Chapter 3

Introduction to Rapide

Rapide is a language for defining and executing models of system architectures.

It has been used to model the Jini and Universal Plug and Play service discovery protocols by the National Institute of Standards and Technology (see chapter 4). And it will be used to model FRODO, a new service protocol (see chapter 6). The models will be used to compare the performance of the new service discovery protocol to that of the existing protocols.

This chapter will give and overview of the Rapide language (section 3.1 and the provided tools (section 3.2). More information can be found on the project website [RAP98].

3.1 Rapide Language

Rapide is structured as a set of languages called the Rapide language framework. This framework consists of the following languages:

- Types Language.
- Pattern Language.
- Executable Module Language.
- Architecture Language.
- Constraint Language.

Together these languages can be used define models of system architectures. How to use these languages will be demonstrated by an example [RAP97].

This example uses the Types, Pattern, Executable Module and Architecture languages to model a producer/consumer architecture:

```
type Producer is interface action out Emit( n : Integer ); end Producer;
type Consumer is interface action in Source( n : Integer ); end Consumer;

module ProducerModule( min, max : Integer ) return Producer is
   function Compute( n : Integer ) return Integer is
      begin
         return n + 1;
      end function Compute;
initial
   Emit(min);
parallel
   when (?x in Integer) Emit(?x) where ?x < max do
      Emit(Compute(?x));
   end when;
end module ProducerModule;

module ConsumerModule() return Consumer is
   function Use( n : Integer ) is
      begin
         null;
      end function Compute;
parallel
   when (?y in Integer) Source(?y) do
      Use(?y);
   end when;
end module ConsumerModule;

architecture ProdCons() is
   Prod : Producer is ProducerModule(1, 4);
   Cons : Consumer is ConsumerModule();
connect
   (?n in Integer)
   Prod.Emit(?n) |> Cons.Source(?n);
end architecture ProdCons;
```

This simple example shows an architecture *ProdCons* that consists of two components: a *Producer* object and a *Consumer* object (called *Prod* and *Cons* respectively). The interfaces *Producer* and *Consumer* define a producer as being able to generate an *Emit* event and a consumer as being able to observe *Source* events. The module *ProducerModule* implements the interface *Producer* and simply generates *Emit* events up to some maximum. The module *ConsumerModule* implements the interface *Producer* and just observes *Source* events.

In the connect part of the architecture the *Emit* events of the producer is connected to the *Source* events of the consumer. Whenever the connection is triggered by an *Emit* event, it executes by generating a *Source* event with the same integer argument.

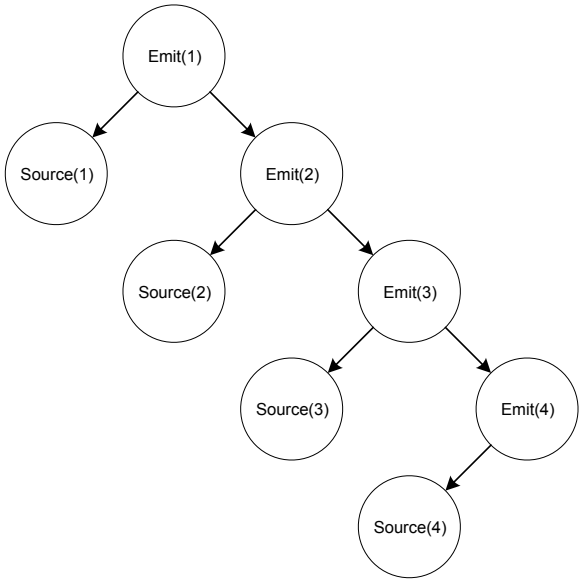


Figure 3.1: Poset generated by example.

When this architecture is executed, the poset (partially ordered event set) in figure 3.1 is generated. The poset contains the set of events (the ellipses in the figure) that occurred during the execution together with the causal relationships (the arrows in the figure) between events. For the example this means that the *Emit* events are causally related because they are generated in sequence by the module *ProducerModule*. The *Source* events are causally related to the *Emit* event with the same argument because of the connection defined in the architecture.

3.2 Rapide Toolset

Besides the Rapide language, the Rapide project also provides a number of tools. These tools are divided into two groups: the command line tools to translate models written in the Rapide language to an executable format and the graphical tools that can be used to analyze the data that result from running the generated executables.

3.2.1 Command line tools

The most important tool from the Rapide toolset is the compiler that translates Rapide source files into executables. This compiler is executed from

the command line and has a syntax similar to that of a C compiler. For example, to compile the example from the previous section, the following commands have to be executed. First the compilation environment has to be created:

```
$ r.mklib
```

This creates a hidden directory containing a number files used by the compiler. After the environment has been set up the Rapide model can be compiled with the following command (assuming the file which contains the model source code is called `example.rpd`):

```
$ rpdc -M ProdCons -o example example.rpd
```

If the source file is free of errors, then this will generate the executable file `example`, otherwise the compilation process will abort with a description of the error. The generated executable can be run with the following command:

```
$ .\example
```

This will generate the file `example.log` containing the poset data.

3.2.2 Graphical tools

The Rapide toolset contains a number of tools that can be used to analyze the poset data generated by executing a Rapide model:

Pov The poset viewer: a tool for graphically browsing the posets generated by Rapide executables. Figure 3.1 was generated using this tool.

Raptor This tool provides a way to visualize the flow of events by using the poset data and a graphical representation of the modeled architecture to create an animation. It is only useful for small architectures, and it does not provide many options to customize the animation.

Raparch A tool for editing Rapide architectures graphically. It is used to draw the graphical representation of an architecture for use with the Raptor animation tool.

The *Raptor* and *Raparch* tools seem far from finished and are therefore not usable: they are very unstable and have an awkward user interface. The poset viewer can be useful, but when a model generates a lot of events, then it becomes difficult to distinguish between them.

3.2.3 Toolset installation

Installing Rapide on a GNU/Linux system proved to be more difficult than expected. The reason for this is the fact that the Rapide toolset is provided as a package containing binaries using the a.out object file format. This package was created in 1997 and contains the tools as pre-compiled binaries and some of the libraries required by the tools, also in binary format. Some other libraries required by the tools must be provided by the host system. Since the creation of the Rapide package, GNU/Linux systems have migrated to a different binary file format. For a long time it has been possible to use the old binary file format on migrated systems, but more and more modern Linux distributions are now dropping support for the old format.

After an unsuccesful attempt to install the Rapide toolset on a Debian 3.1 system, the toolset was installed on another computer running an old version of the Redhat GNU/Linux distribution (version 6.2). Installation instructions for the toolset on this distribution can be found in appendix A. On the Redhat 6.2 system all the provided tools are working.

It should be possible to use a virtual machine to run the Rapide tools instead of dedicating a whole computer to it, as long as an older distribution is used as the guest operating system.

Chapter 4

Rapide Models of Jini and UPnP

Using the Rapide language the National Institute of Standards and Technology (NIST) has modeled the Jini and Universal Plug and Play service discovery protocols [DM02a, DM02b]. These models will be used as a basis for the model of FRODO and to compare the performance of FRODO to that of existing service discovery protocols.

Each of the models is contained in a single large Rapide source file (> 10000 lines of code), containing ten sections of code. This chapter will give an overview of the Rapide models of Jini and UPnP by describing the contents of each section and showing how the components defined in the different sections combine to form the actual model. Because the two models use a similar structure and file layout, the description given in this chapter applies to both models, although the examples given are from the Jini model.

4.1 Section Zero: Utility Functions

The first section of the source files contains the definition of some global utility functions used throughout the rest of the source. The most important of these functions is *ConsoleWrite*. This function is used to write messages to the standard console output.

4.2 Section One: Global Types and Specifications

This section contains model specific global types, variables and functions: for example the functions to compute processing delays, and the variables

containing node and network link states.

4.3 Section Two: Service Interface Definitions

Section two contains service interface definitions. These represent the messages that can be sent by the service discovery protocol. The messages are grouped by function. The following example shows the definition of one of the service interface definitions from the Jini model:

```
TYPE TCP_2_STEP_SEQUENCE IS INTERFACE
ACTION
    OUT
        Connect_Request (SourceID: IP_Address;
                         SCM_ID: SCM_Service_ID;
                         PV : ProtocolVersion);
    IN
        API_Response    (ServiceProxy: JavaObject;
                         SourceID: IP_Address;
                         SCM_ID: SCM_Service_ID);
END;
```

This defines the interface *TCP_2_STEP_SEQUENCE*, which contains outgoing action *Connect_Request* and the incoming action *API_Response*. These messages are part of Jini's lazy discovery.

4.4 Section Three and Four: Service Discovery Subcomponents

Sections three and four of the source files contain the definitions of the subcomponents that implement the protocol behavior concerning the actual discovery of services. For Jini this means the aggressive, lazy and directed modes of discovery. These subcomponents are defined as interfaces, but these interfaces also contain the behavior by using a Rapide feature that allows you to include a behavior definition in an interface definition. The following example shows the behavior of Jini when it receives a connection request:

```
(?SourceID: IP_Address; ?HostName: SCM_Service_ID; ?PV: ProtocolVersion)
LZ_CONN_RESP.Connect_Request (?SourceID, ?HostName, ?PV)
||>
IF $(NodeUp[$MyServiceID])
THEN
  IF $Responding THEN
     LZ_CONN_RESP.API_Response("Java object",
                               ?SourceID,
```

```
                                    $MyServiceID)
            in SimpDel ($MyServiceID, 0);
      END IF;
    END IF;;
```

The first two lines define a pattern that matches a received *ConnectRequest* event. The other lines use the Rapide Executable sub-language to define the correct response to the request. Most events will follow a similar pattern.

4.5 Section Five: Cache Sub-Components

Section five contains the interface definition of the caches used in the protocol. Jini, for example, uses a cache of Service Descriptions on the Service Managers, a cache of registered Service Descriptions on the Service Cache Managers and a cache containing information about active notification requests on the Service Users.

The cache interfaces also contains the protocol behavior concerning the maintenance of the caches. Like in sections three and four, this uses the feature of Rapide to define interface behavior within the interface definition itself.

Because of the code for the manipulation of the cache structures, the event handlers in this section are a bit larger than those in sections three and four, but their structure is basically the same. Due to the size it would take, an example will not be given here.

4.6 Section Six and Seven: SU, SM and SCM Nodes

Sections six and seven contain the definitions of the SU, SM and SCM interfaces and modules. These connect the sub-components defined in sections three, four and five, to define a SU node, a SM node and a SCM node.

Section six also contains the *Node* interface. This interface defines a kind of super-interface of the SU, SM and SCM interfaces.

4.7 Section Eight: Network Definitions

Section eight contains the modules that define the network behavior. The behavior is split into a module for unicast messages and a number of modules for different kinds of multicast messages. A message sent over the network

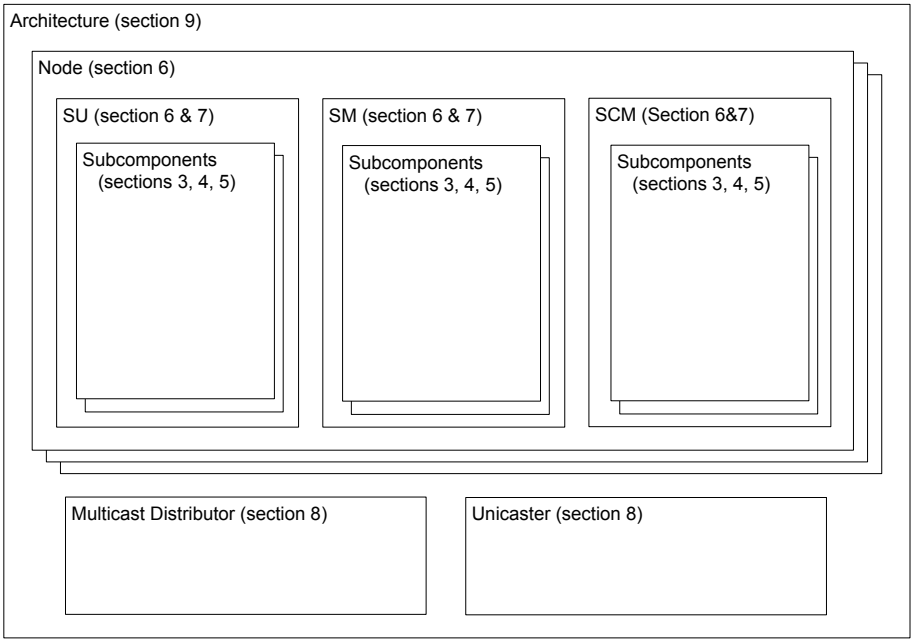


Figure 4.1: Structure of Jini and UPnP models.

can be a TCP message: using a reliable, connection oriented service, with a variable delay, or a UDP message: using an unreliable datagram service, delivered with a fixed delay. The network modules also implement random message loss and link failures.

4.8 Section Nine: Architecture

Section nine contains the architecture, which connects the different parts of the model. Figure 4.1 shows how the parts are connected, together with the section that contains the definition of the part.

The architecture connects a configurable number of the SU, SM and SCM nodes defined in section seven, to the network modules defined in section eight. Each type of node (SU, SM, or SCM) is composed of some of the components from sections three, four and five.

Not shown in the figure are the modules that are used to set up and control the different scenarios simulated using the model. These modules are also defined in section nine.

4.9 Conclusion

Examining the models of Jini and UPnP has provided the knowledge to use them as a basis for a similar model of FRODO. This model of FRODO will be presented in chapter 6.

Chapter 5

Introduction to FRODO

This chapter introduces the FRODO service discovery protocol. FRODO is the service discovery protocol for the At Home Anywhere project. For a more detailed overview of the protocol see [SUN03] (this paper still uses the old name SDP@HA).

The new service discovery protocol addresses the following issues that are lacking in existing protocols [SUN03]:

Participation of resource lean devices Current technologies are suited only for powerful and expensive devices. The protocol for the @HA network should be able to support small and cheap devices.

Delegation of work load In order to support resource lean devices, workload has to be delegated to more powerful devices.

Robust architecture Because in a home environment a system administrator is not available, the protocol should require little to no manual configuration, and it should be able to recover from network errors automatically.

5.1 Device Classes

FRODO divides supported devices into the following three classes [SUN03]:

3C (3+ cent) device Very simple devices with custom hardware only capable of providing the basic Service Manager functions.

3D (3+ dollar) device Medium complex devices, controlled by a microcontroller containing a few kilobytes of memory. Capable of supporting Service User and Service Manager functions.

300D (300+ dollar) device Complex devices controlled by an embedded computer containing a powerful processor (> 200 MHz) and at least 1 megabyte of memory.

5.2 Design

In this section a quick overview of FRODO will be given. Some of the unique features of the protocol will be highlighted. For a complete description of the protocol see [SUN05] and the specification flowcharts in appendix B.

5.2.1 Central Election

FRODO uses a client/server architecture with one Service Cache Manager. This SCM is called the *Central* and it is dynamically assigned. Using a SCM reduces the amount of expensive broadcast messages a 3D device has to process, because a service search can be directed to the SCM, instead of requiring a broadcast.

All 300D devices will participate in a process is called leader election. The most capable device (in terms of processing power and memory size) will be the winner of this election and it will become *Central*. The new *Central* will appoint the runner-up as its *Backup*. When, for any reason, the *Central* become unreachable, the *Backup* will take its place and appoint a new *Backup*. If the *Backup* is unable to take over from an unreachable *Central*, then a new election is started.

In the absence of any 300D devices on a network, or if a *Central* is not elected yet, the protocol falls back to a peer-to-peer search mode.

The leader election process makes it unnecessary to have an administrator configure one or more devices as SCM. It also makes the protocol robust, because it removes the SCM as a single point of failure.

5.2.2 Device and Service Registration

After a *Central* has been elected, it will broadcast a message telling all devices that there is a new *Central*. All SM devices will respond to this broadcast by registering their Service Descriptions.

5.2.3 Service Search and Subscription

When a Service User requires a particular service it sends a service search request to the SCM, containing the required service type and attributes. The

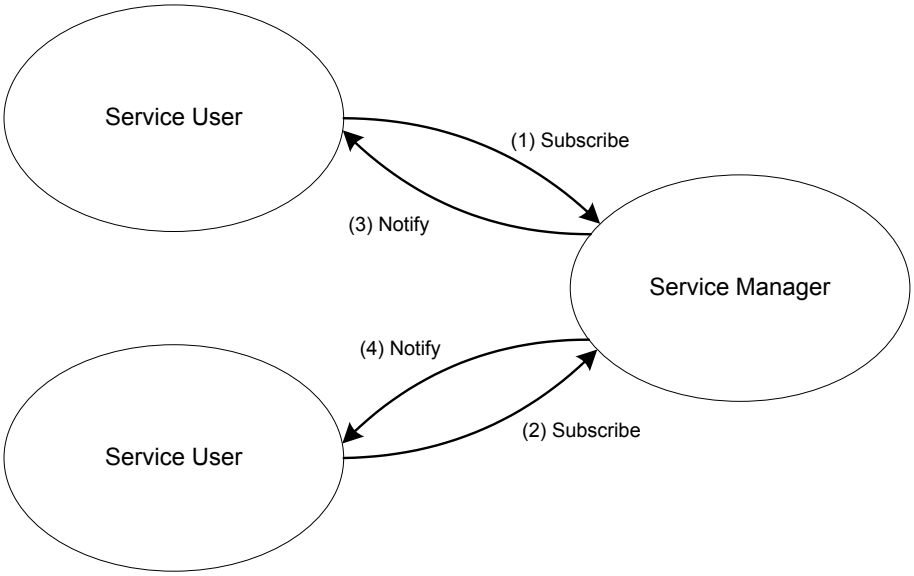


Figure 5.1: 2-way subscription.

SCM will respond with a list of suitable Service Providers and their Service Managers. The SU can communicate directly with the returned SMs.

If a SU whishes to use a Service Provider for a longer period of time, then it can take a subscription. When subscribed to a Service Provider, the Service User will receive a notification when the Service Description changes. To accommodate resource lean devices, FRODO provides two flavors of subscription:

2-way subscription The SM keeps a list of subscribers and sends notifications when a Service Description changes. Figure 5.1 shows an example, where two SUs subscribe to the same SM.

3-way subscription Used when a SM is not capable of storing or maintaining the list of subscriptions, or when it is unable to send notifications to every subscriber. In this case these tasks are delegated to the SCM. Figure 5.2 shows the same example as 5.1, but this time using 3-way subscription.

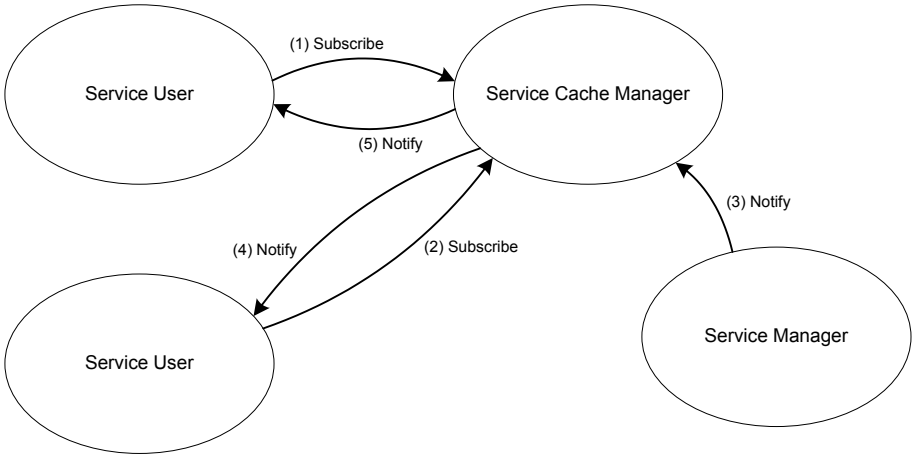


Figure 5.2: 3-way subscription.

5.3 Modeling FRODO

To evaluate the Rapide language and tools the FRODO protocol has been modeled using this language. The resulting model will also be used to validate the behavior of FRODO and to compare its performance to existing protocols.

The next few chapters will describe this model and the simulations that have been conducted on it.

Chapter 6

Rapide Model of FRODO

Using the Rapide language and toolset a model of the FRODO service discovery protocol has been created. The model is based on similar model made by the National Institute of Standards and Technology of the Jini and Universal Plug and Play service discovery protocols.

The model will help in understanding how FRODO behaves under different circumstances and it will be used to compare the performance of FRODO to that of the other service discovery protocols. The model will also serve as a basis for a prototype implementation, actually and modeling a network will reveal the strengths and weaknesses of Rapide in this application.

Because of the limited time available, only a subset of the protocol has been modeled. The parts of the protocol not modeled are the Central Election, the Central Backup and Peer-to-Peer Search. These parts are not needed for the correct operation of the parts of the protocol that have been modeled: Central Discovery, Service Registration, Service Search and Subscription. The missing parts can be added to the model later on. The modeled parts will also allow a first comparison between FRODO and the existing service discovery protocols Jini and UPnP.

At the time the model was created the protocol specification of FRODO still was a work in progress. This caused some extra work, because parts of the protocol changed after they had already been modeled, making it necessary to model them again. But the modeling process also revealed a number of flaws in the protocol and the protocol specification document and these could immediately be fixed.

The rest of this chapter will give an overview of the structure of the Rapide model of FRODO and highlight some of the changes in the simulation framework made by NIST.

6.1 Model Overview

FRODO has been modeled using the models of Jini and UPnP as a basis: it uses the same global structure as the NIST models. It also reuses a lot of the utility functions and the section layout.

The sections used in the FRODO model:

Section 0 Utility functions.

Section 1 Global model-specific types, functions and variables.

Section 2 Service interface definitions.

Section 3 Service Manager behavior (the *SM_Registration* interface).

Section 4 Service User behavior (the *SU_Search* interface).

Section 5 Service Cache Manager behavior (the *SCM_Cache* interface).

Section 6 Node interface definitions.

Section 7 Node module definitions.

Section 8 Network module definitions.

Section 9 Architecture definition.

Figure 6.1 shows a diagram of the main structural elements of the model and the way they fit together. It shows two types of nodes: a SU node and a SM node. These nodes implement the behavior of both 3D and 300D devices. A parameter is used to select the correct behavior.

6.2 Random Values

One of the changes made for the model of FRODO, is the way the *Random* function works. The *Random* function is an important utility function used by the Rapide models of Jini, UPnP and FRODO. It is used at the start of the simulation to determine the times at which important simulation events take place. Later in the simulation it is used to determine message loss, network delays and processing delays. The models made by NIST use the following function to calculate random numbers:

```
FUNCTION Random (Low, High : INTEGER) RETURN INTEGER IS
Number : INTEGER;
BEGIN
   IF High <= 10000
```

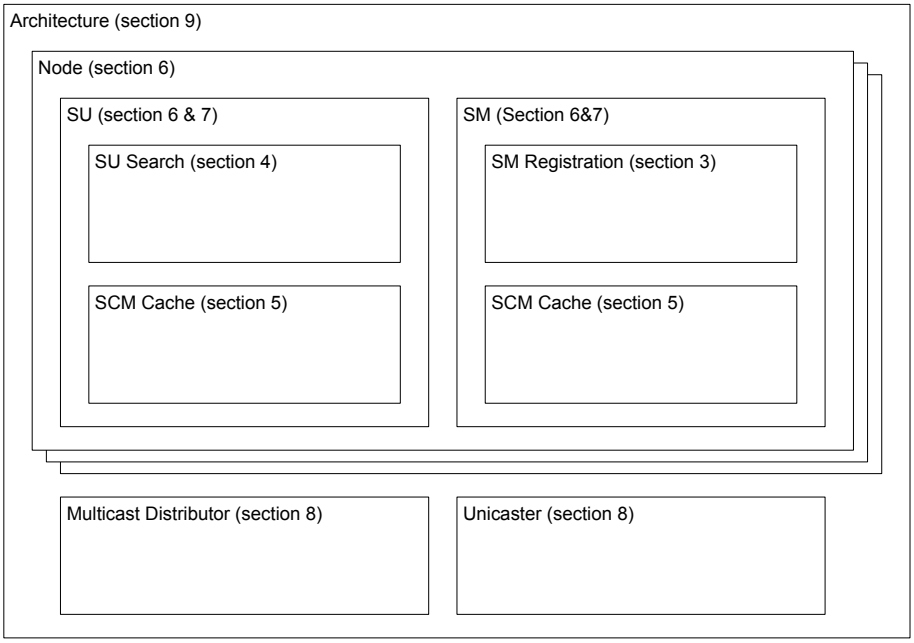


Figure 6.1: Model of FRODO.

```
   THEN
      RETURN (Low + (ABS (Number) MOD (High - Low + 1)));
   ELSIF High <= 100000
   THEN
      .
      .
      .
   END IF;
END;
```

This function will return a random value N in the range $Low <= N <= High$. The code snippet only shows the case where $High <= 10000$, but the other cases are similar to this case.

The *Random* function uses the uninitialized variable *Number* as its source of entropy. Each time the function is called it uses the value of *Number* that happens to be at the memory location occupied by *Number*. This value is different on each call to *Random*, because the same memory location is also used by other functions and will be overwritten when they are called. This seems to work, because it will indeed produce a different value each time the function *Random* is called.

However, a few trail simulation runs of FRODO showed exactly the same results for each run. It turned out that the first few calls to the *Random* function produced the same sequence of values in each run, and these values

were used to determine the simulation scenario. The NIST models did not show this behavior.

The reason for this difference in behavior can be found in the fact that the NIST models use a number of events to set up the simulation run, while the model of FRODO performs the setup procedure in one event. Because the events used in the setup are unrelated, Rapide processes them in a random order. This introduces some variation in sequence of statements executed prior to the calls to the *Random* function and this in turn brings variation to the sequence of generated values. It does not seem a good idea to rely only on this as a source of randomness, so the *Random* function in the FRODO model has been changed to the following:

```
RANDOM_POOL_SIZE: VAR INTEGER := 1000;

RandomPool : ARRAY[INTEGER] OF REF (INTEGER) IS
                (1..$RANDOM_POOL_SIZE,
                 DEFAULT IS REF_TO (INTEGER, 0));
RandomIndex: VAR INTEGER := 1;

FUNCTION Random (Low: INTEGER; High: INTEGER) RETURN INTEGER IS
   Number: VAR INTEGER;
BEGIN
   Number := $(RandomPool[$RandomIndex]);
   RandomIndex := $RandomIndex + 1;

   RETURN (Low + (ABS ($Number) MOD (High - Low + 1)));
END;
```

This function returns a value from the array *RandomPool*, using each value only once. *RandomPool* will be filled at the start of the simulation run with values read from the file *randompool*. Before each simulation run this file should be filled with values from a reliable source of entropy (on a UNIX system, this can be /dev/urandom).

With this new *Random* function the FRODO simulations did work as expected.

6.3 Data Collection

Another difference between the FRODO model and the NIST models is the way that simulation data is collected and processed.

The NIST models contain the module *CONSITENCY* that collects data during the simulation and stores it in variables. After the actual simulation run, the module processes the gathered data and writes the calculated results to the standard output. The processing phase of a simulation run takes a lot of time: up to half the simulation run time.

The model of FRODO simplifies the data processing by performing the necessary calculations in a separate program not written in Rapide. The Rapide model writes the relevant data to the standard output during the simulation run, together with all the debugging information. All information written to the standard output is written to a text file. After the simulation run, an AWK-script filters this text file and extracts the information needed to calculate the simulation results. The run time of this AWK-script is less than a second and the calculations are less complex than the *CONSISTENCY* module in the NIST models.

6.4 Conclusions

Working with the Rapide language to create a model of the FRODO service discovery protocol has served to give a much better understanding of the strengths and weaknesses of the language than studying existing models and tutorials can provide. This leads to the conclusion that the Rapide language is well suited to create models of network protocols. The focus on the parallel processing of events matches the nature op network protocols and the timing model of the language makes it easy to model the delays and timeouts needed in a network protocol. However, the executable sublanguage of Rapide is not as powerful as the rest of the language and its execution is slow. This is not a big problem for the modeling of the actual network protocol, because it does not contain a lot of executable code. It has, however, been the main reason to reorganize the data processing in the model of FRODO to remove it from the Rapide code.

The next two chapters will present the performance evaluation of FRODO made using the model described in this chapter.

Chapter 7

Simulation Setup

Using the model of FRODO presented in the previous chapter two scenarios have been simulated. The purpose of these simulations is to compare the performance of the Rapide model of FRODO against other service discovery protocols.

The protocols that will be used in the comparison are Jini and Universal Plug and Play. Performance data for these protocols is available from [DME02a]. This data is obtained from simulations using the models on which the model of FRODO is based: this makes it easy to follow the same scenario and to compare the results.

7.1 Scenarios

For the simulations of FRODO the scenarios from [DME02a] are used: these scenarios are used to measure the performance of the service discovery protocols in an environment with network interface failures.

The simulation scenarios will use a network of one Service Manager, one Service Cache Manager and five Service Users for each of the simulated protocols (except UPnP which does not use a SCM). The SM will register its service with the SCM and each SU will search for the registered service.

A simulation run, which is depicted in figure 7.1, lasts from $T = T_0$ to $T = D$. The period between $T = T_0$ and $T = Q$ is used to get the system in a consistent state by performing the following tasks (if appropriate for the protocol): SCM election, SCM discovery, Service registration and Service search. During this period no communication failures occur.

In the remaining time each node will experience an interface failure where either the incoming packets, outgoing packets, or both incoming and outgo-

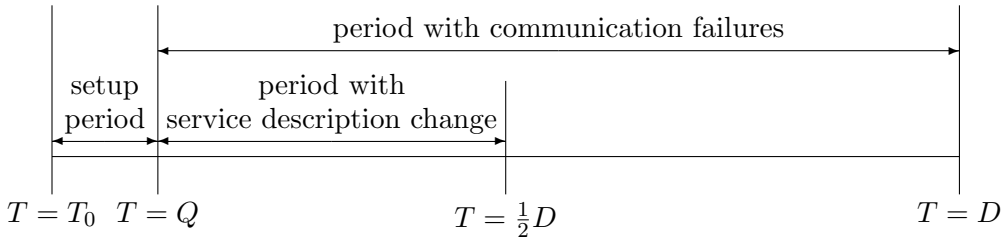


Figure 7.1: Simulation Scenario Timeline

ing packets are lost. The failure will last for $F \times D$, where F is the fraction of the total run time that a node will experience a communication failure. This fraction is the same for each node. The start time of the interface failure of node n (T_{F_n}) is a random value in the range $Q <= T_{F_n} < D - F \times D$.

At T_C the SM will change its service registration. With T_C a random value in the range $Q <= T_C < \frac{1}{2}D$. After this the service change will have to be propagated to each SU, either through notifications sent to the SUs by the SCM, or through polling of the SCM by the SUs.

7.2 Metrics

To compare the model of FRODO with the models of Jini and UPnP the following metrics defined by NIST are used [DME02a]:

Update Responsiveness (R) Let D be a deadline by which some information should be propagated to each SU u. Let T_C be the creation time of the information, where $T_C < D$. Let T_{U_u} be the time that the information is propagated to SU u, where $u = 1$ to U, and U is the total number of SUs in the simulation. Define change-propagation latency (L) for SU u as:

$$L_u = \frac{T_{U_u} - T_C}{max(D, T_{U_u}) - T_C} \tag{7.1}$$

This is effectively the proportion of available time used to propagate the change to SU u. Then define R for SU u as: $R_u = 1 - L_u$. R_u is the proportion of available time remaining after propagating a change to SU u.

A higher value for this metric means that the protocol will respond faster to changes in a Service Description.

Update Effectiveness Measures the probability that a change will propagate successfully. Let $R_{u,i}$ be the update responsiveness R_u for simulation run i, where $i = 1$ to N, and N is the total number of simulation runs under identical conditions for a particular experiment.

$$Effectiveness = 1 - P(F) \tag{7.2}$$

where

$$P(F) = \frac{1}{U \times N} \sum_{u=1}^{U} \sum_{i=1}^{N} (1 | R_{u,i} = 0 \wedge 0 | R_{u,i} \neq 0) \tag{7.3}$$

A higher value for this metric means that the protocol is better at recovering from network failures.

Update Efficiency Is defined as the ratio of M to the actual number of messages observed, where M is the minimum number of messages needed to propagate a change to all SUs. In this case M ($M = 7$) occurs when using notification with Jini. Let S_i be the total number of messages sent while attempting to propagate a change to all SUs in run i. Then define E as:

$$Efficiency = \frac{1}{N} \sum_{i=1}^{N} \frac{M}{S_i} \tag{7.4}$$

A higher value for this metric means that the protocol needs less network traffic to achieve its goal.

7.3 Simulation Procedure

For both scenarios a number of experiments are conducted. An experiment consists of a number of simulation runs for each interface failure rate (F). The interface failure rate is varied from 0 to 75% in 5% increments. All other conditions will be the same throughout the entire experiment.

The protocol, network and simulation parameters used in the experiments are listed in table 7.1.

An experiment is started with a run script. This script takes care of executing the required number of simulation runs for each of the failure rates. After each simulation run the trace log that is produced is processed by an AWK script to extract the data needed to calculate the metrics.

To verify the simulation procedure and the parameters, an experiment has been conducted on the Rapide model of Jini to try to reproduce the results obtained by NIST [DME02a]. Figures 7.2, 7.3 and 7.4 show the results of this

Parameter	Value
Polling interval	$180s$
Registration interval	$1800s$
Subscription interval	$1800s$
UDP transmission delay	$10\mu s$ constant
TCP transmission delay	$10 - 100\mu s$ uniform
Processing delay for cache items	$100\mu s$
Processing delay for non-cache items	$10\mu s$

Table 7.1: Simulation parameters.

experiment: graphs of the median update responsiveness, update effectiveness and update efficiency plotted against the interface failure rate for both the polling scenario and the notification scenario. The reproduced graphs are similar to the graphs published by NIST. This leads to the conclusion that the simulation procedure and the used parameters are correct.

The results of the same experiments conducted on the model of FRODO are presented in the next chapter. m

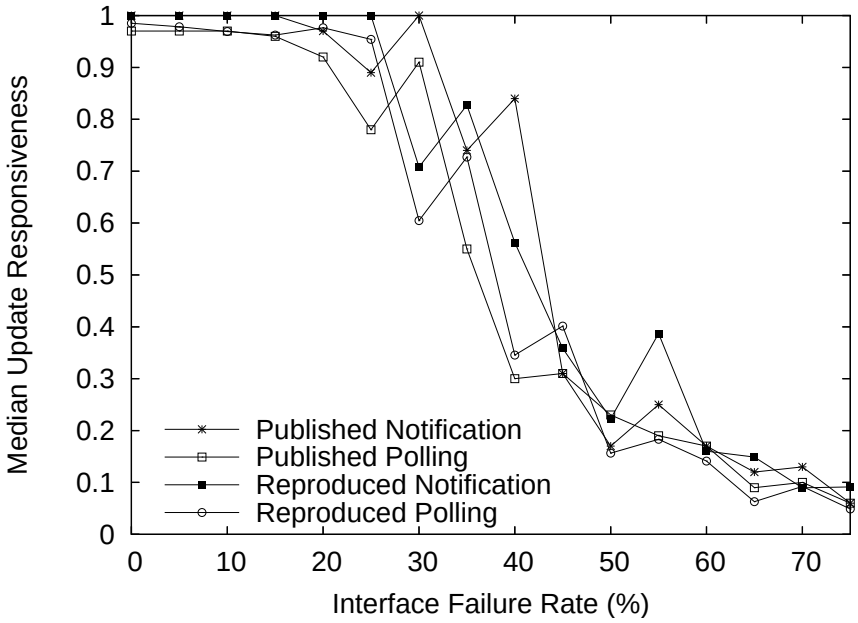


Figure 7.2: Median Update Responsiveness Jini

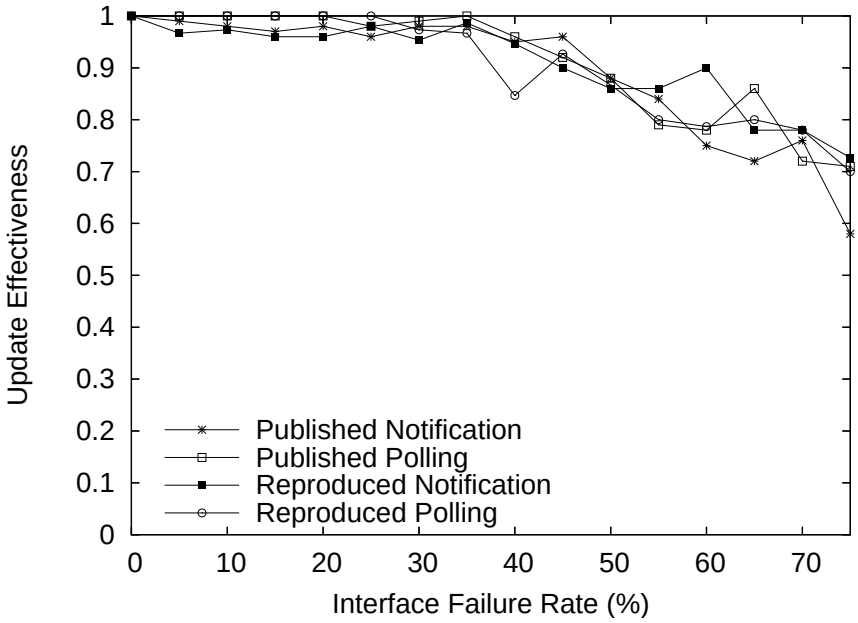


Figure 7.3: Update Effectiveness Jini

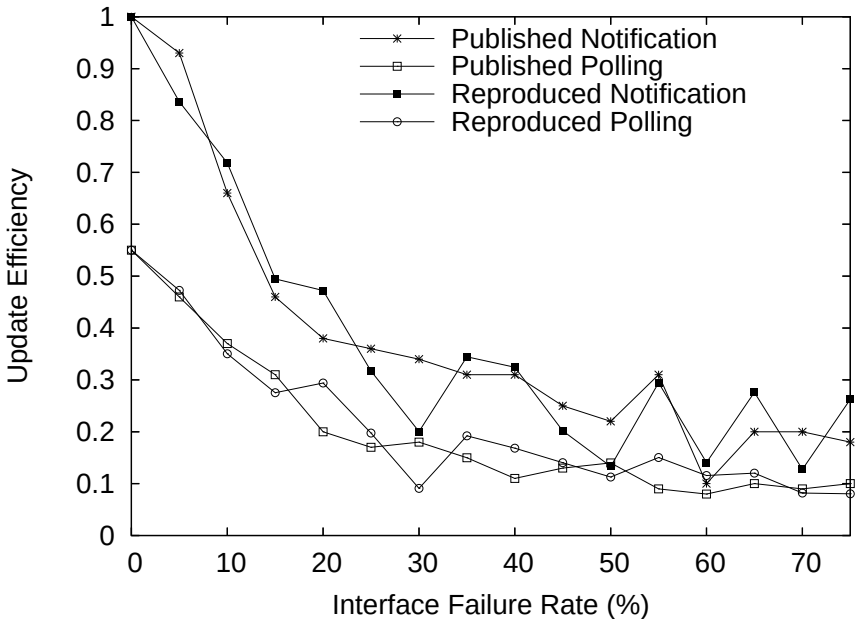


Figure 7.4: Update Efficiency of Jini

Chapter 8

Simulation Results

In this chapter the results of the experiments conducted on the Rapide model of FRODO are presented. In section 8.2 these results will be compared against the results of similar experiments conducted by the National Institute of Standards and Technology on the Rapide models of Jini and Universal Plug and Play.

8.1 Reproducibility

The experiments from NIST use 30 runs for each of the interface failure rates in the calculation of the metrics discussed in the previous chapter (update responsiveness, efficiency and effectiveness). Figure 8.1 shows a graph of the median update responsiveness of FRODO calculated from the results obtained from three separate experiments. Each of the experiments is conducted under the same conditions and uses 30 runs, just like the experiments from NIST.

Because the conditions in each of the three experiments were the same, one would expect to the results to be the same as well. But figure 8.1 clearly shows that this is not the case. Figure 8.2 shows what happens if the number of runs is increased to 300. The curves in this graph are much closer together, showing that the reproducibility is much better.

Because of this all further experiments on FRODO use 300 runs instead of the 30 used by NIST.

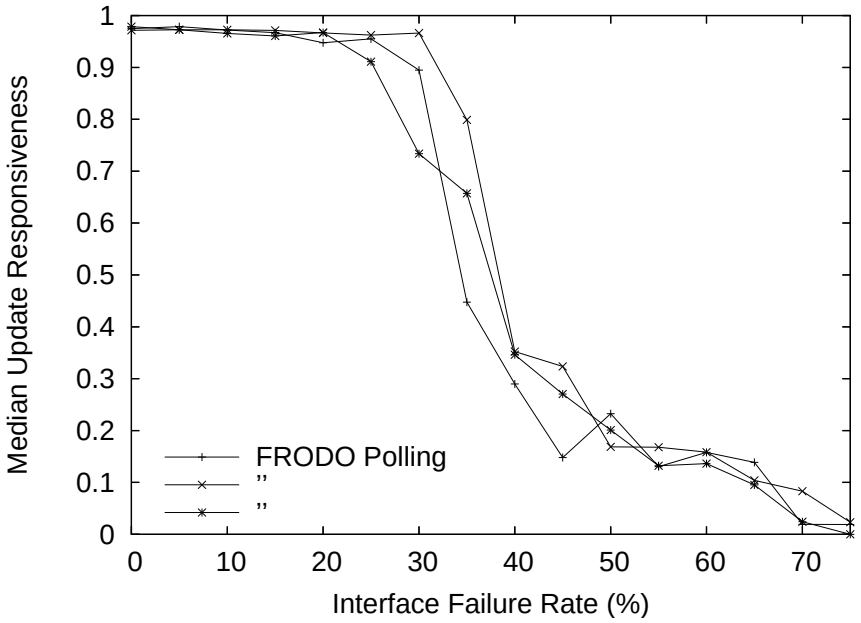


Figure 8.1: Reproducibility of results with 30 runs.

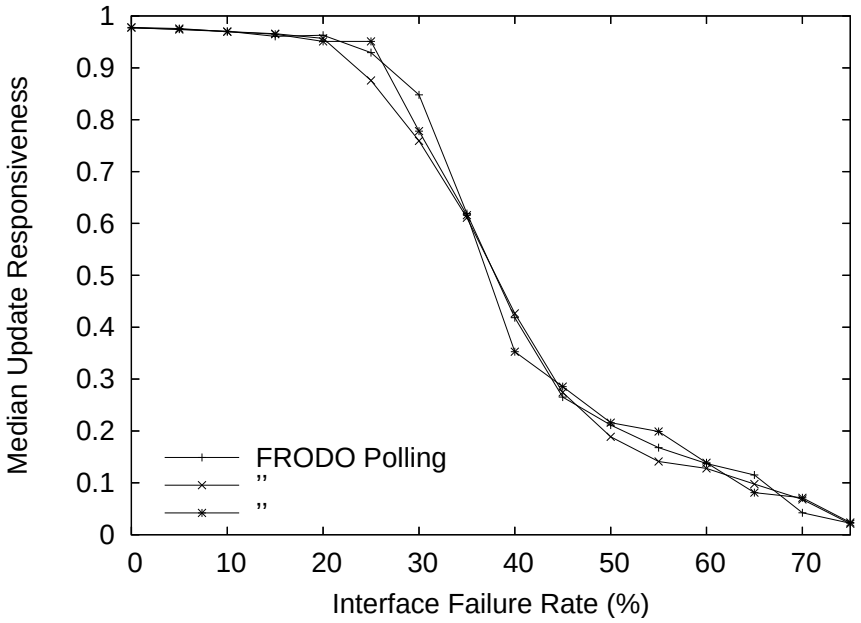


Figure 8.2: Reproducibility of results with 300 runs.

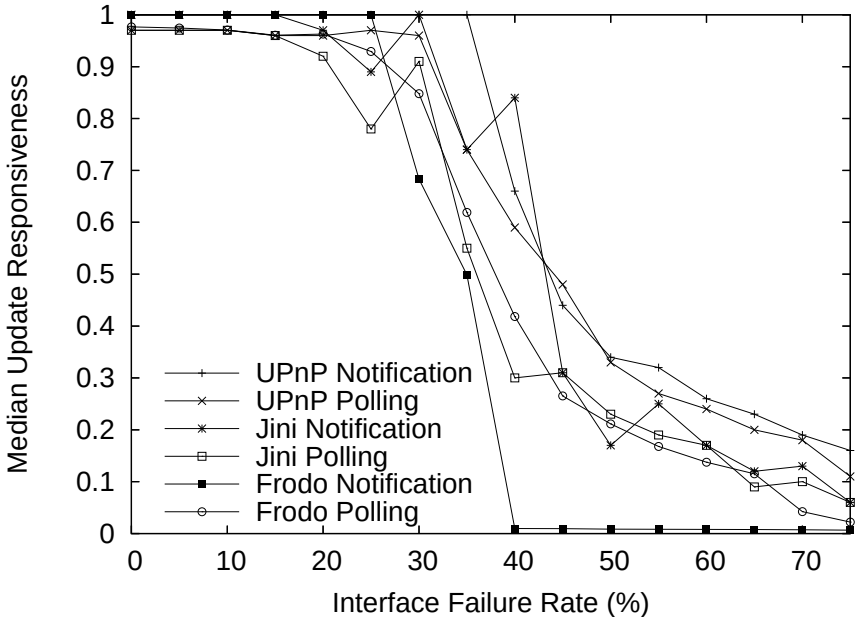


Figure 8.3: Median Update Responsiveness.

8.2 Results

Figures 8.3, 8.4 and 8.5 show the results from simulating the scenarios discussed in the previous chapter using the Rapide model of FRODO. These results are compared with the results obtained from simulating the same scenarios using the Rapide models of Jini and UPnP as published by NIST.

The figures show that the experiments on FRODO using the polling scenario yield similar results to those on Jini. This is as expected: both these protocols use a client/server architecture, unlike UPnP, which uses a peer to peer architecture. And in these experiments both protocols use one dedicated SCM. The saw-tooth behavior for FRODO is much less prominent than for Jini and UPnP. This is caused by the fact that the graphs for FRODO are the result of combining 300 simulation runs, while the graphs created by NIST use only 30 runs.

The results for the experiments using the notification scenario show something different: the performance of FRODO is much worse than the performance of the other protocols. Above an interface failure of 30%, the update responsiveness drops of to a value close to zero. The update effectiveness confirms that it is a value close to zero and not zero itself: from the definitions of the median update responsiveness and the update effectiveness follows that the value of the update effectiveness is less than 0.5 for an update responsiveness of 0, and the update effectiveness stays above 0.7.

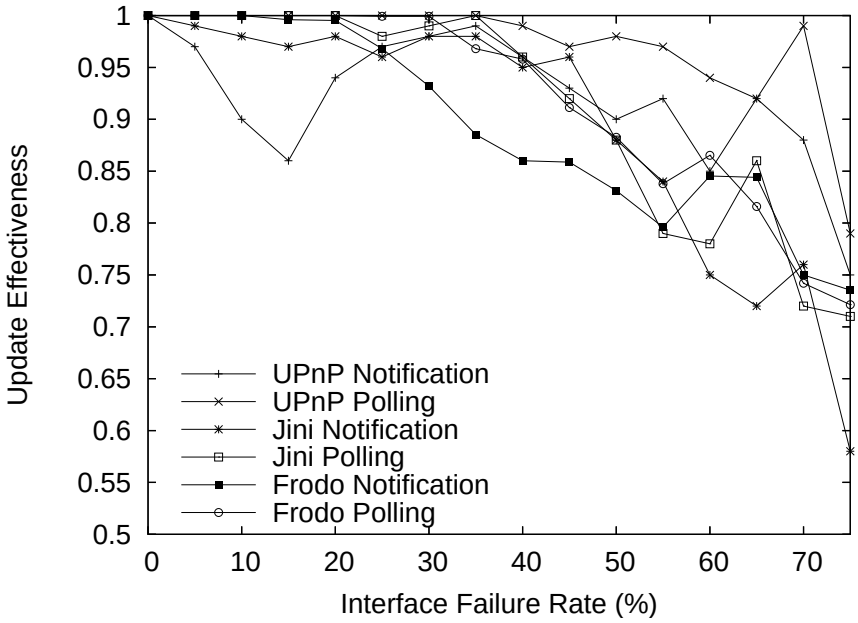


Figure 8.4: Update Effectiveness.

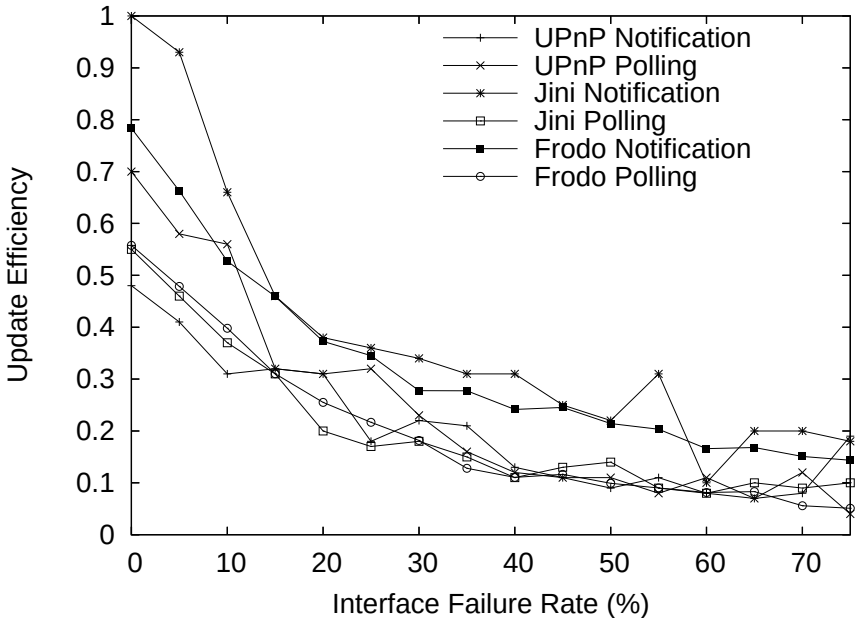


Figure 8.5: Update Efficiency.

Analysis of the trace logs of the simulation runs reveals the cause of the sudden drop in responsiveness: when using notification the SM will send a message to the SCM on changing its service description. If the SM does not receive an acknowledgement to this message, it will be resend only once, but because the interface failure will probably not have been cleared, this message will also be lost. This means that the SCM will not receive the service description change until the SM renews its service lease. Because of the timings used in the protocol and simulation, this renew happens just before the simulation deadline, which explains why the responsiveness is very close to zero.

The evaluation of the simulation results leads to the following conclusions: the conducted experiments show that the Rapide model is valid and that the simulation enviroment is equivalent to that used by NIST. The simulations also gave an insight in the behavior of FRODO and helped to enhance the protocol specification by uncovering some of the weaknesses it contained.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

The conclusions of the work done in this thesis can be summarized by the following:

- *The Rapide language is well suited for creating models of network protocols.* The focus on the parallel processing of events matches the nature op network protocols and the timing model of the language makes it easy to model the delays and timeouts needed in a network protocol. However, the executable sublanguage of Rapide is not as powerful, and long and complicated calculations can best be done in a separate program written in another language. Also, the Rapide toolset is difficult to install on a modern computer.
- *The modeling of FRODO has created a flexible simulation framework.* It allows the user to conduct a large number of different experiments without the need to make changes to the source code. The framework can also be extended to include parts of the protocol that have been left unmodeled in this version.
- *Modeling and simulating FRODO has proved an efficient way to validate the design of FRODO.* Flaws in the protocol specification were found both during the modeling and during the simulation sessions. Most of these could immediately be corrected, because the specification was still a work in progress.

9.2 Future Work

The following ideas for future work are proposed:

- *Find an easy way to install Rapide on a modern GNU/Linux system.* Maybe even create a new binary distribution from the source code.
- *Model unmodeled parts of the protocol.* Add Central Election, Central Backup, Peer-to-Peer Search and Remote Control to the model. Some of this work has already been done [GLI05].
- *Perform more expermiments.* Many more scenarios can be simulated to gather more information on FRODO's characteristics.
- *Create a prototype implementation of FRODO.* The model of FRODO can be used as a basis for a prototype implementation.
- *Evaluate other parts of the Rapide language.* Rapide contains a constraint language not discussed in this thesis that could be usefull.

Bibliography

[AHA03] The At Home Anywhere Research Project. `http://wwwes.cs.utwente.nl/aha`, last modified: July 2003

[BLUE99] Specification of the Bluetooth System, Core, Volume 1, Version 1.1 Bluetooth SIG, Inc., February 2001

[DM01] C. Dabrowski and K. Mills. Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-based Approach. In *Working Conference on Complex and Dynamic Systems Architectures, Brisbane, Australia*, December 2001.

[DM02a] C. Dabrowski and K. Mills. Rapide Specification for Discovery Architecture: Jini 3-Party. Last update: March 2002, Contact Authors.

[DM02b] C. Dabrowski and K. Mills. Rapide Specification for Discovery Architecture: Universal Plug and Play 2-Party. Last update: March 2002, Contact Authors.

[DME02a] C. Dabrowski, K. Mills, and J. Elder. Understanding Consistency Maintenance in Service Discovery Architectures During Communications Failures. In *Third International Workshop on Software Performance, Rome, Italy*, July 2002.

[DME02b] C. Dabrowski, K. Mills, and J. Elder. Understanding Consistency Maintenance in Service Discovery Architectures in Response to Message Loss. In *Fourth Annual Workshop on Active Middleware Services, Edinburgh, Scotland*, July 2002.

[GLI05] G.J. van de Glind. *Implementation and Analysis of FRODO in Rapide*. University of Twente, March 2005

[GOL02] Golden G. Richard III *Service and Device Discovery, Protocols and Programming*. McGraw Hill, 2002.

[JINI03] Jini Architecture Specification. Sun Microsystems, June 2003.

[RAP97] Rapide Design Team. Guide to the Rapide 1.0 Language Reference Manuals. Stanford University, July 1997.
`http://pavg.stanford.edu/rapide/lrms/overview.ps`

[RAP98] The Stanford Rapide Project.
`http://pavg.stanford.edu/rapide`, last modified: July 1998

[RFC2608] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2, RFC2165 Internet Engineering Task Force, June 1999.
`http://www.ietf.org/rfc/rfc2608.txt`

[SAL99] Salutation Architecture Specification (Part 1), Version 2.1. Salutation Consortion, 1999.

[SUN03] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service Discovery at Home. In *4th International Conference on Information, Communications & Signal Processing and 4th IEEE Pacific-Rim Conference on Multimedia (ICICS/PCM), Singapore*, volume III, December 2003

[SUN05] V. Sundramoorthy. An Analytical Approach Towards Designing a Service Discovery Protocol: Introducing FRODO, Draft Version. University of Twente, March 2005

[UPNP00] Universal Plug and Play Device Architecture, Version 1.0. Microsoft Corporation, June 2000.

Abbreviations

NIST	National Institute of Standards and Technology
SCM	Service Cache Manager
SD	Service Description
SDP	Service Discovery Protocol
SLP	Service Location Protocol
SM	Service Manager
SP	Service Provider
SU	Service User
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UPnP	Universal Plug and Play

Appendix A

Installing Rapide

Because the GNU/Linux versions of the Rapide tools have been compiled as an a.out binary, it is impossible to install it on a modern distribution. It is possible to install it using an older distribution (possibly using a virtual machine). The following procedure can be followed to install the tools on a freshly installed Redhat 6.2 system:

1. Download the following Rapide install files from `ftp://pavg.stanford.edu/pub/Rapide-1.0/toolset/`:

   ```
   rapide.LINUX.build72.tar.gz
   debug/pov2.LINUX.tar.gz
   debug/raparch.0.9.6.Linux.tar.gz
   debug/povwish.LINUX.build72.1.tar.gz
   ```

2. Unpack the Rapide distribution:

   ```
   # cd /usr
   # tar -xvzf <path to files>/rapide.LINUX.build72.tar.gz
   # cd /usr/rapide
   # tar -xvzf <path to files>/pov2.LINUX.tar.gz
   # tar -xvzf <path to files>/raparch.0.9.6.Linuxcd /.tar.gz
   # cd /usr/rapide/lib/povwish
   # tar -xvzf <path to files>/povwish.LINUX.build72.1.tar.gz
   ```

3. Add the following directory to /etc/ld.so.conf:

   ```
   /usr/rapide/lib/gcc-lib/i586-unknown-linux/i2.6.3
   ```

4. Update `ld.so.cache` by running:

```
# ldconfig
```

5. Create a symbolic link to `types.h` (Rapide can't find the one already on the system):

```
# cd /usr/rapide/lib/gcc-lib/i586-unknown-linux/i2.6.3/include
# mkdir gnu
# cd gnu
# ln -s /usr/include/bits/types.h types.h
```

6. Add the following lines to the initialization file (for bash: `~/.bashrc`)

```
export RAPIDEHOME=/usr/rapide
export PATH=$PATH:$RAPIDEHOME/bin:.
export MANPATH=$RAPIDEHOME/man:$MANPATH
```

7. Make sure that a.out binaries are supported. Add the following line to /etc/rc.d/rc.modules on a Redhat 6.2 installation (`/etc/rc.d/rc.modules` is an executable shell script):

```
modprobe binfmt_aout
```

8. Apply the following patch to the regression test makefile:

```
# cd /usr/rapide/regress/tests
# cat << EOF | patch
--- Makefile 1997-12-16 02:30:10.000000000 +0100
+++ Makefile 2004-09-20 16:49:12.000000000 +0200
@@ -23,12 +23,16 @@

 all: latest_library latest_rpdc execs tests_passed

-latest_library: $(R.MANAGER) .rpdlib/PATH
+latest_library: $(R.MANAGER) clean_library .rpdlib/PATH
+       @touch latest_library
+
+.PHONY: clean_library
+
+clean_library:
        @if [ `$(EXPR) length "$(DEBUG)"` != 0 ] ; then \
                echo "Going to clean because new Rapide library manager"; \
        fi
        @$(MAKE) clean
-       @touch latest_library
```

```
 .rpdlib/PATH:
        @if [ `$(EXPR) length "$(DEBUG)"` != 0 ] ; then \
EOF
```

9. Run the Rapide installation script:

```
# installation_setup
```

10. Fix the file ownership and permissions:

```
# cd /usr/rapide
# chown -R root.root .
# chmod -R go-w .
```

11. Done.

Appendix B

Protocol Specification

This chapter contains the protocol specification flowchart used for the model of FRODO. This is not the final version of the flowchart and it should only be used as guide to the Rapide model of FRODO presented in this thesis.

Figure B.1 shows the building blocks that are used in the flowchart. The circle refers to a node or device in the network. It gives the type of node that should implement the behavior. An arrow represents a transition from one building block to another. Any text next to the arrow represents the event that triggers the transition. If there is no event attached to an arrow, the corresponding transition is triggered directly. A transition can go to another page on in the flowchart: this is represented by the grey box. The diamond symbol represents a decision and the box a process.

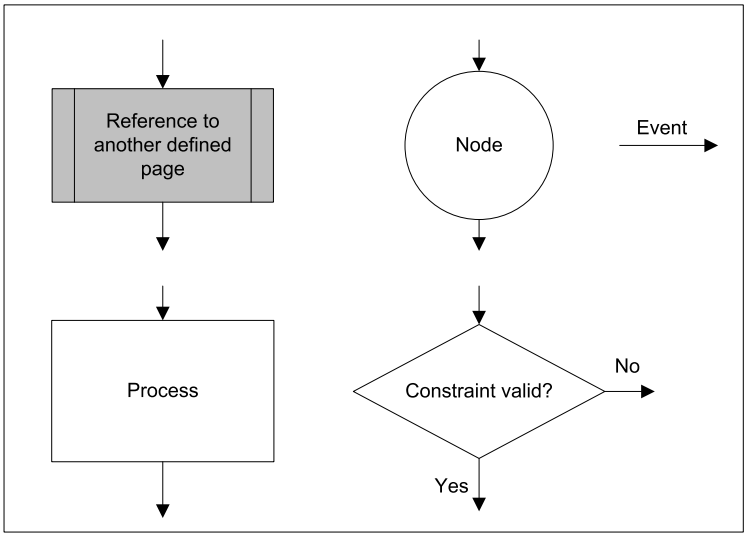
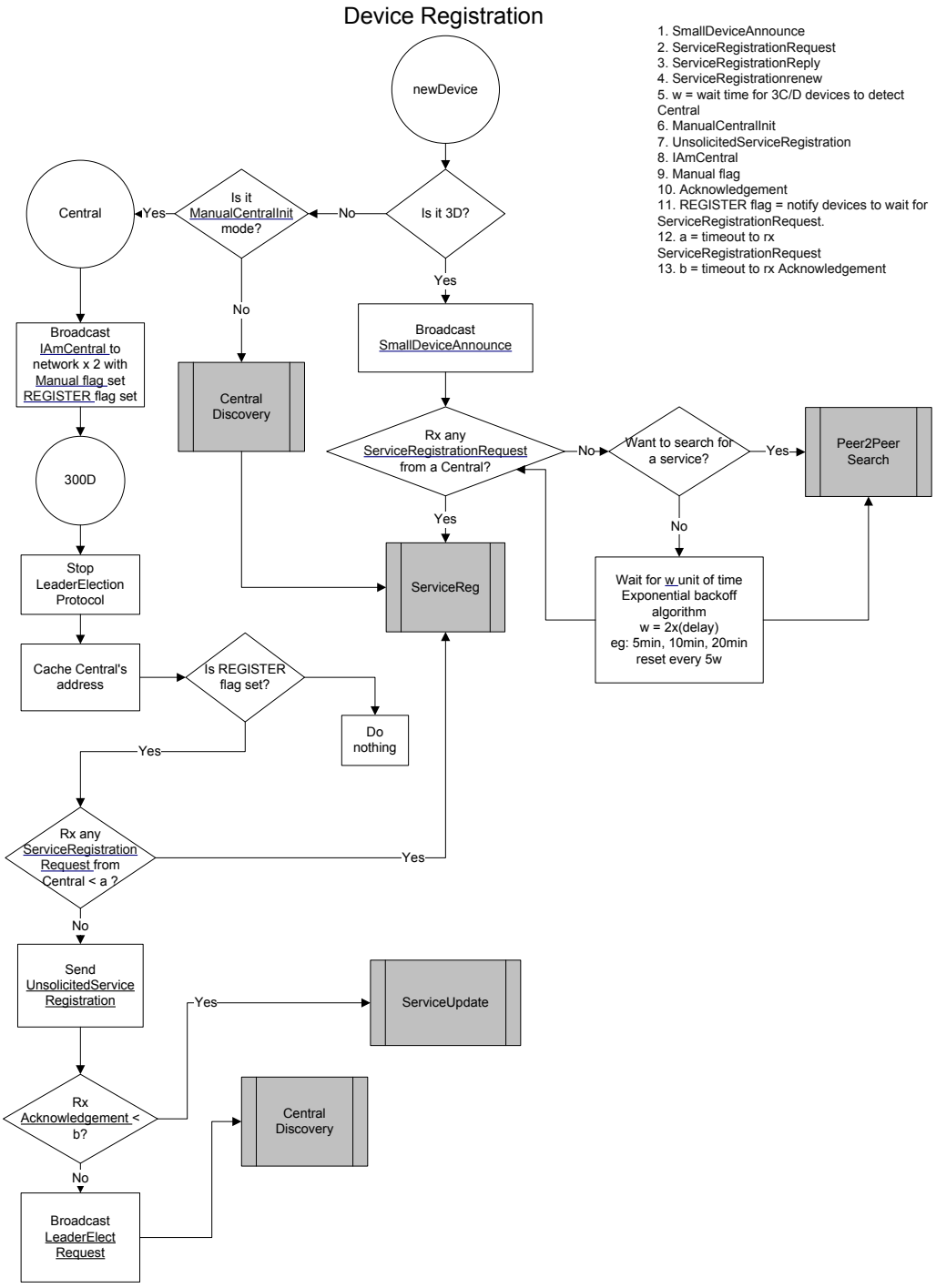
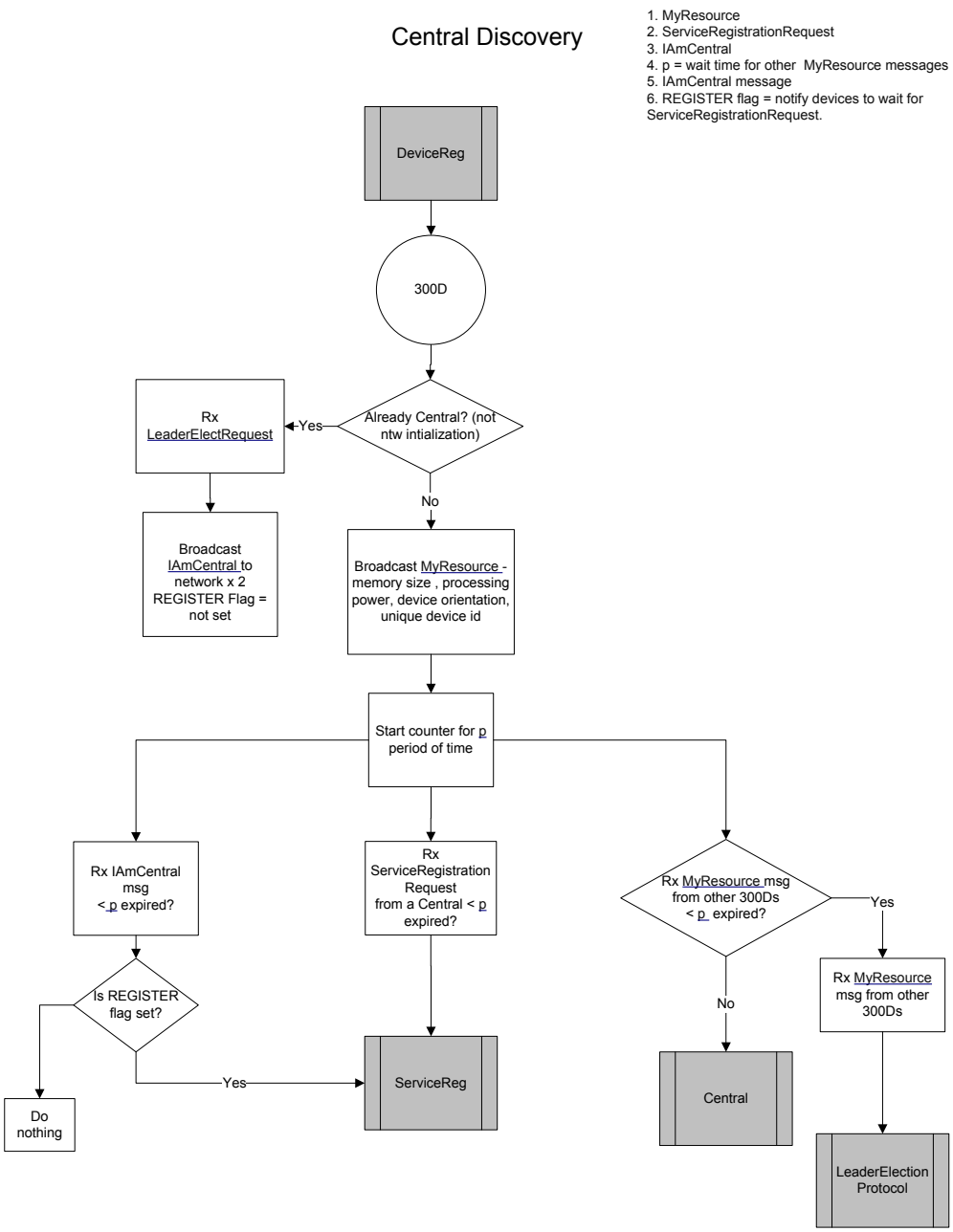
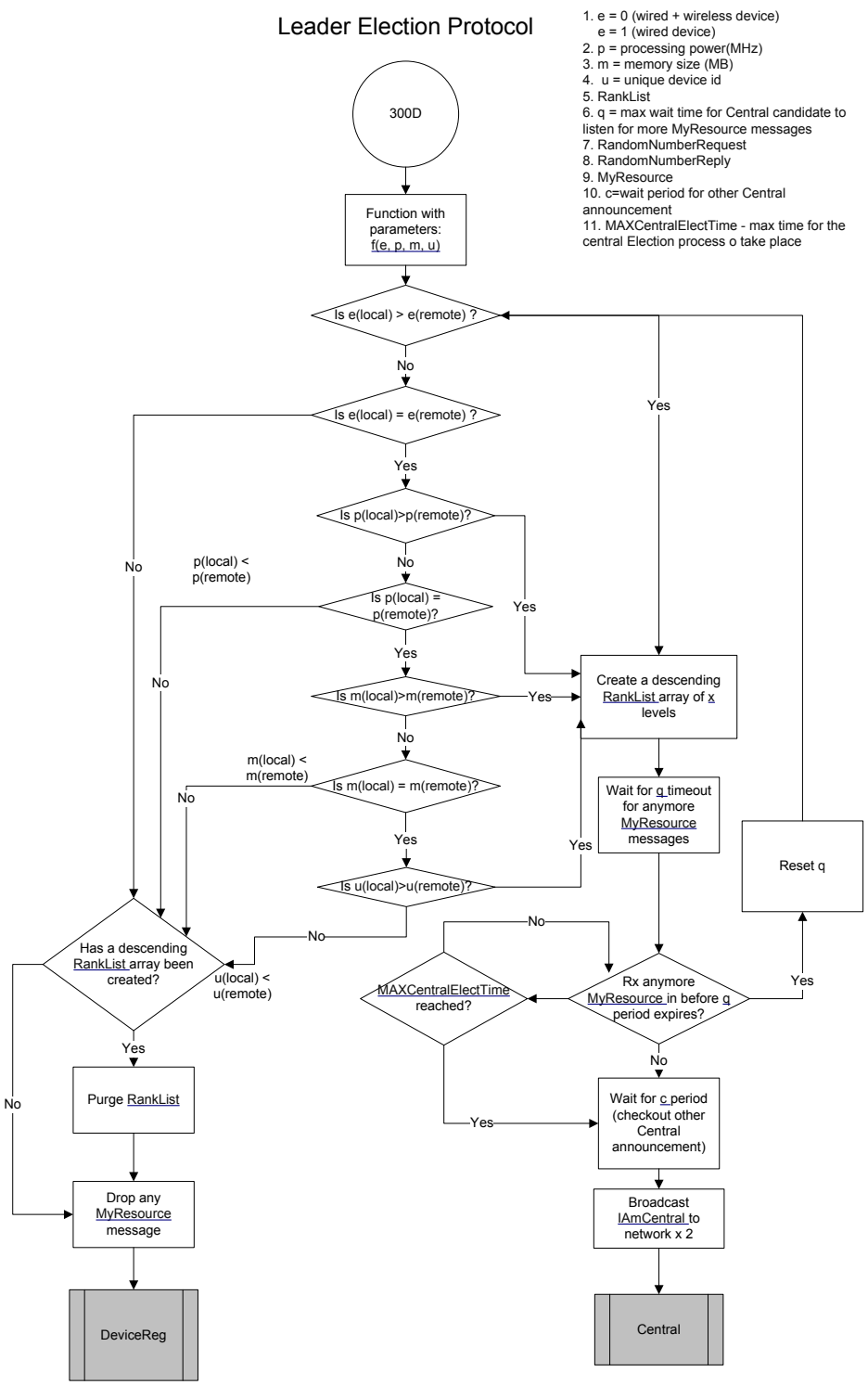
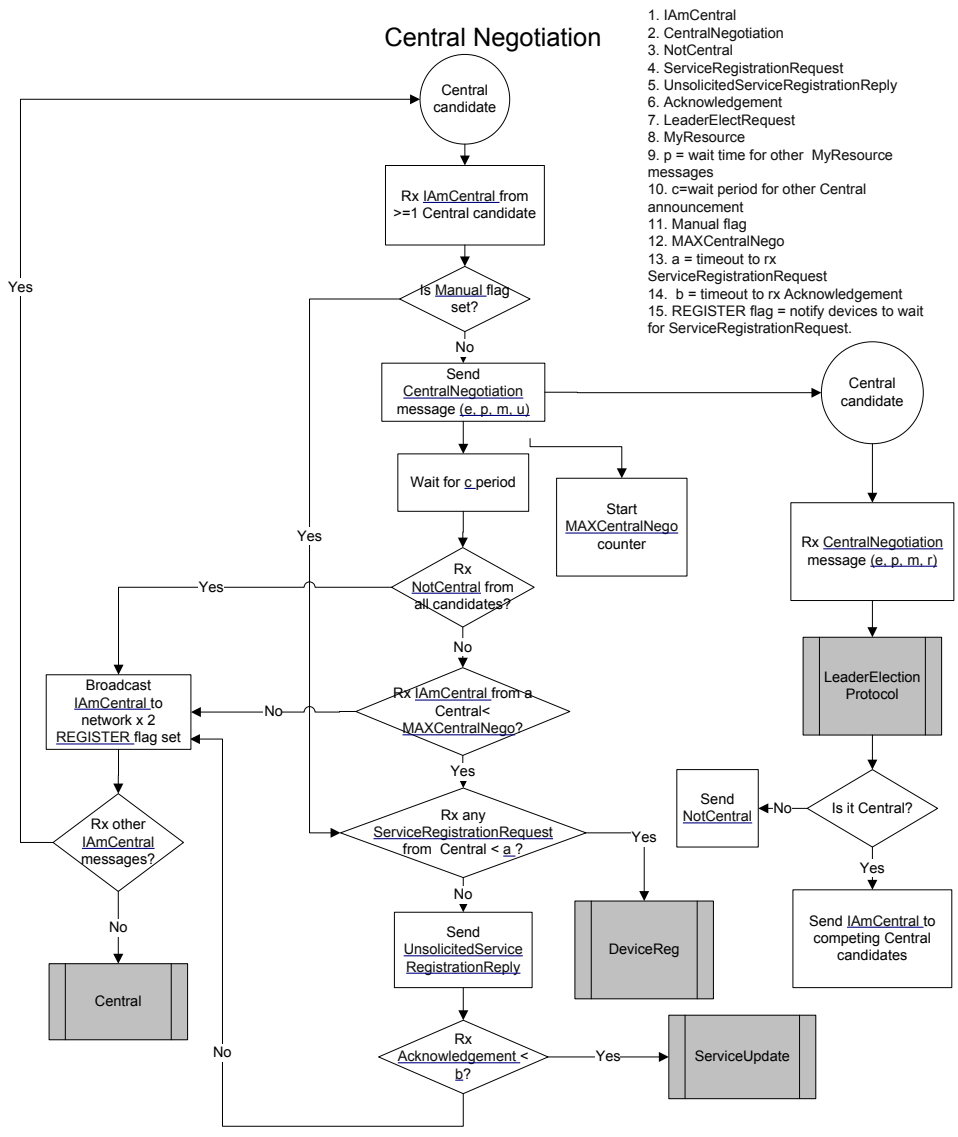


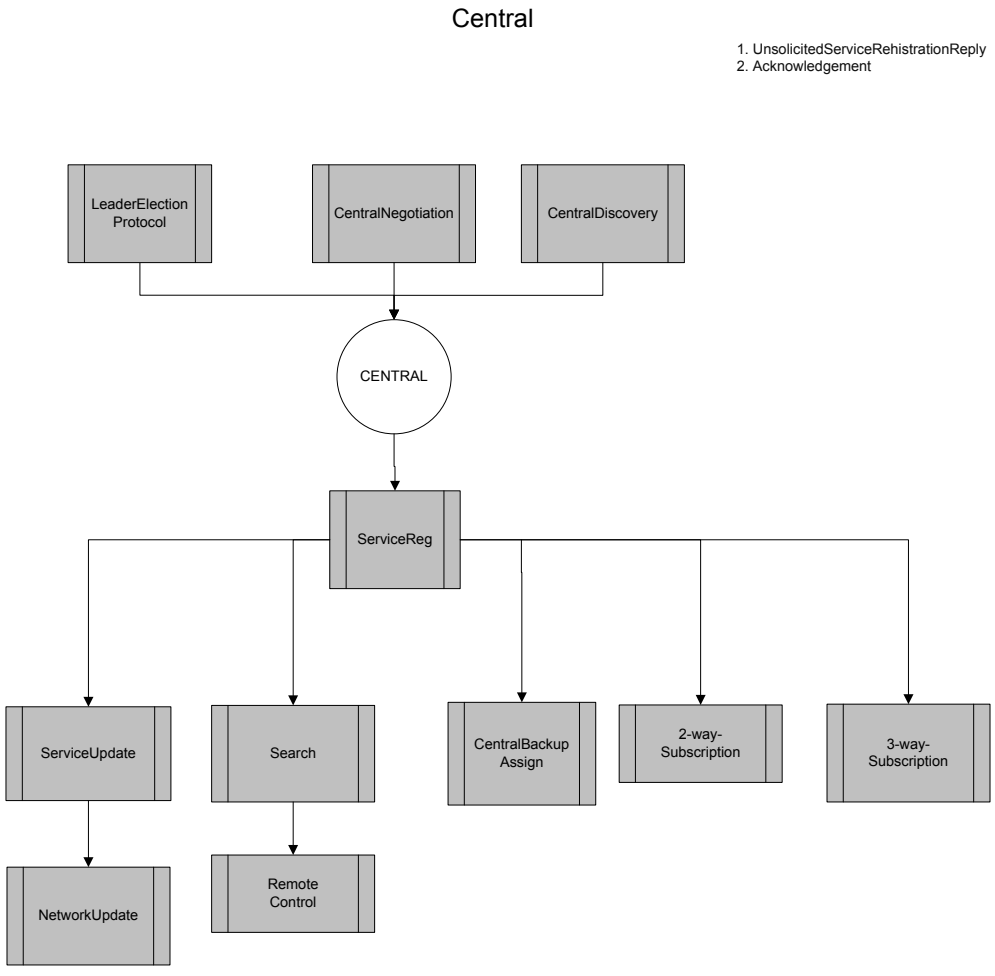
Figure B.1: Flowchart building blocks.

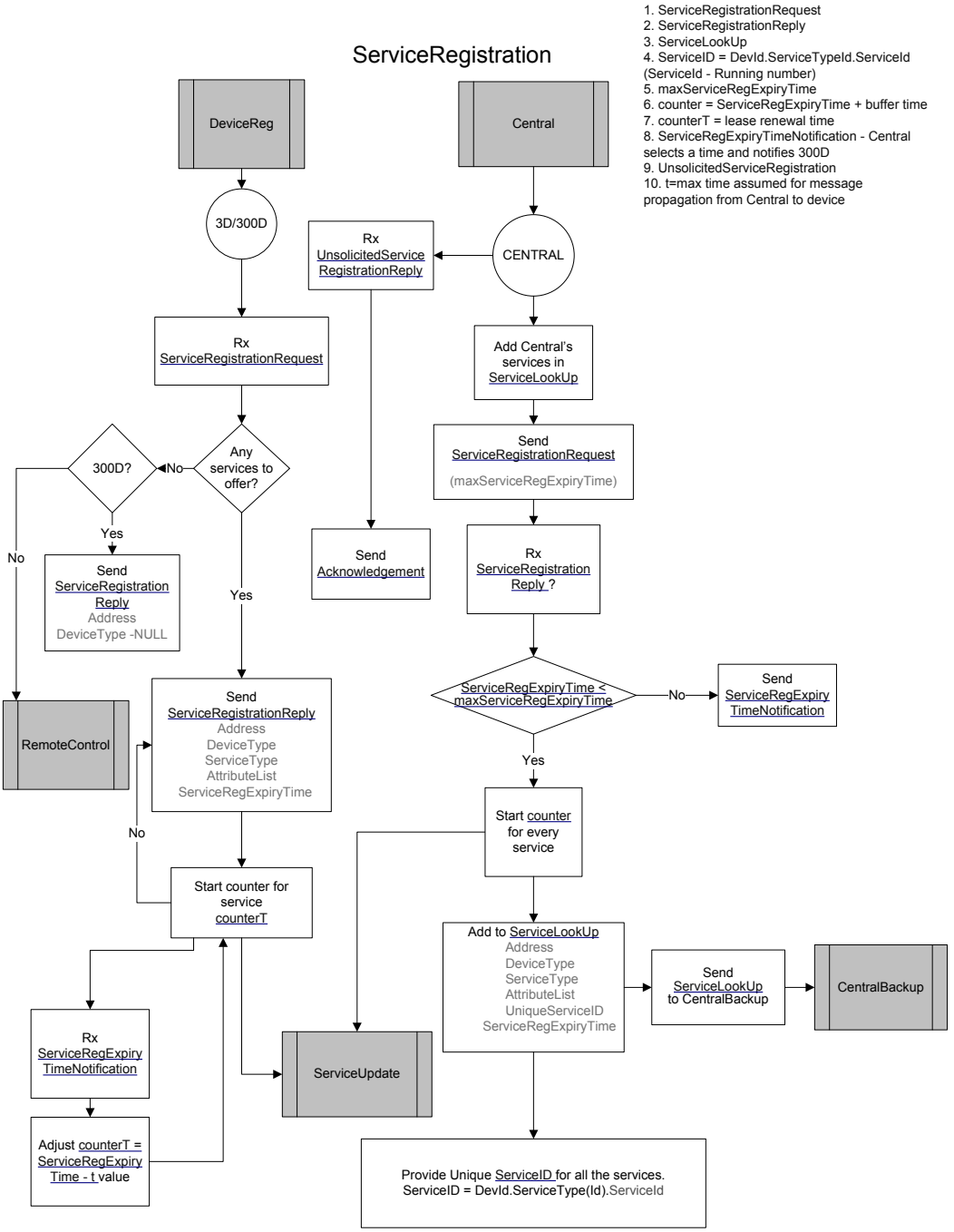

Device Registration
1. SmallDeviceAnnounce
2. ServiceRegistrationRequest
3. ServiceRegistrationReply
4. ServiceRegistrationrenew
5. w = wait time for 3C/D devices to detect Central
6. ManualCentralInit
7. UnsolicitedServiceRegistration
8. IAmCentral
9. Manual flag
10. Acknowledgement
11. REGISTER flag = notify devices to wait for ServiceRegistrationRequest.
12. a = timeout to rx ServiceRegistrationRequest
13. b = timeout to rx Acknowledgement
newDevice
Is it 3D?
No
Is it ManualCentralInit mode?
Yes
Central
Yes
No
Broadcast SmallDeviceAnnounce
Broadcast IAmCentral to network x 2 with Manual flag set REGISTER flag set
Central Discovery
Rx any ServiceRegistrationRequest from a Central?
No
Want to search for a service?
Yes
Peer2Peer Search
300D
Yes
No
Stop LeaderElection Protocol
ServiceReg
Wait for w unit of time Exponential backoff algorithm w = 2x(delay) eg: 5min, 10min, 20min reset every 5w
Cache Central's address
Is REGISTER flag set?
Do nothing
Yes
Rx any ServiceRegistration Request from Central < a ?
Yes
No
Send UnsolicitedService Registration
Yes
ServiceUpdate
Rx Acknowledgement < b?
Central Discovery
No
Broadcast LeaderElect Request

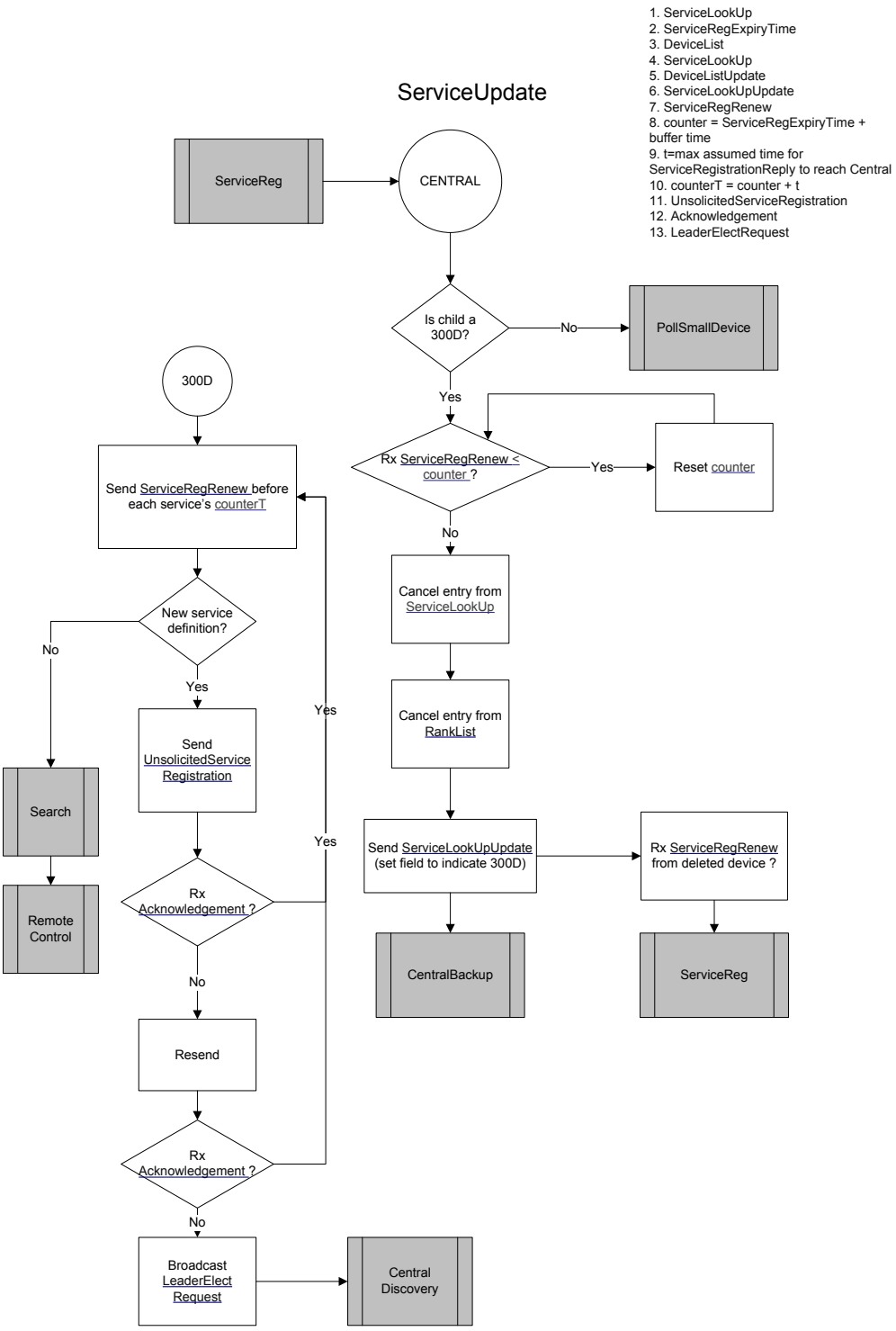

Central Discovery
1. MyResource
2. ServiceRegistrationRequest
3. IAmCentral
4. p = wait time for other MyResource messages
5. IAmCentral message
6. REGISTER flag = notify devices to wait for ServiceRegistrationRequest.
DeviceReg
300D
Already Central? (not ntw intialization)
Yes
Rx LeaderElectRequest
No
Broadcast IAmCentral to network x 2 REGISTER Flag = not set
Broadcast MyResource - memory size , processing power, device orientation, unique device id
Start counter for p period of time
Rx IAmCentral msg < p expired?
Rx ServiceRegistration Request from a Central < p expired?
Rx MyResource msg from other 300Ds < p expired?
Yes
No
Is REGISTER flag set?
Yes
Do nothing
ServiceReg
Central
Rx MyResource msg from other 300Ds
LeaderElection Protocol


Leader Election Protocol
1. e = 0 (wired + wireless device)
e = 1 (wired device)
2. p = processing power(MHz)
3. m = memory size (MB)
4. u = unique device id
5. RankList
6. q = max wait time for Central candidate to listen for more MyResource messages
7. RandomNumberRequest
8. RandomNumberReply
9. MyResource
10. c=wait period for other Central announcement
11. MAXCentralElectTime - max time for the central Election process o take place
300D
Function with parameters: f(e, p, m, u)
Is e(local) > e(remote) ?
No
Yes
Is e(local) = e(remote) ?
Yes
No
Is p(local)>p(remote)?
Yes
No
p(local) < p(remote)
Is p(local) = p(remote)?
Yes
No
Is m(local)>m(remote)?
Yes
No
m(local) < m(remote)
Is m(local) = m(remote)?
Yes
No
Is u(local)>u(remote)?
Yes
No
Create a descending RankList array of x levels
Wait for q timeout for anymore MyResource messages
Reset q
Has a descending RankList array been created?
u(local) < u(remote)
Yes
No
Purge RankList
Drop any MyResource message
DeviceReg
MAXCentralElectTime reached?
No
Yes
Rx anymore MyResource in before q period expires?
Yes
No
Wait for c period (checkout other Central announcement)
Broadcast IAmCentral to network x 2
Central

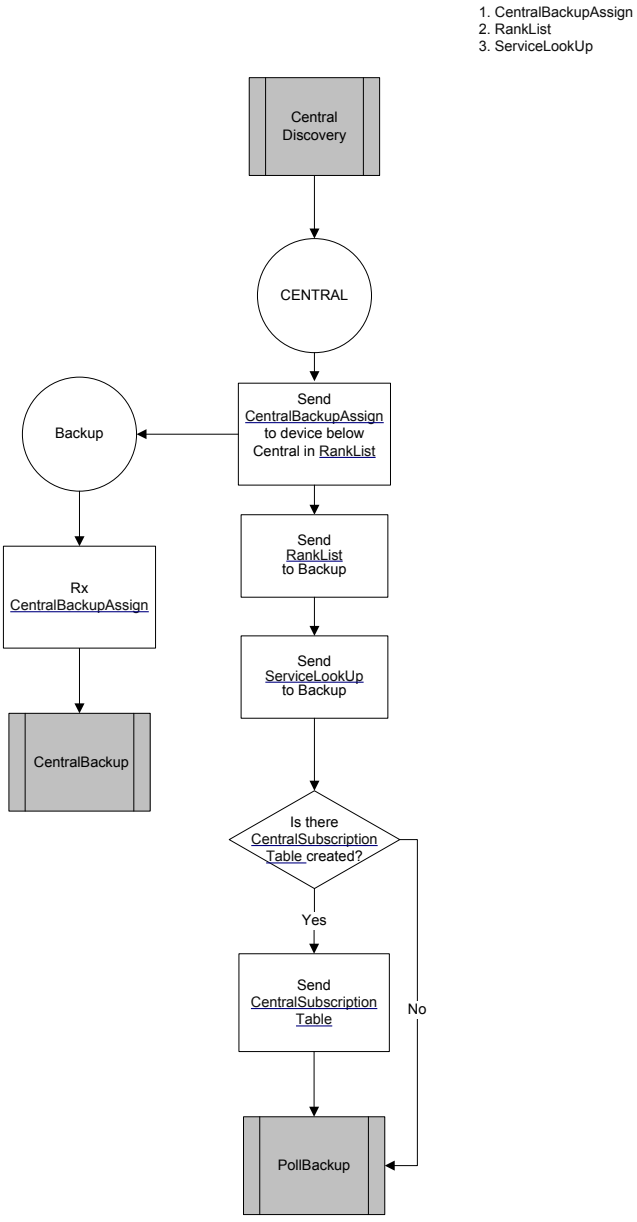

Central Negotiation
1. IAmCentral
2. CentralNegotiation
3. NotCentral
4. ServiceRegistrationRequest
5. UnsolicitedServiceRegistrationReply
6. Acknowledgement
7. LeaderElectRequest
8. MyResource
9. p = wait time for other MyResource messages
10. c=wait period for other Central announcement
11. Manual flag
12. MAXCentralNego
13. a = timeout to rx ServiceRegistrationRequest
14. b = timeout to rx Acknowledgement
15. REGISTER flag = notify devices to wait for ServiceRegistrationRequest.
Central candidate
Rx IAmCentral from >=1 Central candidate
Is Manual flag set?
Yes
No
Send CentralNegotiation message (e, p, m, u)
Central candidate
Wait for c period
Start MAXCentralNego counter
Rx CentralNegotiation message (e, p, m, r)
Yes
Rx NotCentral from all candidates?
Yes
No
LeaderElection Protocol
Broadcast IAmCentral to network x 2 REGISTER flag set
No
Rx IAmCentral from a Central< MAXCentralNego?
Yes
Send NotCentral
No
Is it Central?
Yes
Rx other IAmCentral messages?
Rx any ServiceRegistrationRequest from Central < a ?
Yes
No
Send IAmCentral to competing Central candidates
No
Central
Send UnsolicitedService RegistrationReply
DeviceReg
No
Rx Acknowledgement < b?
Yes
ServiceUpdate

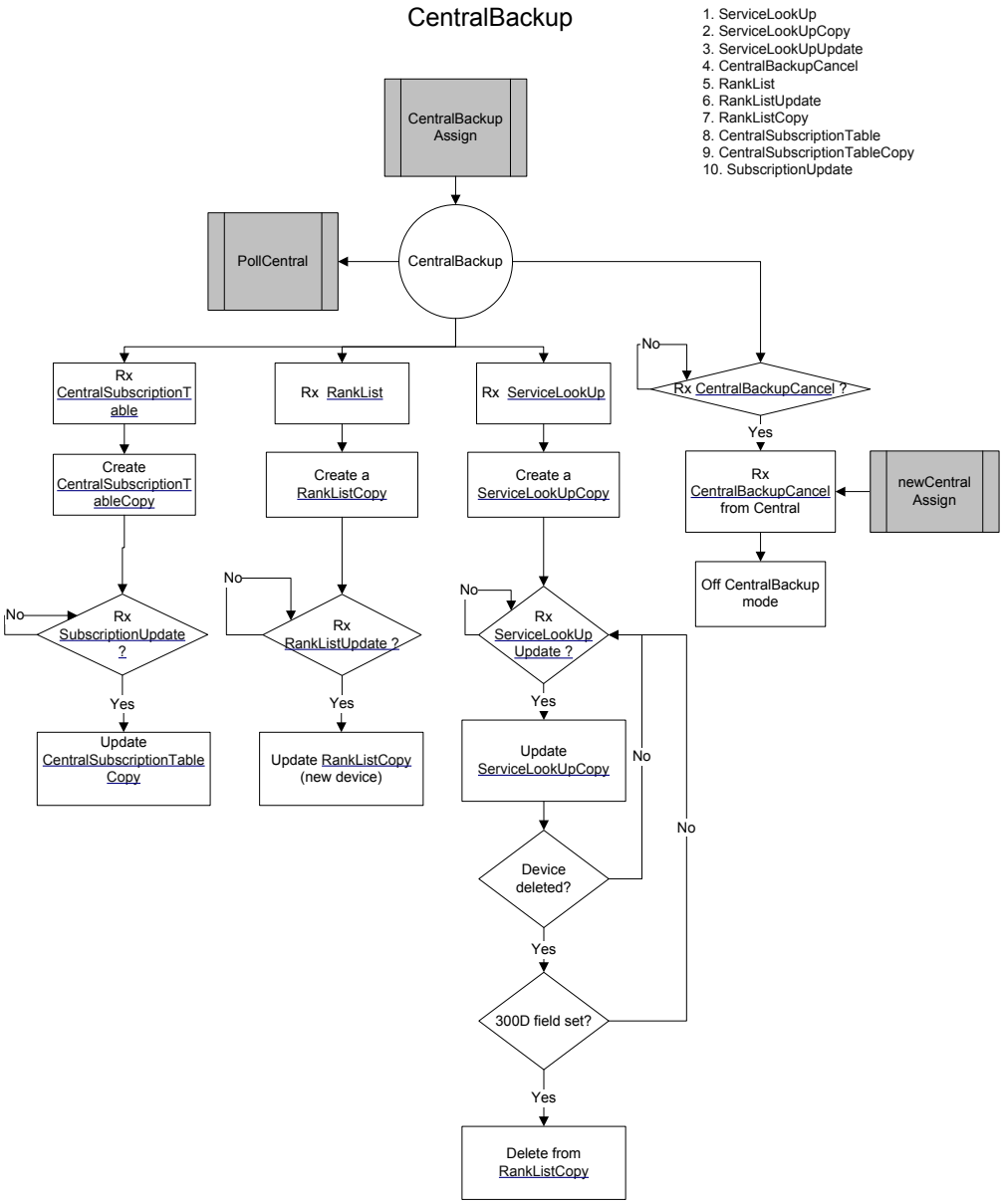

Central
1. UnsolicitedServiceRehistrationReply
2. Acknowledgement
LeaderElection Protocol
CentralNegotiation
CentralDiscovery
CENTRAL
ServiceReg
ServiceUpdate
Search
CentralBackup Assign
2-way-Subscription
3-way-Subscription
NetworkUpdate
Remote Control

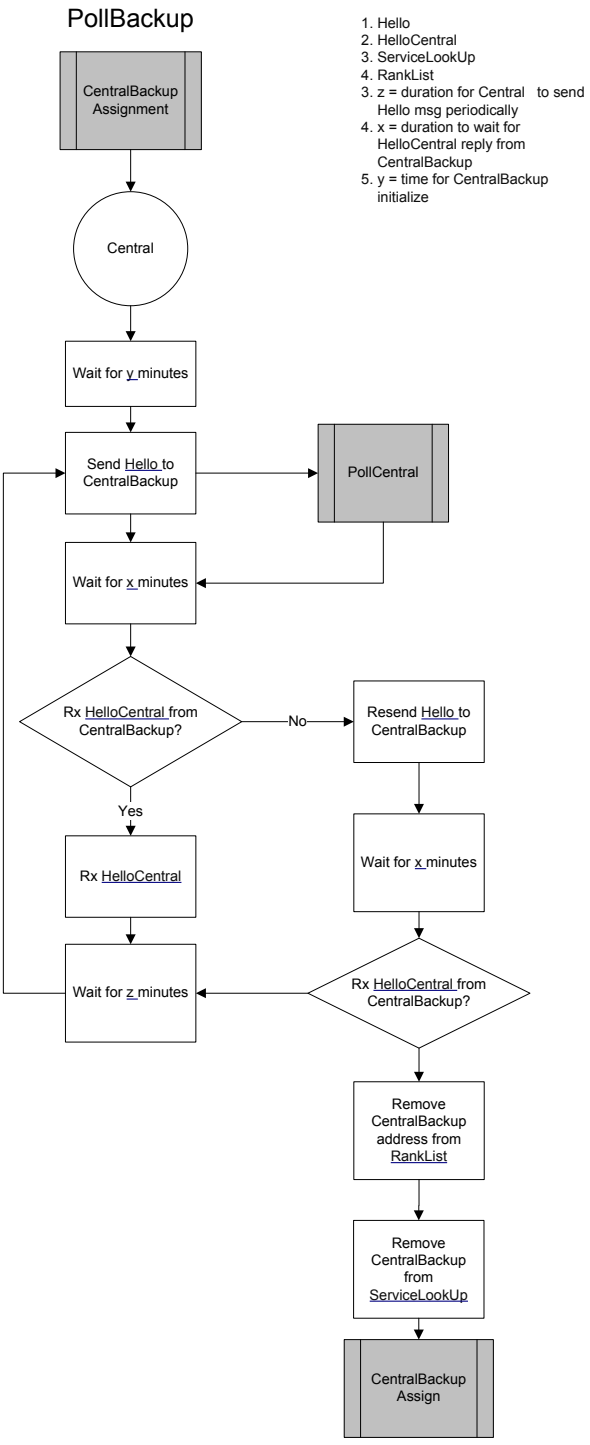

ServiceRegistration
1. ServiceRegistrationRequest
2. ServiceRegistrationReply
3. ServiceLookUp
4. ServiceID = DevId.ServiceTypeId.ServiceId (ServiceId - Running number)
5. maxServiceRegExpiryTime
6. counter = ServiceRegExpiryTime + buffer time
7. counterT = lease renewal time
8. ServiceRegExpiryTimeNotification - Central selects a time and notifies 300D
9. UnsolicitedServiceRegistration
10. t=max time assumed for message propagation from Central to device
DeviceReg
Central
3D/300D
CENTRAL
Rx UnsolicitedService RegistrationReply
Rx ServiceRegistrationRequest
Add Central's services in ServiceLookUp
Send ServiceRegistrationRequest
(maxServiceRegExpiryTime)
Any services to offer?
No
300D?
Yes
No
Yes
Send ServiceRegistration Reply
Address
DeviceType -NULL
Send Acknowledgement
Rx ServiceRegistration Reply ?
ServiceRegExpiryTime < maxServiceRegExpiryTime
No
Send ServiceRegExpiry TimeNotification
Yes
RemoteControl
Send ServiceRegistrationReply
Address
DeviceType
ServiceType
AttributeList
ServiceRegExpiryTime
No
Start counter for every service
Start counter for service counterT
Add to ServiceLookUp
Address
DeviceType
ServiceType
AttributeList
UniqueServiceID
ServiceRegExpiryTime
Send ServiceLookUp to CentralBackup
CentralBackup
Rx ServiceRegExpiry TimeNotification
ServiceUpdate
Adjust counterT = ServiceRegExpiry Time - t value
Provide Unique ServiceID for all the services.
ServiceID = DevId.ServiceType(Id).ServiceId


1. ServiceLookUp
2. ServiceRegExpiryTime
3. DeviceList
4. ServiceLookUp
5. DeviceListUpdate
6. ServiceLookUpUpdate
7. ServiceRegRenew
8. counter = ServiceRegExpiryTime + buffer time
9. t=max assumed time for ServiceRegistrationReply to reach Central
10. counterT = counter + t
11. UnsolicitedServiceRegistration
12. Acknowledgement
13. LeaderElectRequest
ServiceUpdate
ServiceReg
CENTRAL
Is child a 300D?
No
PollSmallDevice
Yes
300D
Rx ServiceRegRenew < counter ?
Yes
Reset counter
No
Send ServiceRegRenew before each service's counterT
Cancel entry from ServiceLookUp
New service definition?
No
Yes
Cancel entry from RankList
Send UnsolicitedService Registration
Yes
Search
Send ServiceLookUpUpdate (set field to indicate 300D)
Rx ServiceRegRenew from deleted device ?
Yes
Rx Acknowledgement ?
Remote Control
CentralBackup
ServiceReg
No
Resend
Rx Acknowledgement ?
No
Broadcast LeaderElect Request
Central Discovery

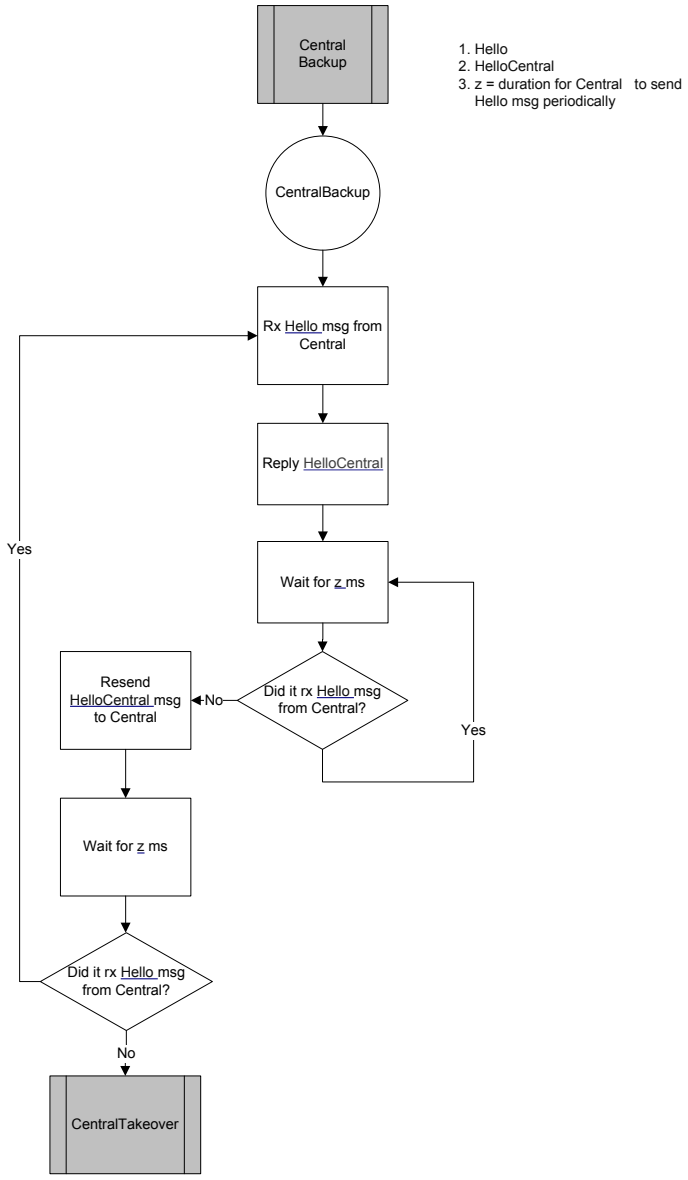
CentralBackupAssignment



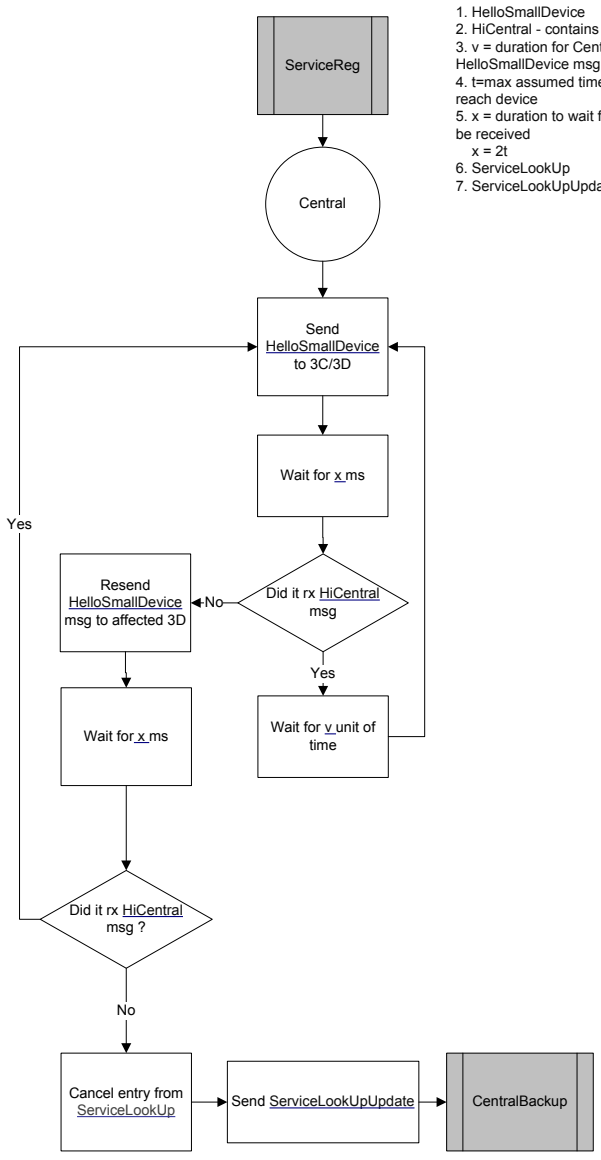

CentralBackup
1. ServiceLookUp
2. ServiceLookUpCopy
3. ServiceLookUpUpdate
4. CentralBackupCancel
5. RankList
6. RankListUpdate
7. RankListCopy
8. CentralSubscriptionTable
9. CentralSubscriptionTableCopy
10. SubscriptionUpdate
CentralBackup Assign
PollCentral
CentralBackup
Rx CentralSubscriptionTable
Rx RankList
Rx ServiceLookUp
Rx CentralBackupCancel ?
No
Yes
Create CentralSubscriptionTableCopy
Create a RankListCopy
Create a ServiceLookUpCopy
Rx CentralBackupCancel from Central
newCentral Assign
Off CentralBackup mode
No
Rx SubscriptionUpdate ?
Yes
Update CentralSubscriptionTableCopy
No
Rx RankListUpdate ?
Yes
Update RankListCopy (new device)
No
Rx ServiceLookUp Update ?
Yes
Update ServiceLookUpCopy
No
Device deleted?
Yes
300D field set?
Yes
Delete from RankListCopy


PollBackup
1. Hello
2. HelloCentral
3. ServiceLookUp
4. RankList
3. z = duration for Central to send Hello msg periodically
4. x = duration to wait for HelloCentral reply from CentralBackup
5. y = time for CentralBackup initialize
CentralBackup Assignment
Central
Wait for y minutes
Send Hello to CentralBackup
PollCentral
Wait for x minutes
Rx HelloCentral from CentralBackup?
No
Resend Hello to CentralBackup
Yes
Rx HelloCentral
Wait for x minutes
Wait for z minutes
Rx HelloCentral from CentralBackup?
Remove CentralBackup address from RankList
Remove CentralBackup from ServiceLookUp
CentralBackup Assign

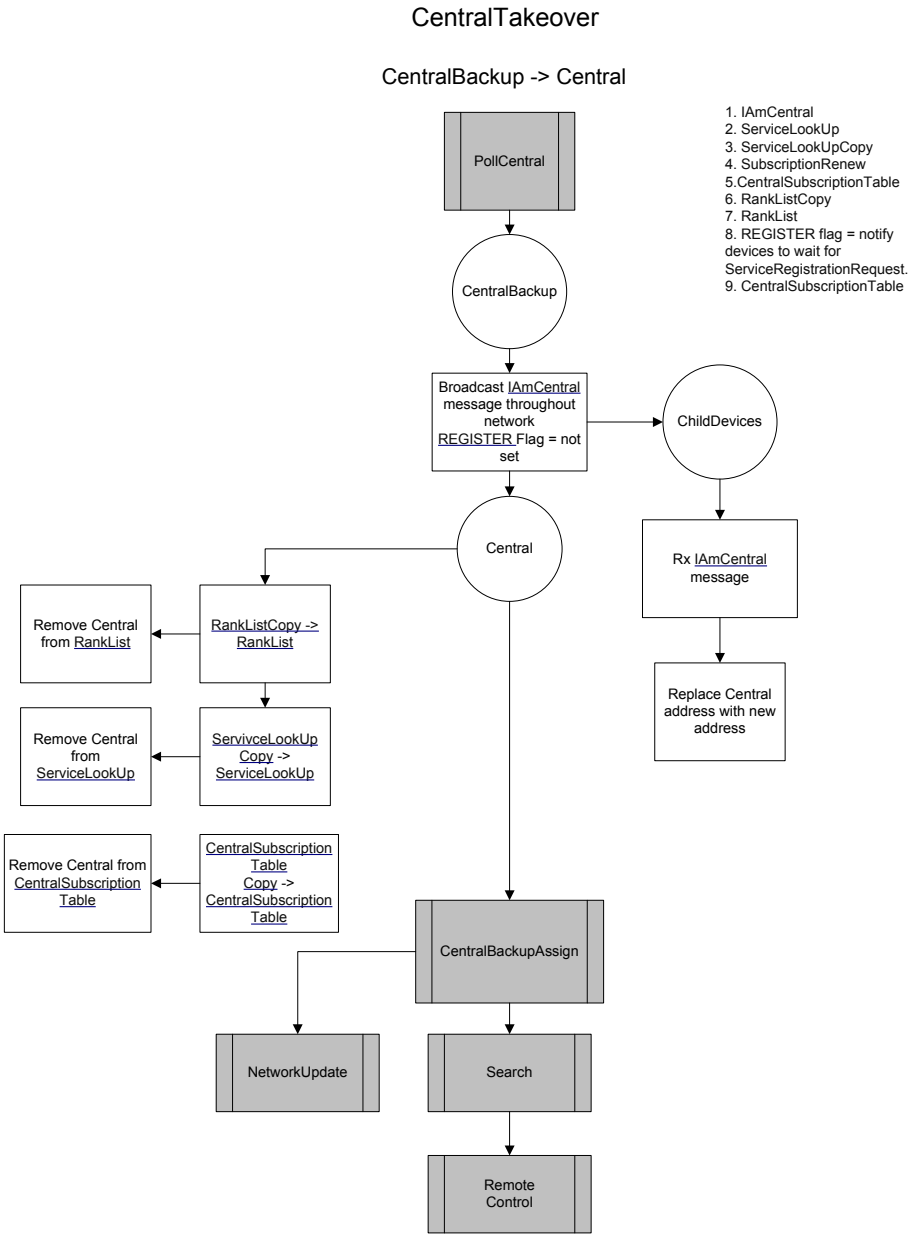
Poll Central

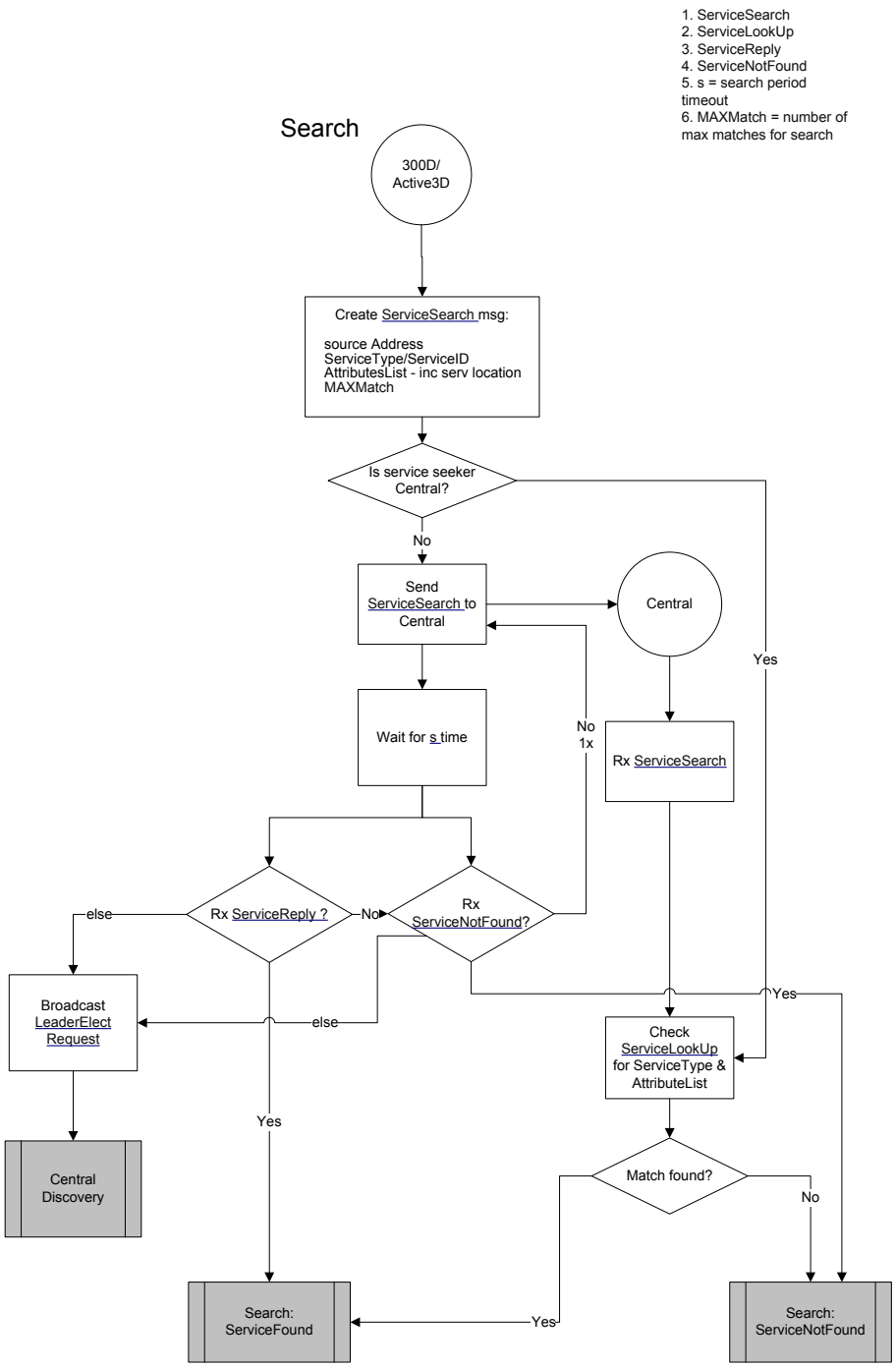


PollSmallDevices

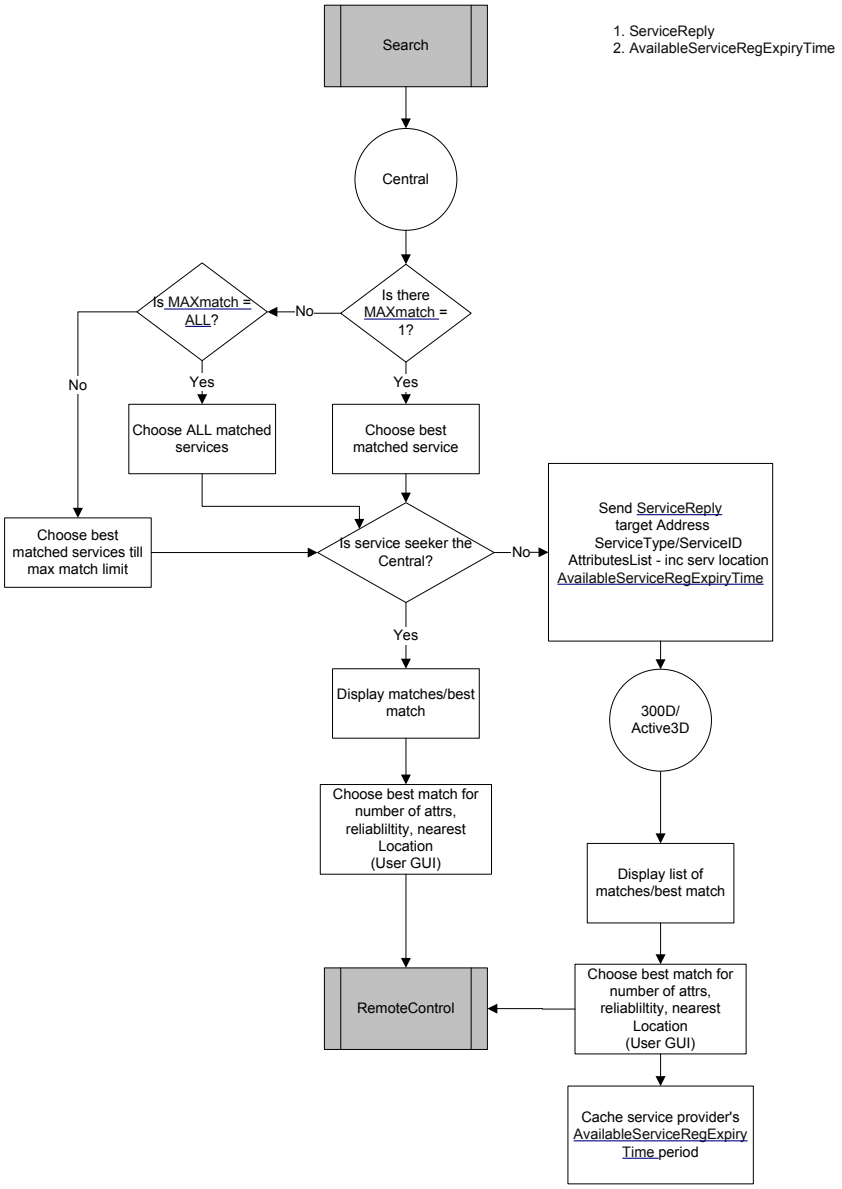


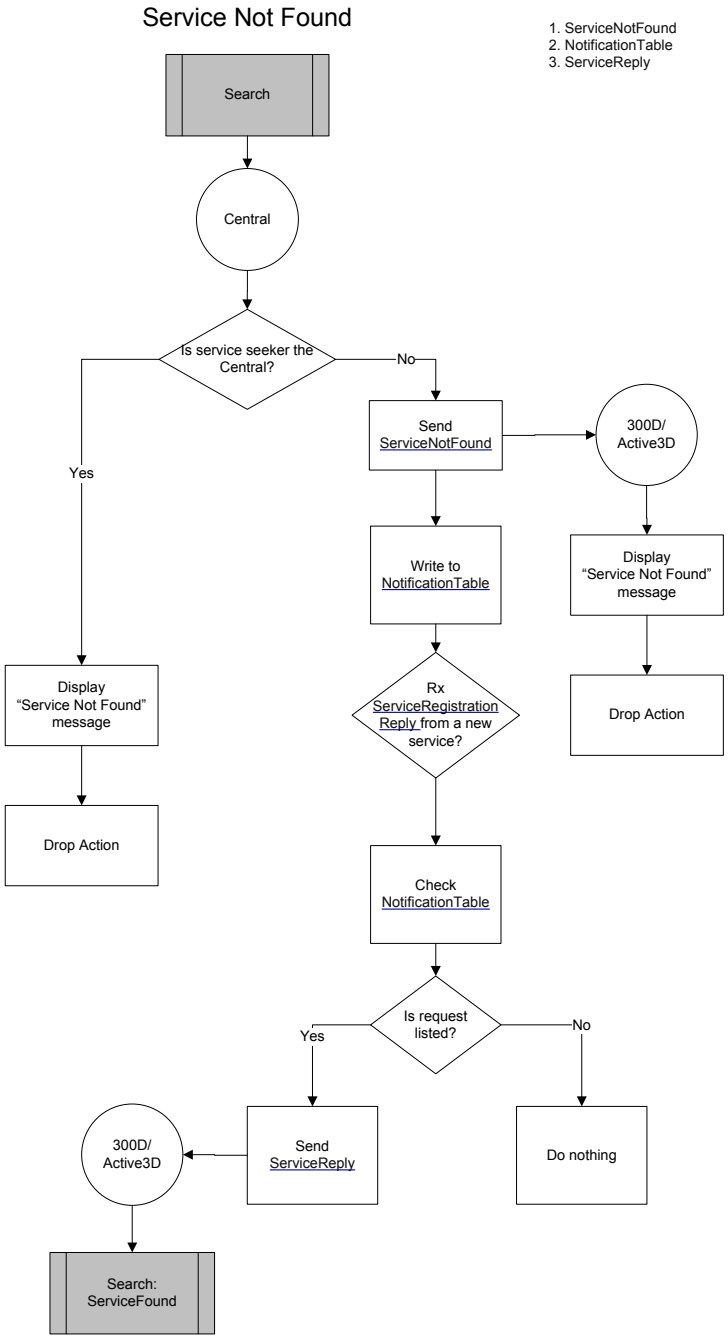
1. HelloSmallDevice
2. HiCentral - contains lease period, v
3. v = duration for Central to send HelloSmallDevice msg periodically (lease)
4. t=max assumed time for HelloSmallDevice to reach device
5. x = duration to wait for HiCentral message to be received
   x = 2t
6. ServiceLookUp
7. ServiceLookUpUpdate

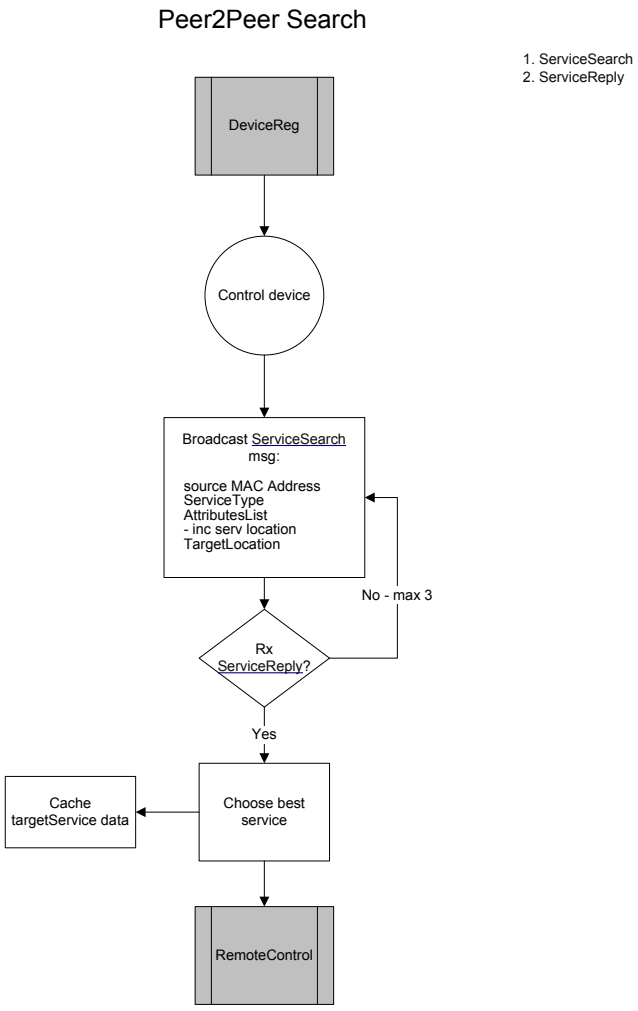

CentralTakeover
CentralBackup -> Central
1. IAmCentral
2. ServiceLookUp
3. ServiceLookUpCopy
4. SubscriptionRenew
5.CentralSubscriptionTable
6. RankListCopy
7. RankList
8. REGISTER flag = notify devices to wait for ServiceRegistrationRequest.
9. CentralSubscriptionTable
PollCentral
CentralBackup
Broadcast IAmCentral message throughout network REGISTER Flag = not set
ChildDevices
Central
Rx IAmCentral message
Replace Central address with new address
Remove Central from RankList
RankListCopy -> RankList
Remove Central from ServiceLookUp
ServivceLookUp Copy -> ServiceLookUp
Remove Central from CentralSubscription Table
CentralSubscription Table Copy -> CentralSubscription Table
CentralBackupAssign
NetworkUpdate
Search
Remote Control

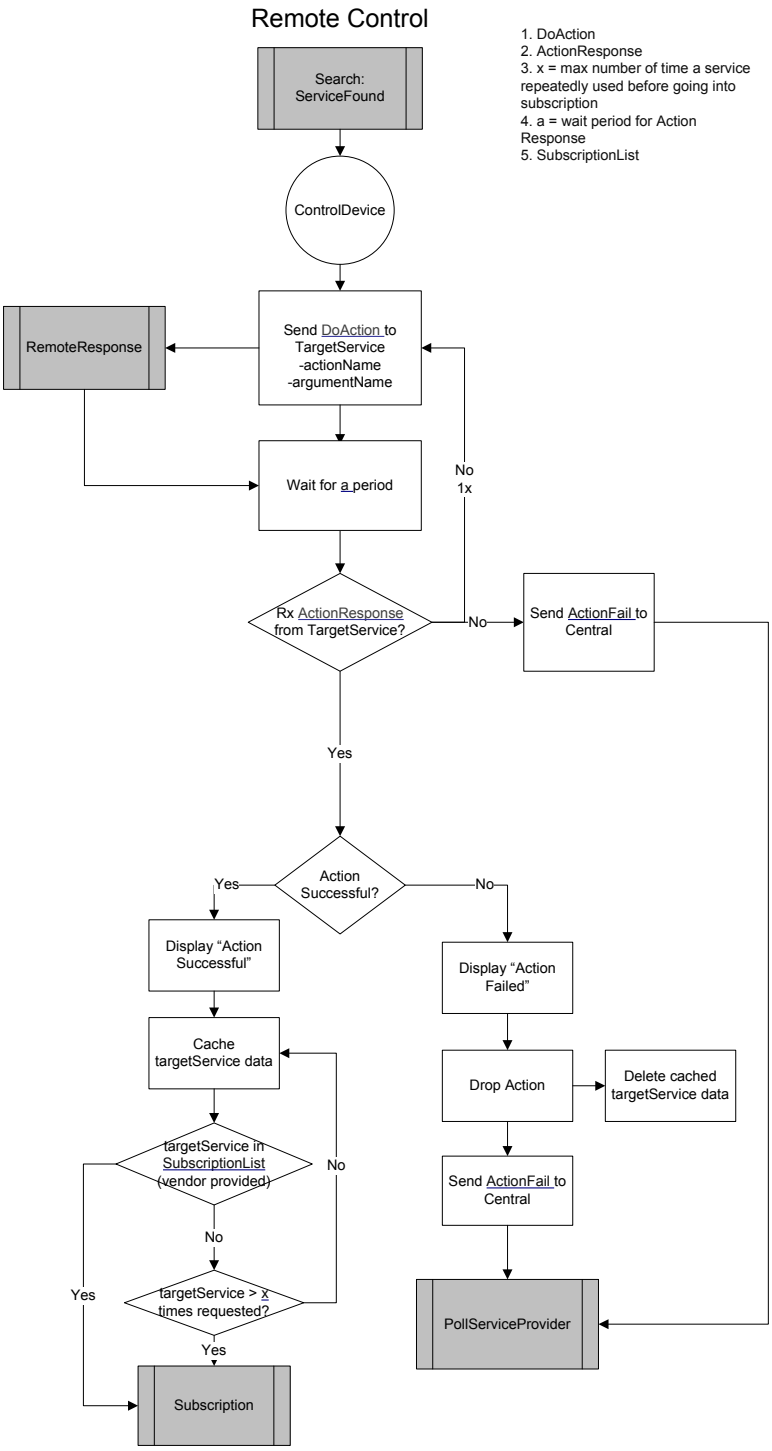

1. ServiceSearch
2. ServiceLookUp
3. ServiceReply
4. ServiceNotFound
5. s = search period timeout
6. MAXMatch = number of max matches for search
Search
300D/ Active3D
Create ServiceSearch msg:
source Address
ServiceType/ServiceID
AttributesList - inc serv location
MAXMatch
Is service seeker Central?
No
Yes
Send ServiceSearch to Central
Central
Wait for s time
No 1x
Rx ServiceSearch
Rx ServiceReply ?
else
No
Rx ServiceNotFound?
Broadcast LeaderElect Request
else
Yes
Yes
Check ServiceLookUp for ServiceType & AttributeList
Central Discovery
Match found?
No
Search: ServiceFound
Yes
Search: ServiceNotFound

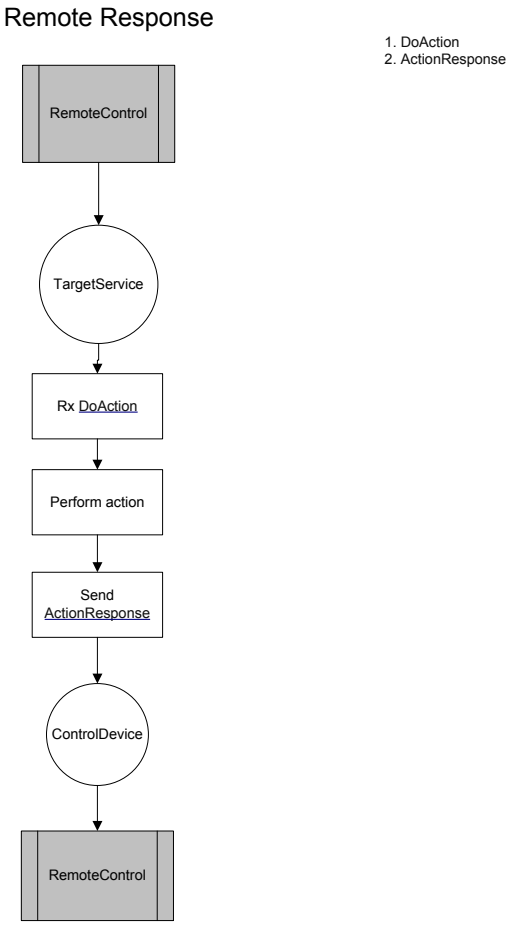
Service Found




Service Not Found
1. ServiceNotFound
2. NotificationTable
3. ServiceReply
Search
Central
Is service seeker the Central?
No
Yes
Send ServiceNotFound
300D/ Active3D
Write to NotificationTable
Display "Service Not Found" message
Drop Action
Display "Service Not Found" message
Drop Action
Rx ServiceRegistration Reply from a new service?
Check NotificationTable
Is request listed?
Yes
No
Send ServiceReply
Do nothing
300D/ Active3D
Search: ServiceFound

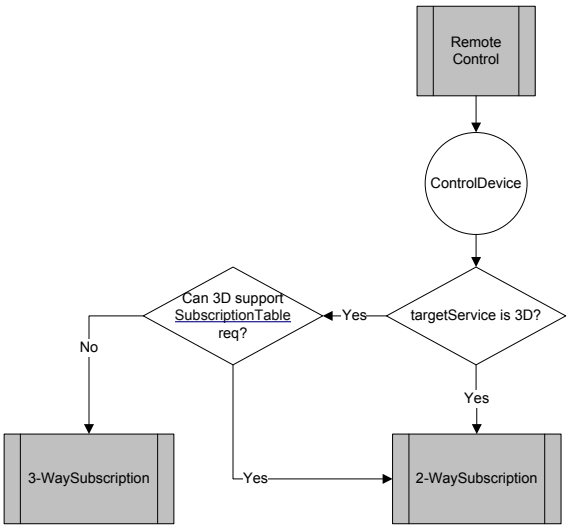

Peer2Peer Search
1. ServiceSearch
2. ServiceReply
DeviceReg
Control device
Broadcast ServiceSearch msg:
source MAC Address
ServiceType
AttributesList
- inc serv location
TargetLocation
No - max 3
Rx ServiceReply?
Yes
Cache targetService data
Choose best service
RemoteControl

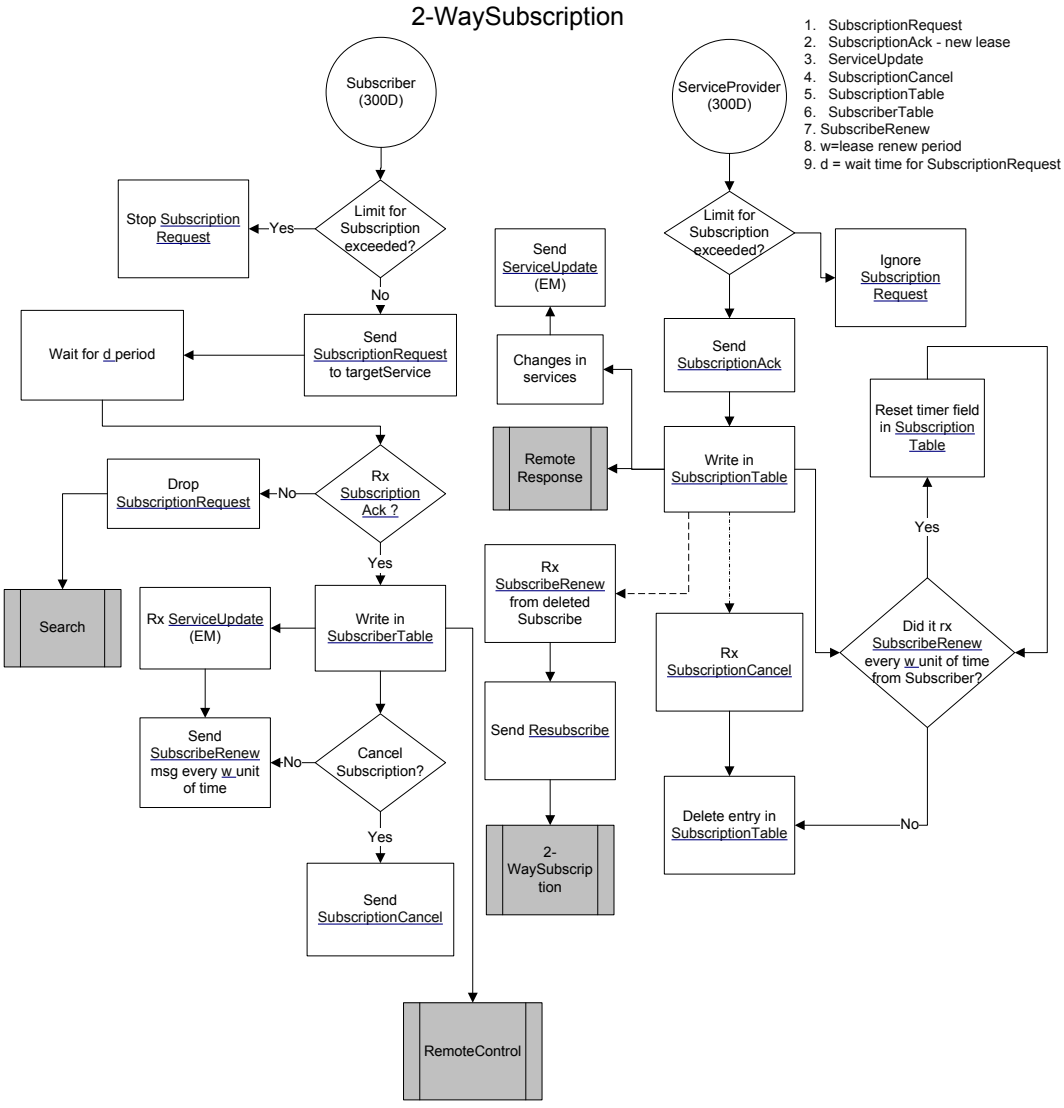

Remote Control
1. DoAction
2. ActionResponse
3. x = max number of time a service repeatedly used before going into subscription
4. a = wait period for Action Response
5. SubscriptionList
Search: ServiceFound
ControlDevice
Send DoAction to TargetService -actionName -argumentName
RemoteResponse
Wait for a period
No 1x
Rx ActionResponse from TargetService?
No
Send ActionFail to Central
Yes
Action Successful?
Yes
No
Display "Action Successful"
Display "Action Failed"
Cache targetService data
Drop Action
Delete cached targetService data
targetService in SubscriptionList (vendor provided)
No
Send ActionFail to Central
No
targetService > x times requested?
Yes
PollServiceProvider
Yes
Subscription

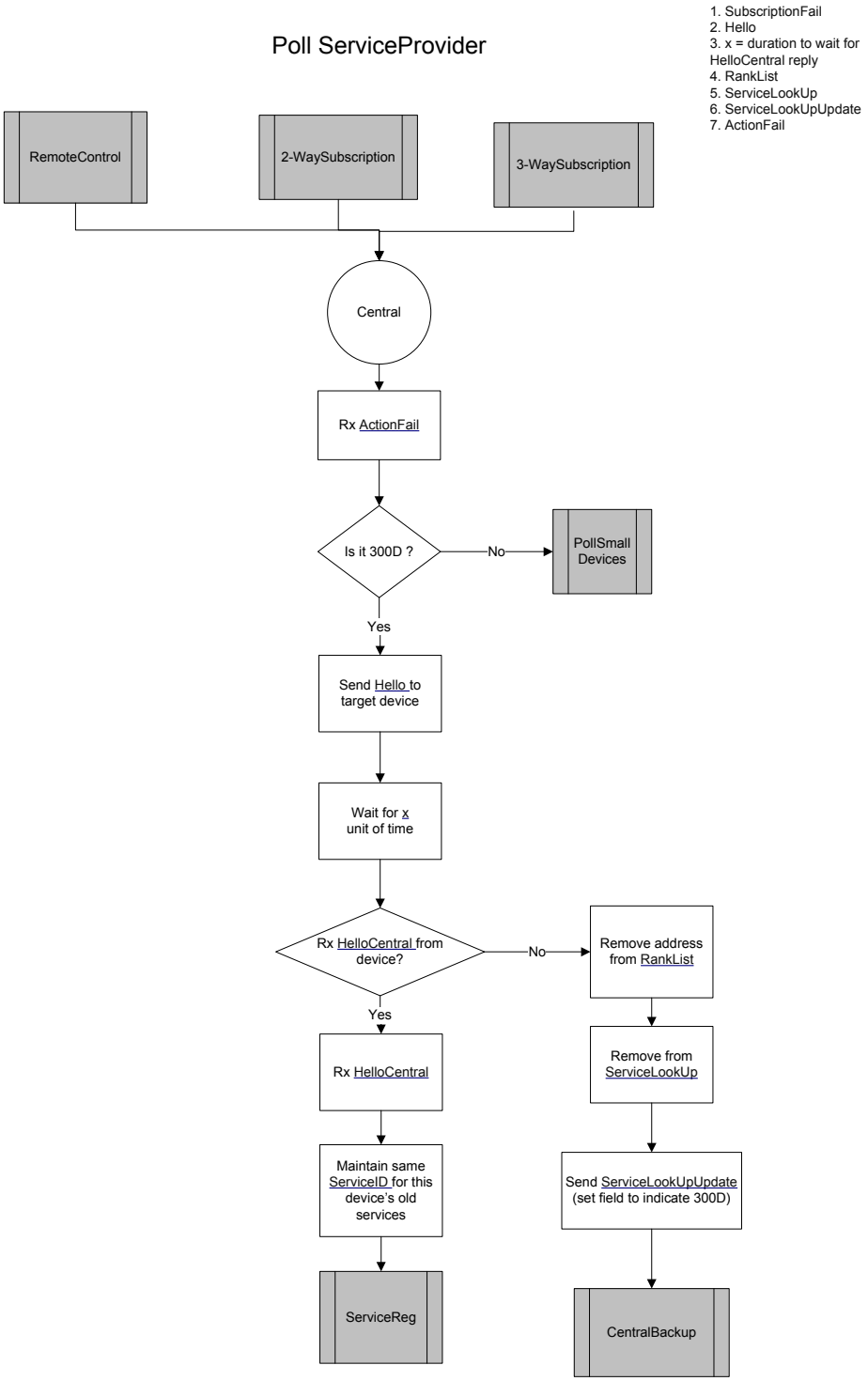

Remote Response
1. DoAction
2. ActionResponse
RemoteControl
TargetService
Rx DoAction
Perform action
Send
ActionResponse
ControlDevice
RemoteControl

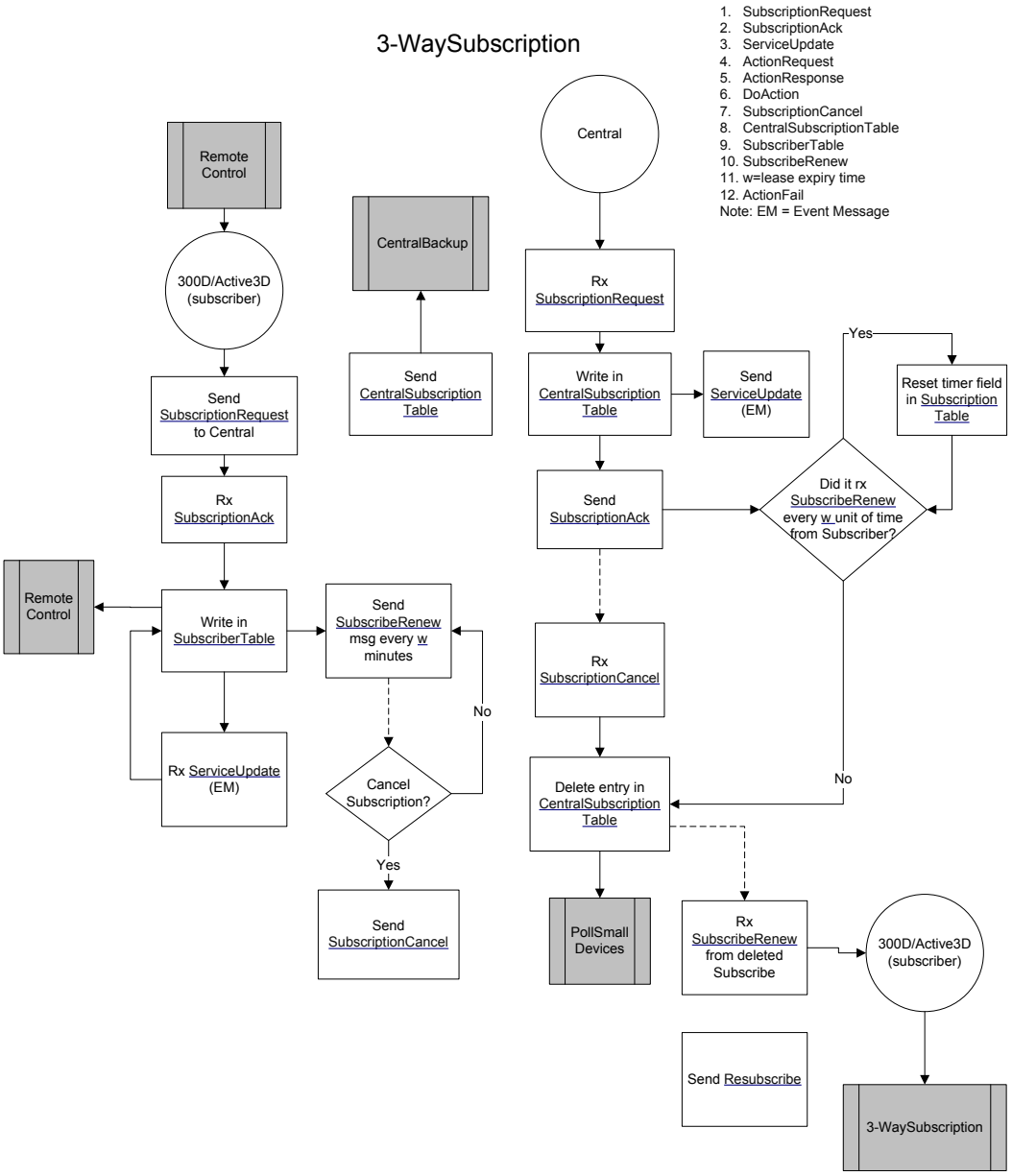
Subscription

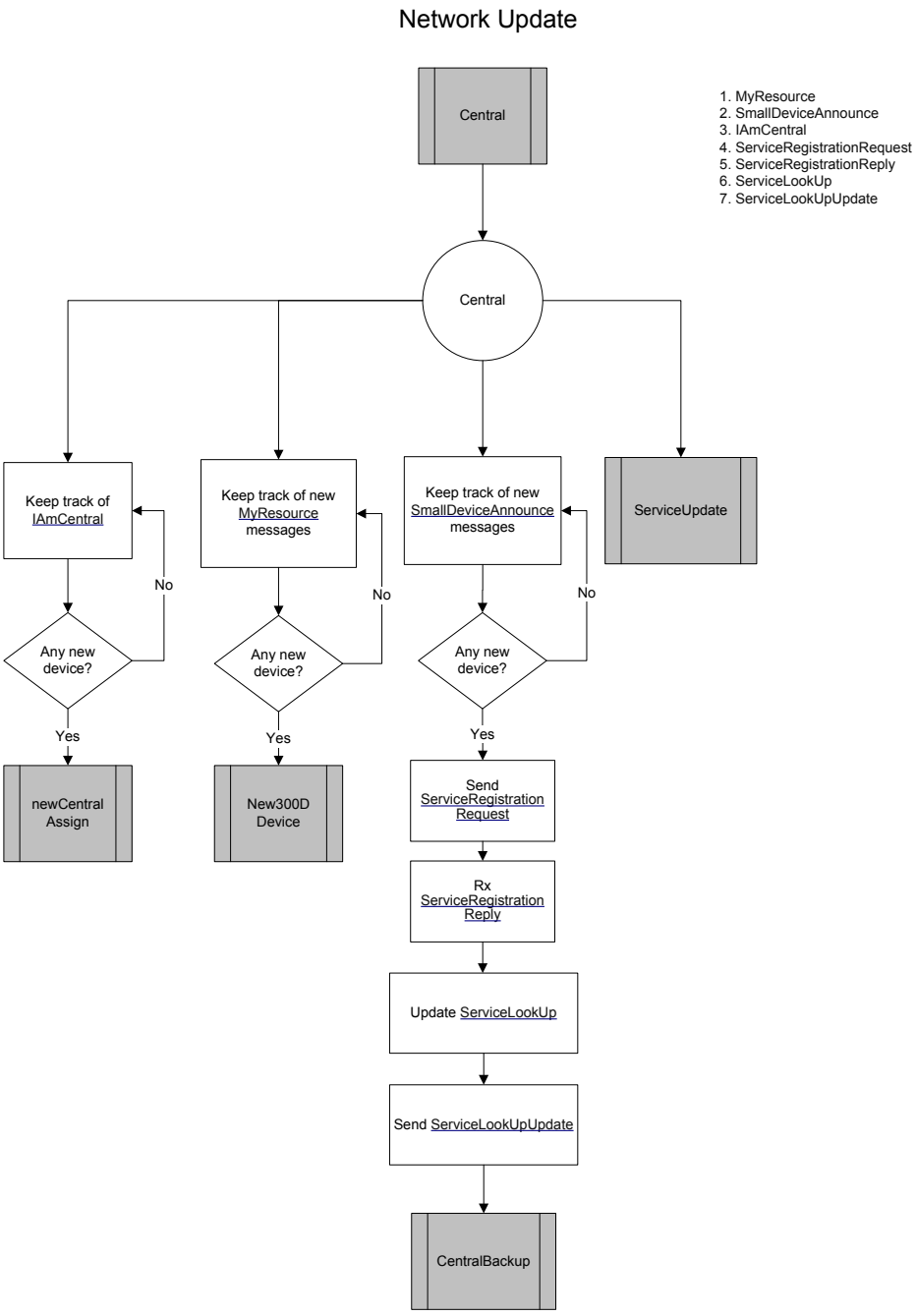
1. SubscriptionTable

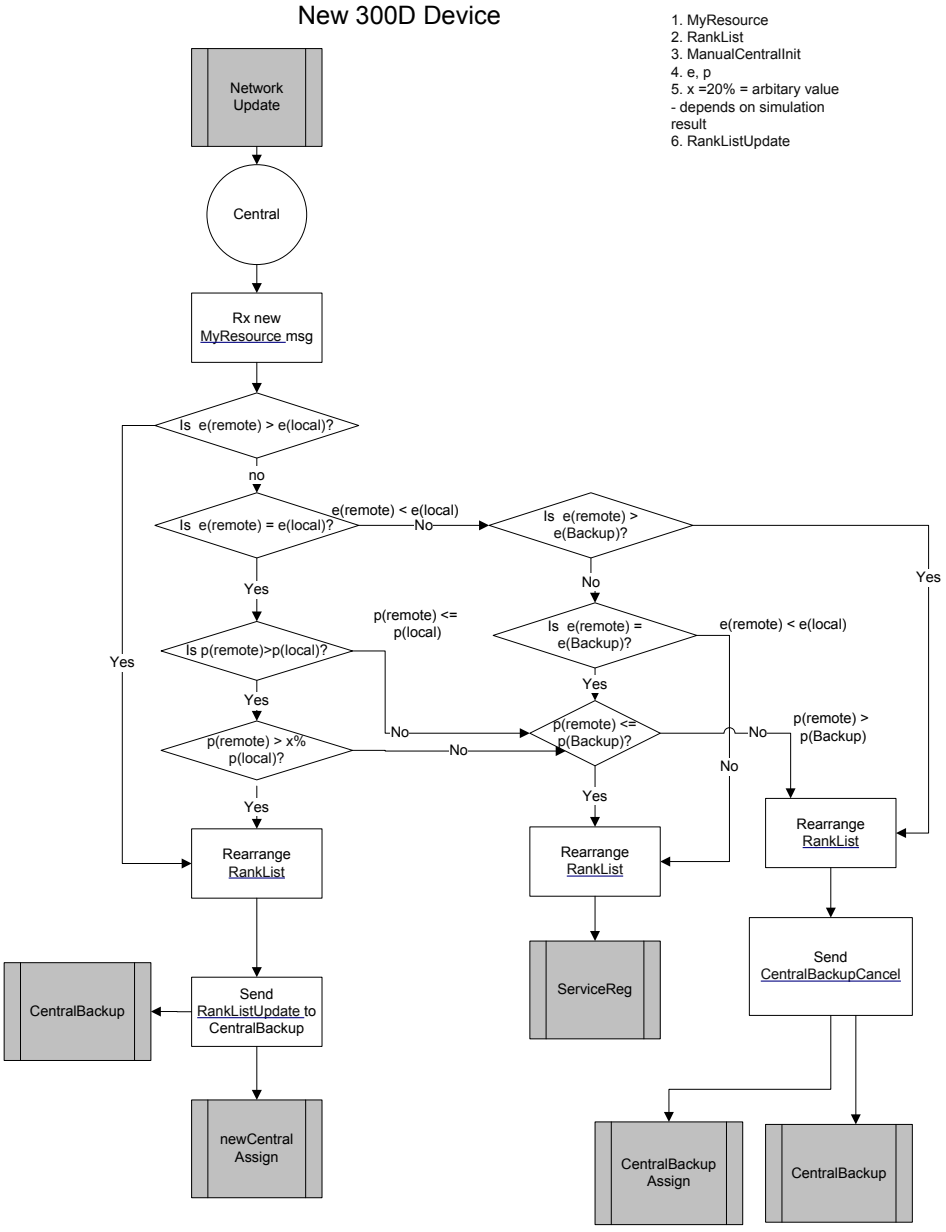



2-WaySubscription
1. SubscriptionRequest
2. SubscriptionAck - new lease
3. ServiceUpdate
4. SubscriptionCancel
5. SubscriptionTable
6. SubscriberTable
7. SubscribeRenew
8. w=lease renew period
9. d = wait time for SubscriptionRequest
Subscriber (300D)
ServiceProvider (300D)
Limit for Subscription exceeded?
Yes
Stop Subscription Request
No
Send SubscriptionRequest to targetService
Wait for d period
Rx Subscription Ack ?
No
Drop SubscriptionRequest
Yes
Search
Write in SubscriberTable
Rx ServiceUpdate (EM)
Send SubscribeRenew msg every w unit of time
Cancel Subscription?
No
Yes
Send SubscriptionCancel
RemoteControl
Limit for Subscription exceeded?
Ignore Subscription Request
Send SubscriptionAck
Write in SubscriptionTable
Remote Response
Send ServiceUpdate (EM)
Changes in services
Rx SubscribeRenew from deleted Subscribe
Send Resubscribe
2-WaySubscription
Rx SubscriptionCancel
Delete entry in SubscriptionTable
Reset timer field in Subscription Table
Yes
Did it rx SubscribeRenew every w unit of time from Subscriber?
No

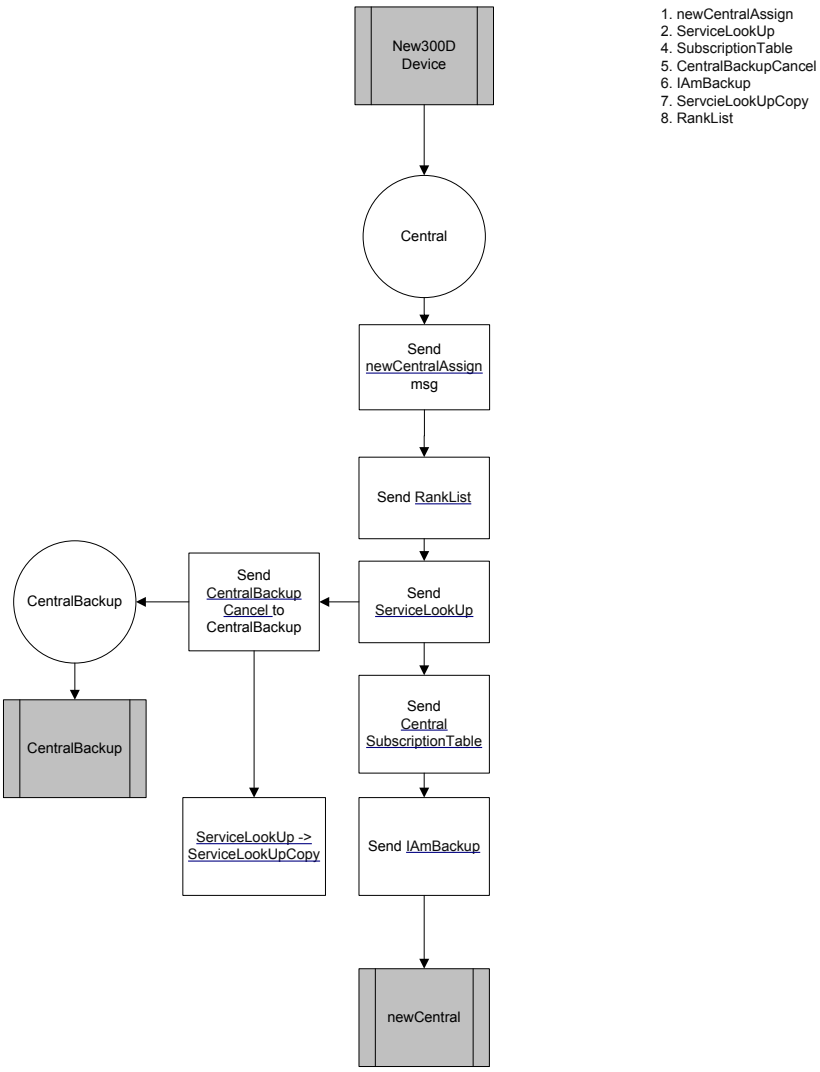

Poll ServiceProvider
1. SubscriptionFail
2. Hello
3. x = duration to wait for HelloCentral reply
4. RankList
5. ServiceLookUp
6. ServiceLookUpUpdate
7. ActionFail
RemoteControl
2-WaySubscription
3-WaySubscription
Central
Rx ActionFail
Is it 300D ?
No
PollSmall Devices
Yes
Send Hello to target device
Wait for x unit of time
Rx HelloCentral from device?
No
Remove address from RankList
Yes
Rx HelloCentral
Remove from ServiceLookUp
Maintain same ServiceID for this device's old services
Send ServiceLookUpUpdate (set field to indicate 300D)
ServiceReg
CentralBackup

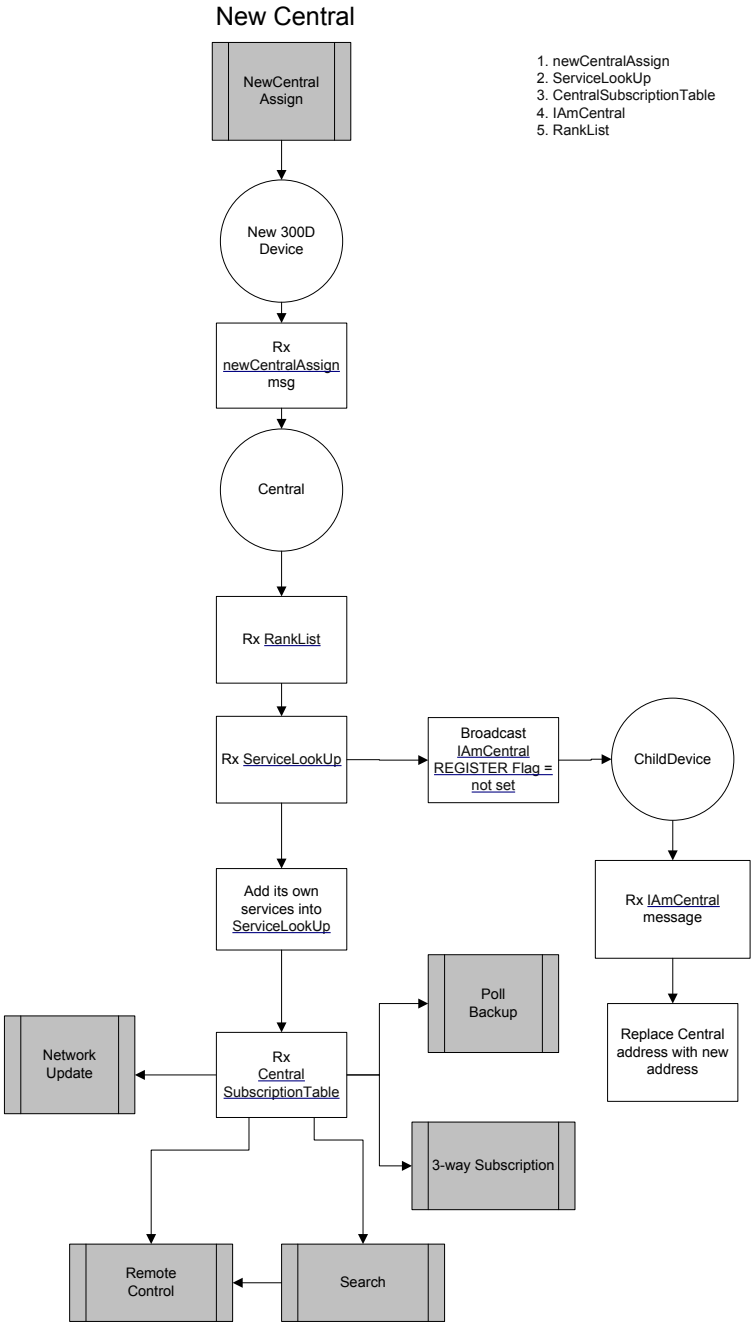

3-WaySubscription
1. SubscriptionRequest
2. SubscriptionAck
3. ServiceUpdate
4. ActionRequest
5. ActionResponse
6. DoAction
7. SubscriptionCancel
8. CentralSubscriptionTable
9. SubscriberTable
10. SubscribeRenew
11. w=lease expiry time
12. ActionFail
Note: EM = Event Message
Remote Control
300D/Active3D (subscriber)
Send SubscriptionRequest to Central
Rx SubscriptionAck
Remote Control
Write in SubscriberTable
Rx ServiceUpdate (EM)
Send SubscribeRenew msg every w minutes
Cancel Subscription?
No
Yes
Send SubscriptionCancel
CentralBackup
Send CentralSubscription Table
Central
Rx SubscriptionRequest
Write in CentralSubscription Table
Send ServiceUpdate (EM)
Send SubscriptionAck
Yes
Reset timer field in Subscription Table
Did it rx SubscribeRenew every w unit of time from Subscriber?
Rx SubscriptionCancel
No
Delete entry in CentralSubscription Table
PollSmall Devices
Rx SubscribeRenew from deleted Subscribe
300D/Active3D (subscriber)
Send Resubscribe
3-WaySubscription


Network Update
Central
1. MyResource
2. SmallDeviceAnnounce
3. IAmCentral
4. ServiceRegistrationRequest
5. ServiceRegistrationReply
6. ServiceLookUp
7. ServiceLookUpUpdate
Central
Keep track of IAmCentral
Keep track of new MyResource messages
Keep track of new SmallDeviceAnnounce messages
ServiceUpdate
No
No
No
Any new device?
Any new device?
Any new device?
Yes
Yes
Yes
newCentral Assign
New300D Device
Send ServiceRegistration Request
Rx ServiceRegistration Reply
Update ServiceLookUp
Send ServiceLookUpUpdate
CentralBackup


New 300D Device
1. MyResource
2. RankList
3. ManualCentralInit
4. e, p
5. x =20% = arbitary value - depends on simulation result
6. RankListUpdate
Network Update
Central
Rx new MyResource msg
Is e(remote) > e(local)?
no
Is e(remote) = e(local)?
e(remote) < e(local)
No
Is e(remote) > e(Backup)?
Yes
No
Yes
Is p(remote)>p(local)?
p(remote) <= p(local)
Is e(remote) = e(Backup)?
e(remote) < e(local)
Yes
Yes
p(remote) > x% p(local)?
No
No
p(remote) <= p(Backup)?
No
p(remote) > p(Backup)
No
Yes
Yes
Rearrange RankList
Rearrange RankList
Rearrange RankList
ServiceReg
Send CentralBackupCancel
CentralBackup
Send RankListUpdate to CentralBackup
newCentral Assign
CentralBackup Assign
CentralBackup

New Central Assign




New Central
NewCentral Assign
1. newCentralAssign
2. ServiceLookUp
3. CentralSubscriptionTable
4. IAmCentral
5. RankList
New 300D Device
Rx newCentralAssign msg
Central
Rx RankList
Rx ServiceLookUp
Broadcast IAmCentral REGISTER Flag = not set
ChildDevice
Rx IAmCentral message
Replace Central address with new address
Add its own services into ServiceLookUp
Poll Backup
Network Update
Rx Central SubscriptionTable
3-way Subscription
Remote Control
Search