On Consistency Maintenance In Service Discovery

V. Sundramoorthy, P.H. Hartel, J. Scholten
University of Twente
Enschede, The Netherlands
vasughi.sundramoorthy@utwente.nl

Abstract

Communication and node failures degrade the ability of a service discovery protocol to ensure Users receive the correct service information when the service changes. We propose that service discovery protocols employ a set of recovery techniques to recover from failures and regain consistency. We use simulations to show that the type of recovery technique a protocol uses significantly impacts the performance. We benchmark the performance of our own service discovery protocol, FRODO against the performance of first generation service discovery protocols, Jini and UPnP during increasing communication and node failures. The results show that FRODO has the best overall consistency maintenance performance.

1. Introduction

A service discovery protocol has two types of entities: *User* and *Manager*. A Manager is a service provider, which has a set of services. Each service is represented as a *Service Description (SD)*, which describes the service in terms of: (1) device type (e.g. printer), (2) service type (e.g. color printing) and (3) attribute list (e.g. location, paper size). A User is an entity that has a set of requirements for the services it needs. There are two main types of service discovery architectures: *registry-based* (e.g. Jini [6]) and *peer-to-peer* (e.g. UPnP [5]). A registry-based architecture has a third entity, called the *Registry*. A Manager *registers* its services at a Registry, and Users discover the services through unicast *queries* to the Registry. The peer-to-peer architecture has no Registries, and Users discover Managers through broadcast or multicast queries. The registry-based architecture reduces network traffic and makes a network more manageable by allowing Registries to keep track of arriving and departing services. The peer-to-peer architecture avoids single point of failure problems, as may exist in the registry-based architecture, but increases network traffic. A hybrid of these two architectures can be implemented to allow the protocol to be more resilient against failure on the Registry, while reducing network traffic (e.g., SLP [1] and FRODO [8]).

Users typically cache the discovered service description to reduce the access time to the service, and reduce bandwidth usage (by avoiding repeated queries to rediscover the service). The tradeoff to caching is the need to maintain a consistent view of the service. *Polling* for updates (pull model), and *notification* by the Manager when the service changes (push model) are two consistency maintenance mechanisms in service discovery. However, communication and node failures may cause the consistency maintenance mechanism to fail to update the Users. We find that as communication and node failures increase, a number of failure scenarios are created. How well a service discovery protocol performs depends on the type of recovery technique that the protocol adopts when dealing with each failure scenario. We classify consistency maintenance recovery techniques according to the failure scenarios, and propose techniques that improve performance.

Contribution. This paper contributes to the research and development of service discovery systems by (1) providing a novel classification and detailed analysis of consistency maintenance recovery techniques, according to failure scenarios, (2) comparing recovery techniques in state of the art service discovery protocols (FRODO, UPnP and Jini), and (3) showing that by employing a combination of selected recovery techniques, FRODO has the best overall consistency maintenance performance.

Section 2 provides a brief overview of FRODO. Section 3 describes the consistency maintenance fundamentals including the principles, mechanisms, recovery techniques and the performance metrics. Section 4 presents how we have modeled FRODO based on existing UPnP and Jini models, and explains the experimental design. Section 5 compares the performance of FRODO against UPnP and Jini, and discusses the impact of the recovery techniques we present in Section 3 on the consistency maintenance performance. Section 6 concludes.

2. FRODO

A home environment differs in two aspects from the professional connected environment. These are:

- Resource-awareness - Cost of devices is a prime factor for the home user. New sophisticated technologies should not demand too much additional resources on existing services, and raise cost.

- Robustness - Unlike the professional environment, the home environment does not have the luxury of a network administrator to monitor and resolve network disturbances. Home owners should not be restricted in how they manage their appliances (unplugging, moving).

Thus resource-awareness and robustness are two main objectives for service discovery in the home environment.

To satisfy resource-awareness, FRODO introduces device classification: (1) 3C (Cent) device class - simple devices with restricted resources (e.g. simple sensors). Nodes in this class are only Managers. (2) 3D (Dollar) device class- medium complex devices (e.g. temperature controller). A node in this class can be a Manager and a User with limited behaviors and (3) 300D (Dollar) device class - powerful devices, controlled by a complex embedded computer. A node in this class can be a Manager, a User and a Registry (e.g. set-top boxes).

To address robustness, the system is made resilient to single point of failure problems through a *leader election protocol.* The 300D nodes elect the most powerful node as the Registry. We call the Registry the *Central*, because besides being the repository for service descriptions, the Central also actively monitors the system for new and defunct nodes, and responds according to their device classes. A Backup is appointed by the Central to store configuration information. The Backup takes over automatically in case of Central failure.

Unlike first generation service discovery systems, FRODO does not depend on the recovery abilities of lower layer protocols (such as TCP) to perform its tasks. This allows the protocol to be deployed together with leaner lower layer protocol stacks, with little or no error recovery mechanisms.

3. Fundamentals of Consistency Maintenance

Users become consistent with the Manager when they successfully receive the correct update information from the Manager. To explain update information, we use an example of a Manager which offers a printing service. The service description is a list of attribute-value pairs.

```
SD = {DeviceType=Printer,
        ServiceType=ColorPrinter,
        AttributeList{PaperSize=A4,
        Location=Study}}
```

If the structure, or information in the attribute-value pair changes, the service description changes. For example, the “ServiceType” changes from “ColorPrinter” to “Black&WhitePrinter”. In the registry-based architecture, the printer updates the Users via the Registry, while in the peer-to-peer architecture, the printer updates the interested Users directly.

3.1 Consistency maintenance principles

Service discovery complies with *eventual consistency*, because it is client-centric, which tolerates transient inconsistencies, just like Bayou [7], the distributed database system for mobile users. For discovered services to be useful, it is important that the consistency guarantees are specified clearly. We specify the requirements for consistency maintenance in service discovery protocols in the *Service Discovery Principles* [9], where the Configuration Update Principles require the User and/or Registry to *always eventually* regain consistency with the Manager after the service changes. The User detects the change in the Manager, and regains consistency by obtaining the correct view of the service, either from the Manager directly (2-party Configuration Update Principle), or via the Registry (3-party Configuration Update Principle). The principles hold true only when there is *connectivity* among the communicating entities (e.g., valid communication paths). The term *always eventually* states that a successful update invariably takes place at some time in the future, without giving a concrete time constraint.

3.2. Consistency maintenance mechanisms

Before we delve deeper into the issues facing consistency maintenance in an environment with communication and node failures, we first introduce the basic mechanisms that existing service discovery protocols implement. The User has to *subscribe* either directly to the Manager (*2-party subscription*) or to a Registry (*3-party subscription*) to receive updates. A subscription between the User and the Manager or between the User and the Registry remains valid as long as the subscription *lease* does not expire. To maintain a valid subscription lease, Users are required to send messages periodically to the lessee to indicate their continued interest with the service.

There are two basic consistency maintenance mechanisms:

(CM1) *Notification (push-based update)* - The User receives an update when the service description changes. In 3-party subscription, the Manager notifies the Registry which then propagates the update to subscribed Users. In 2-party subscription, the Manager notifies subscribed Users directly. Update notification is a built-in mechanism in a service discovery protocol. Examples of state of the art protocols that have this capability are FRODO, UPnP and Jini.

(CM2) *Polling (pull-based update)* - The User regains consistency by polling the Manager or the Registry to retrieve the updated service description. In 3-party subscription, the Manager updates the Registry by re-registering its services. In both subscription schemes, periodic queries from the User eventually retrieve the updated service description. Typically, polling is implemented in the application layer, with hooks from the service discovery protocol. In FRODO, UPnP, Jini and SLP, polling is implemented by requiring the User to query the service periodically.

Dabrowski and Mills [3] show that periodic polling is the more effective method if the application allows persistent polling (even when the lower protocol layers signal a connection failure), therefore increasing the chances for the User to retrieve the updated service description eventually. However, Dabrowski and Mills show that polling is a slower mechanism than update notification because of the dependency on the period of polling. We find that polling is also a less efficient mechanism than update notification in scenarios where services rarely change, causing multiple redundant polls. Furthermore, unlike update notification, polling is an application-dependent approach (e.g., frequency of poll) and does not reflect the actual built-in consistency maintenance capability of service discovery protocols. Thus, in this paper, we focus only on issues and performance of consistency maintenance through notification to assess the ability of the protocols (FRODO, UPnP and Jini) to recover from failures and regain consistency.

During update notification, the Manager updates the Users by: (1) propagating an *invalidation* message that indicates that the service has been updated. In UPnP, the Manager notifies the interested User that a change has occurred, whenever the service changes. Consecutive polling by the User retrieves the updated data. This method is efficient for a service that has frequent updates, but causes unwanted redundancy and delay for services that rarely change. (2) Propagating the *updated data*, as used in Jini and FRODO. This method is fast and efficient for a service that changes infrequently. An adaptive method that dynamically switches between sending an invalidation message or sending the update can be implemented, as done in the Alex protocol [2], a filesystem that adapts the type of update propagation based on the age of the file (it assumes older files are less likely to be modified than younger files). However, to our knowledge, no existing service discovery protocols adopt the adaptive mechanism, due to the complexity in implementation.

FRODO is unique because it implements both 2-party (for 300D Managers) and 3-party (for 3D/3C Managers) subscriptions. The User is able to detect which subscription process to use, based on the device class of the Manager.

3.3. Recovery techniques for consistency maintenance

Due to temporary communication failures or node failures, notification of updates may fail. Nevertheless, when connectivity among entities is restored, the service discovery system is expected to recover and regain consistency, as stated by the Configuration Update Principles. Our in-depth analysis of the behavior of service discovery during communication and node failures results in a novel method of identifying, classifying and proposing new recovery techniques based on the type of update and failure scenario. Table 1 is a summary of our classification of recovery techniques.

During communication and node failures, the subscription between the entities may remain valid, even though update notification fails. This is because the participating entities may face short-term failures, and restore connectivity before the subscription lease expires. Hence, it is up to the continuing subscription process to ensure Users regain consistency. We call this type of recovery *subscription-recovery.* When the subscription lease expires, consistency maintenance depends on the inherent capability of the service discovery protocol to detect, and rediscover purged nodes and services. Hence, this type of recovery is called *purge-rediscovery.*

Table 1. Classification of recovery techniques for consistency maintenance. Subscription-recovery techniques for each type of update take effect when subscription still remains valid. Purge-rediscovery techniques occur when subscription expires, and may coincide based on the failure scenario.

Subscription-recovery		Purge-rediscovery	
Update scenario	Recovery technique	Purge and rediscover scenario	Recovery technique
Critical update	SRC1	Manager rediscovers the Registry (and vice-versa)	PR1
	SRC2	User rediscovers the Registry	PR2
Non-critical update	SRN1	Registry rediscovers the User	PR3
	SRN2	Manager rediscovers the User	PR4
		User rediscovers the Manager	PR5

1. Subscription-recovery. The success of consistency maintenance using this type of recovery depends on how persistently the subscription process tries to update the User. The degree of persistence in notifying updates depends on the type of update scenario: *critical update* and *non-critical update*. This is because not all applications require the same level of persistence in sending and receiving updates. By specifying the update scenario, we isolate necessary techniques for successful consistency recovery.

Critical update. This update scenario applies to services that are critical, and need correct information urgently. An example is a fire alarm Manager that changes the value of an attribute, "status" from "ON" to "OFF", and is required to update the PDA of the homeowner. For critical updates, we propose two types of recovery techniques.

(SRC1) Acknowledgements and retransmissions of notification - Update notifications sent by the Manager or the Registry must be acknowledged to indicate success or failure. We propose no retransmission limit for the notification messages. Retransmission is only stopped when (a) the subscription expires, (b) acknowledgement for the notification is received, or (c) the application layer indicates loss of connectivity. Update retransmissions can be spaced in a periodic manner, until acknowledged by the Registry or the User.

(SRC2) Active User and Registry monitoring of updates - This technique takes effect if the User requires a history of missed updates from the Manager. The User and the Registry monitor either the sequence number on the update notifications, or the time period for the next notification (the latter applies only to Managers that provide fixed, periodic updates). When an expected update is missed, the User or the Registry requests the update. The Manager caches the history of service changes and only purges the cached updates after all interested Users successfully obtained the complete view of the service.

Non-critical update. Unlike the critical update scenario, the non-critical update scenario applies to services that are not sensitive to, or not overly affected by missed updates. An example is a printer Manager that updates a User when its paper tray empties. We propose the following recovery techniques to improve consistency maintenance performance.

(SRN1) Acknowledgements and retransmissions of notification - Update notifications sent by the Manager or the Registry are acknowledged to indicate success or failure. Retransmissions of unsuccessful notification is done until either (a) retransmission limit is reached, (b) acknowledgement is received, (c) the subscription expires, (d) the application layer indicates lack of connectivity, or (e) the service changes again, requiring the Manager to reset the notification process.

(SRN2) Future retry of unsuccessful notification - This technique occurs after SRN1 fails to update the User. The Manager caches information on inconsistent Users and retries notification once a message from the inconsistent User is received (such as the subscription lease renewal message). The status of the inconsistent User is cached until (a) the subscription expires, (b) the service changes again, requiring the Manager to reset the notification process, or (c) the update is acknowledged.

The Configuration Update Principles only require the User to regain consistency eventually, but not necessarily recover particular values of previously missed updates. Therefore recovering updates caused by multiple changes are not treated in the non-critical update scenario.

2. Purge-rediscovery. The success of consistency maintenance using this type of recovery depends on the proficiency of the service discovery protocol to detect, register and rediscover nodes and services after the subscription expires. We propose the following recovery techniques for the User to regain consistency, based on the "purge" scenario. A combination of purge-rediscovery techniques take effect if several failure scenarios occur simultaneously.

(PR1) The Manager purges the Registry, or vice-versa: the Manager and the Registry rediscover each other through (a) the Registry's periodic broadcast/multicast announcement, or (b) the Manager's periodic broadcast/multicast announcement (here, the Registry contacts the Manager). When the Manager re-registers, the Registry notifies interested Users of the new registration. The User regains consistency from the Registry notification. Users receive notifications of new service registrations by explicitly requesting for service notification, when they first establish contact with the Registry.

(PR2) The User purges the Registry: the User rediscovers the Registry through (a) the periodic Registry announcement, or (b) the User's periodic broadcast/multicast announcement (here, the Registry contacts the User). The User then queries the Registry for the required service to regain consistency with the Manager (provided that the Manager registers the updated service description).

(PR3) The Registry purges the User: subsequent lease renewal from the User to the Registry results in a resubscription process, where the User then receives the updated service description from the Registry.

(PR4) The Manager purges the User: subsequent messages received from the purged User allows a resubscription process, where the User then receives the updated service description.

(PR5) The User purges the Manager: the User can purge the Manager when the service lease expires, or when the Registry notifies the User when it purges the Manager. The User purges the Manager only if the application layer is not communicating with the Manager. The User rediscovers the Manager through (a) broadcast/multicast query with its requirements, where the matching Manager replies with the updated service description, or (b) broadcast/multicast periodic announcement of the Manager, where the User then queries the Manager for the service description, or (c) unicast query to the Registry for the service

During communication failure through message loss [10], retransmissions and acknowledgements through SRC1 and SRN1 are useful, as long as subscription remains valid. SRC2 and SRN2 are necessary for satisfying the eventual consistency guarantee in the Configuration Update Principles [9]. In Section 5, we show that during short-term interface and node failures (where nodes recover from failures before the subscription expires), SRN2 is the most effective technique. When the subscription expires, PR5 in 2-party subscription is found to be most effective, where Users can listen to the broadcast/multicast announcement of the Manager, and retrieve the updated service. PR1 is beneficial in 3-party subscription, where the Registry notifies the Users when the Manager registers.

3.4. Consistency maintenance through notification in Jini, UPnP and FRODO

We now compare the consistency maintenance of Jini, UPnP and FRODO. Jini uses 3-party subscription. The Manager sends an update to the Registry, and receives an acknowledgement. The Registry propagates the update to the subscribed Users. In UPnP, the Manager sends update notifications to the subscribed Users through 2-party subscription. The notification (invalidation message) indicates only that the service has changed. A User receives the actual update after it requests the change. In both Jini and UPnP, a message is sent only if the reliable transmission using TCP successfully sets up a connection between the sender and the receiver. Messages for setting up the connection and notifying the update are acknowledged and retransmitted, as part of the TCP behavior.

FRODO with 3-party subscription supports resource lean 3D and 3C Managers, while 2-party subscription is used for 300D Managers. The task of maintaining subscriptions for resource-lean Managers is delegated to the Central, so that the Manager needs only to notify the Central if its service changes. The Central notifies the subscribers when the Manager sends an update. In both subscriptions, every update message sent by the Central and Manager is acknowledged. This is still a smaller overhead compared to that incurred by reliable transmission used by Jini and UPnP.

The recovery techniques used during purge-rediscovery depends mainly on the type of architecture, whether peer-to-peer or Registry-based. The competence of the protocols in performing consistency maintenance rely upon how they actually implement the recovery techniques. For example, both UPnP and FRODO with 3-party subscription implement PR5, but Users use different approaches in how they rediscover the Manager. UPnP uses multicast User queries and Manager announcements, while FRODO uses unicast queries to the Registry, before trying multicast queries. We show the differences in implementation in Table 2.

FRODO is unique because it supports both 2-party and 3-party subscriptions. FRODO is also the only protocol to support SRN2, where the Manager retries an unsuccessful update when it receives a subscription renewal message from the User. In our work on Service Discovery Principles, we show that FRODO satisfies the Configuration Update Principles for the critical update scenario with a combination of SRNC1 and SRC2 failure recovery techniques, where periodic updates are monitored by the User, and when the expected update does not arrive, the User requests for the update from the Registry or the Manager.

3.5. Update Metrics

We can benchmark the consistency maintenance performance of state of the art service discovery systems by using the Update Metrics, developed by Dabrowski and Mills. The Update Metrics measure the consistency maintenance performance of service discovery systems against a particular failure rate, λ ($0 \leq \lambda \leq 1$). An example of failure rate is the proportion of time that a node is unable to communicate during the lifetime of the system.

1. Update Responsiveness, $R(\lambda)$. Measures the ratio of the time left after the update is propagated to a User, before a deadline, D to the total time available for the Manager to propagate the update before D.

Let X be the number of runs repeated in the experiment, N the number of Users in the system, $C(i)(< D)$ the time when the service changes, and $U(i,j)$ the time a User receives the update and reaches consistency, where $j = 1$ to N, and $i = 1$ to X. The *relative change-propagation latency*, $L(i,j,\lambda)$ is:
$L(i,j,\lambda) = [(U(i,j,\lambda) - C(i))/(D - C(i))]$
Update Responsiveness, $R(\lambda)$ is the median of $1 - L(i,j,\lambda)$, taken over $j \in 1..N$ and $i \in 1..X$. The median calculation eliminates biasing from extreme scenarios where only messages from the Manager or the Registry are lost (outliers), unlike mean calculation.

2. Update Effectiveness, $F(\lambda)$. Measures the probability of success for a User to reach consistency.

Define $F(\lambda) = \dfrac{\sum_{i=1}^{X}\sum_{j=1}^{N} chg(i,j,\lambda)}{X.N}$

where $chg(i,j,\lambda) = 1$ if $U(i,j,\lambda) < D$ and $chg(i,j,\lambda) = 0$ otherwise.

3. Efficiency Degradation, $G(\lambda)$. This metric is our slight modification of the Update Efficiency metric by Dabrowski and Mills. This metric takes the ratio of a protocol's own minimum number of messages to propagate an update, m (at 0% failure rate), against the total number of messages y, propagated to get *all* the Users in the system to regain consistency.

$G(\lambda) = \dfrac{\sum_{i=1}^{X}(m'/y(i,\lambda))}{X}$

In the less accurate Update Efficiency metric, the value m was fixed to the number of messages propagated by the most efficient protocol at 0% failure rate. This gives an unfair advantage to the baseline protocols as failure rate increases. It is possible that a baseline protocol that propagates the least messages at 0% failure rate degrades faster than the other protocols at higher failure rates. Therefore the Efficiency Degradation metric permits a more accurate evaluation of protocol efficiency because it reflects the effort in the protocols as failure rate increases, to get the Users to regain consistency.

4. Modeling Methodology

We use *Rapide* [4], an Architectural Description Language and tool suite to build an executable model of FRODO. Rapide is designed to support component-based development of systems by utilizing architecture definitions as the development framework. It offers event-based simulation for distributed, time-sensitive systems.

We simulate a total of five models: (1) UPnP, (2) Jini with 1 Registry, (3) Jini with 2 Registries, (4) FRODO with 3-party subscription using 1 300D node as the Registry and (5) FRODO with 2-party subscription, using 8 300D nodes (but still a single Registry system). Our model on FRODO with 2-party subscription contains only 300D nodes, because the nodes have resources similar to the nodes in Jini and UPnP. We reproduce the published results for UPnP and Jini from Dabrowski and Mills to benchmark against FRODO. The following steps describe our approach.

Step 1: Modeling FRODO. The main challenge in modeling FRODO is in developing a framework of behaviors for User and Manager, according to the type of device class. In UPnP and Jini, nodes are homogenous, allowing more straightforward models. In this

Table 2. Recovery techniques, as implemented in the UPnP, Jini and FRODO models.

.

Consistency maintenance recovery techniques	**UPnP**	**Jini**	**FRODO**
Topology	1 Manager, 5 Users	2 topologies. (a) 1 Registry, 1 Manager, 5 Users, (b) 2 Registries, 1 Manager, 5 Users	2 topologies. (a) 1 300D Registry, 1 3D Manager, 5 3D Users (b) 1 300D Registry, 1 300D Manager, 5 300D Users, 1 300D Backup
SRN1: Retransmissions and acknowledgements	TCP enables SRN1	TCP enables SRN1	Retransmissions and acknowledgements of selected messages
SRN2: Retry on unsuccessful notification	-	-	Manager in 2-party subscription retries update notification when it receives subscription renewals from inconsistent Users
PR1: Manager re-registers, and Registry notifies User	-	Users are notified when the Manager registers in the future.	Users are notified if the Manager is available or registers in the future
PR2: User queries the rediscovered Registry for service	-	Users query for the service when the Registry is rediscovered	-
PR3: Registry rediscovers the User, and requests resubscription	-	Registry responds to an unknown User with an error message that requires the User to rediscover the Registry	Registry requests the User to resubscribe
PR4: Manager rediscovers User, and requests resubscription	Manager requests purged Users to resubscribe	-	300D Manager in 2-party subscription requests purged Users to resubscribe
PR5: Users purges and rediscovers Manager	Users rediscover the Manager through muticast queries, or by listening for multicast announcements from the Manager	-	3-party subscription: Users purge the subscription when the Registry purges the Manager. Managers are rediscovered by querying the Registry or by sending multicast queries when the Registry is not responding

experiment, we do not include 3C Managers because they behave exactly the same as 3D Managers during consistency maintenance.

Step 2: Interface failure We use interface failure to model communication and node failures. During the experiment, failures on the receiver or the transmitter simulate communication failure, where a node may send messages, but is not be able to receive messages, or vice-versa. Simultaneous receiver *and* transmitter failure on a node simulates node failure. For each node, the transmitter and/or receiver are failed randomly, at a failure rate λ, varying from 0.00 to 0.90, in increments of 0.05. Interface failure occurs at a random time, from 100s to 5400s. Once the interface failure is activated, it remains in effect for a portion of the simulation duration ($\lambda \times$ 5400s), where 5400s is the entire simulation duration (the reason for 5400s is given in Step 5).

Step 3: Constructing the failure response of transmission protocols. All three protocols use unreliable multicast transmission (UDP). For unicast transmission, FRODO also uses inexpensive UDP, while Jini and UPnP use reliable unicast transmission (TCP). In UDP, when a message is discarded, the source does not learn of the loss. In TCP, a *Remote Exception (REX)* is sent to the service discovery layer of UPnP and Jini when an acknowledgement is not received, after four retries.

Step 4: Constructing the service discovery behavior and recovery techniques. In UPnP, the Manager sends 6 multicast announcement messages every 1800s. In Jini, the Registry sends 6 multicast announcements messages every 120s, while in FRODO, the Registry sends 2 multicast announcements every 1200s. In Jini and FRODO, when the Registry is purged, the Manager rediscovers the Registry by listening for the Registry announcements. FRODO also requires 3D Managers to announce their presence periodically until the Registry is discovered. 300D Managers multicast announcements to start the leader election

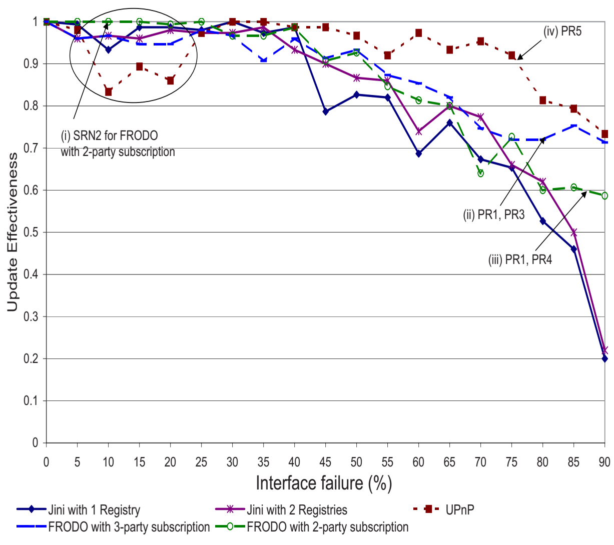

Figure 1. (i) SRN2 is most effective because the Manager resends the update notification when the lease is renewed. (ii) Efficient PR1 in FRODO allows the Registry to update the Users when the Manager or the Registry recovers from failures. PR3 and PR4 in (ii) and (iii) allows Users to re-subscribe to the Registry and the Manager respectively. (iv) PR5 is most effective at high failure rates where Users rediscover the Manager through the Manager's periodic announcements.

process to discover the Registry. We deliberately model FRODO parameters to reflect resource-awareness by not requiring all messages to be retransmitted and acknowledged (only a selected few). We set the period of the Registry announcements so that it is short enough for the discovery process, but long enough so that severe interface failures at high failure rates do not imbalance the system by continuously restarting the leader election process.

The registration lease period for a discovered service to remain valid in the cache of the Registry or User is set to 1800s for all three protocols. In UPnP and FRODO with 2-party subscription, the User subscribes to a discovered Manager. The subscription lease is 1800s for both systems.

Table 2 compares the recovery techniques in the models, and show the differences in implementation.

Step 5: Experiment design We use the application scenarios and parameters used by the UPnP and Jini models [3] for fair comparison. A simulation run lasts for 5400s. The run time is based on the UPnP recommended service lease period of 1800s. All three systems use this period for maintaining a lease for registration and subscription. Thus, using three announcements provides a reasonable opportunity for a system to regain consistency. Five Users discover the Manager and obtain the service description. This process occurs within the first 100s without interface failure. At a random time between 100s to 2700s, the Manager's service changes, causing the Users to become inconsistent with the Manager. Users are notified of this change through 3-party or 2-party subscription.

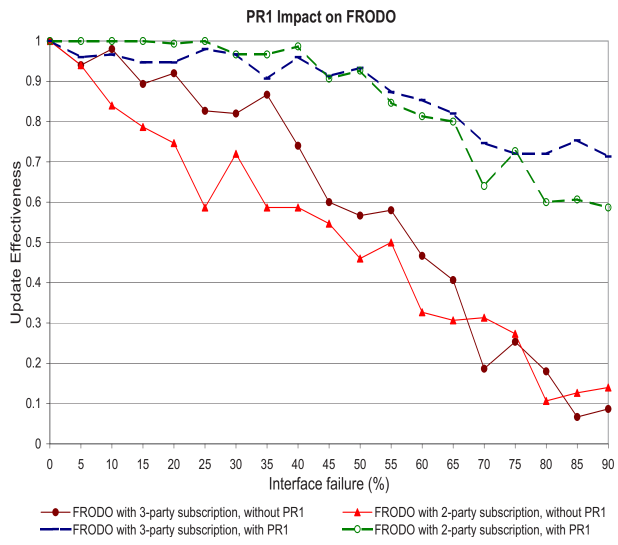

Figure 2. Impact of PR1 recovery technique on the Update Effectiveness of FRODO with 2-party and 3-party subscriptions

5. Results and Discussion

The results we present in this section is a product of a detailed analysis on a random selection of 5 to 10 event logs (out of 30 logs) for each simulated system, at every failure rate. We use 30 runs for each failure rate, to obtain median and average values for the Update Metrics.

Our analysis show that Update Effectiveness is the predominant metric that reflects the impact of recovery techniques. We find that at lower failure rates (below 30%), the prominent recovery technique is SRN2, as implemented in FRODO with 2-party subscription (Figure 1(i)). The impact of SRN2 is apparent especially at low failure rates because Users recover from failures quickly, before subscription is purged. An example of a scenario with a lack of SRN2 is given below.

```
Failure Rate: 15%
Manager Tx down at 381, up at 1191
User Tx and Rx down 2023, up at 2833
UPnP 15% User 2507 5400
```

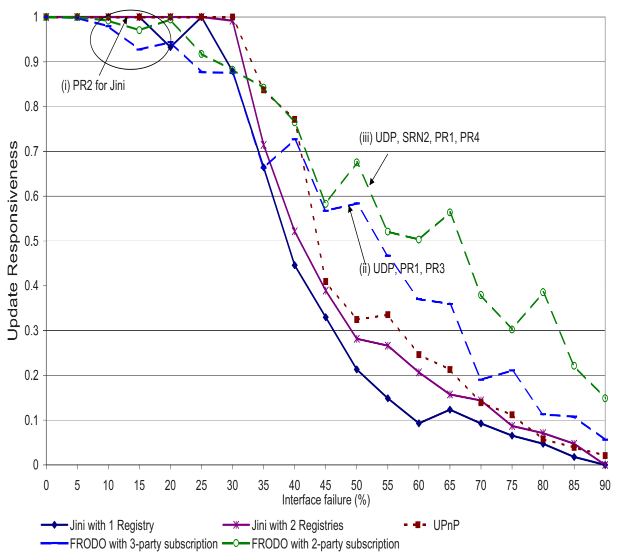

Figure 3. (i) PR2 allows Users in Jini to regain consistency by querying the Registry. FRODO uses SRN2, which depends on the subscription lease period to regain consistency. (ii) UDP transmits messages faster than TCP. PR1 enables the Registry to update the Users when the Registry or the Manager recovers from failures. PR3 enables purged Users to resubscribe with the Registry. (iii) 2-party subscription, UDP, SRN2, PR1 and PR4 allow Users in FRODO to be the most responsive.

The example shows the simulation result of UPnP at 15% failure rate. Tx and Rx mean transmitter and receiver, and the numbers represent time, in seconds. The service changes at 2507s, but the Manager fails to update the User which has both interfaces down from 2023s until 2833s. The update notification fails, and the User never regains consistency! This is a failure to satisfy the Configuration Update Principles. FRODO with 2-party subscription implements SRN2, where the Manager retries the update at a later point of time, when the User renews its subscription with the Manager. Therefore, the effectiveness of FRODO with 2-party subscription is the highest at low failure rates (Figure 1(i)).

At higher failure rates, PR5 as implemented in UPnP (Figure 1(iv)) is the most effective. In FRODO with 2-party subscription, Users rediscover the Manager via the Registry, as opposed to direct, peer-to-peer communication in UPnP, causing the PR5 implementation in UPnP to be more effective than in FRODO. PR1 as implemented in FRODO (Figure 1(ii)) yields the next highest effectiveness. A control experiment with and without PR1, shown in Figure 2 demonstrates the impact of PR1 on the Update Effectiveness of both

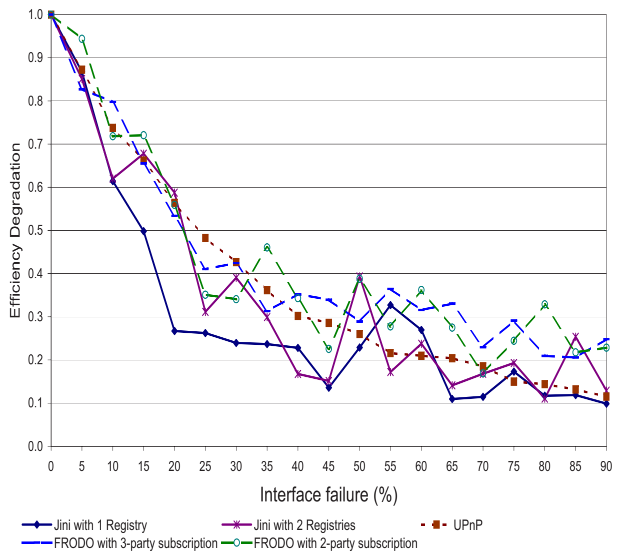

Figure 4. FRODO uses lesser messages during consistency messages than Jini and UPnP. In average, UPnP propagates lesser messages than Jini, even though UPnP sends more messages when there are no failures.

FRODO systems.

Update Responsiveness metric as shown in Figure 3 reveals that FRODO with 2-party subscription incurs the overall shortest delay for Users to regain consistency, due to a combination of direct, peer-to-peer communication between the User and the Manager, fast UDP transmission, low number of messages during consistency maintenance and the use of SRN2 and PR1 recovery techniques.

Efficiency Degradation metric is shown in Figure 4. FRODO gives the best performance for the Efficient Degradation metric. At 0% failure rate, Jini with two Registries and UPnP are the least efficient because they propagate 14 and 15 messages each, while the rest of the systems propagate an average of 7 messages to regain consistency. As mentioned earlier in Section 3.2, update propagation through invalidation, used by UPnP is not efficient for services that do not change frequently, as is the scenario in the experiment. In Jini with 2 Registries, update notification has twice the number of messages than in Jini with a single Registry because the Manager notifies both Registries, and the Users also receive notifications from both Registries. Although Jini with a single Registry is as efficient as FRODO and more efficient than UPnP, it degrades faster than the other two protocols when failure rate increases. The Efficiency Degradation metric of the UPnP and Jini models do not take into account the messages used by the transmission layers. Therefore,

Table 3. Average metrics results across failure rates from 0% to 90%

.

Update Metrics	UPnP	Jini with 1 Registry	Jini with 2 Registries	FRODO with 3-party subscription	FRODO with 2-party subscription
Update Responsiveness, R	0.553	0.474	0.476	0.580	0.666
Update Effectiveness	0.922	0.802	0.825	0.878	0.861
Efficiency Degradation, G	0.385	0.311	0.361	0.428	0.429

the true performance for Jini and UPnP is even lower than shown in Figure 4.

Table 3 shows the average metric results for all failure rates. Although FRODO is a single Registry architecture, and uses unreliable transmission, it has the highest overall performance where it is the most responsive and efficient protocol, while maintaining a high degree of effectiveness. UPnP has the best average effectiveness because of its better performance at high failure rates.

6. Conclusion

Consistency maintenance in service discovery ensures that Users eventually obtain the correct view of the discovered services. We present a novel classification of recovery techniques for consistency maintenance, and propose new techniques to improve performance. We use simulations to show that the type of recovery technique a protocol uses, and how it implements them significantly impacts the proficiency of consistence maintenance. We benchmark the performance of our own service discovery protocol, FRODO against the performance of first generation service discovery protocols, Jini and UPnP during increasing communication and node failures. The results show that FRODO has the best overall consistency maintenance performance.

7. Acknowledgement

This research is part of the At Home Anywhere project sponsored by the Netherlands Organization for Scientific Research (NWO) under grant number 612.060.111, and by the IBM Equinox program. We thank Christopher Dabrowski and Kevin Mills from the US National Institute of Standards and Technology for their support and contribution to this paper.

References

[1] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings of 6th EUNICE Open European Summer School: Innovative Internet Applications*, pages 101–108. University of Twente, September 2000.

[2] V. Cate. Alex - a global file system. In *Proceedings of the USENIX File System Workshop*, pages 1–11, Ann Arbor, Michigan, 1992.

[3] C.Dabrowski, K.Mills, and J.Elder. Understanding consistency maintenance in service discovery architectures during communication failure. In *Proceedings of the Third International Workshop on Software and Performance*, pages 168–178. ACM Press, July 2002.

[4] D. Luckham. Rapide: A language and toolset for simulation of distributed systems by partial ordering of events. In *Proceedings of Worldwide Computing and Its Applications, International Conference, WWCA '98, Second International Conference*, volume 1368 of *Lecture Notes in Computer Science*, pages 88 – 96. Springer-Verlag, March 1998.

[5] Microsoft. *Universal Plug and Play Architecture, V1.0*, Jun 2000.

[6] Sun Microsystems. *The Jini Architecture Specification, version 2.0*, June 2003.

[7] K. Petersen, M. Spreitzer, D. Terry, and M. Theimer. Bayou: replicated database services for world-wide applications. In *EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pages 275–280, 1996.

[8] V. Sundramoorthy, J. Scholten, P. G. Jansen, and P. H. Hartel. Service discovery at home. In *4th Int. Conf. on Information, Communications & Signal Processing and 4th IEEE Pacific-Rim Conf. On Multimedia (ICICS/PCM)*, page 1929. IEEE Computer Society Press, December 2003.

[9] V. Sundramoorthy, C. Tan, P. H. Hartel, J. I. den Hartog, and J. Scholten. Functional principles of registry-based service discovery. In *30th Annual IEEE Conf. on Local Computer Networks (LCN)*, page to appear, Sydney, Australia, Nov 2005. IEEE Computer Society Press.

[10] V. Sundramoorthy, G. J. van de Glind, P. H. Hartel, and J. Scholten. The performance of a second generation service discovery protocol in response to message loss. In *1st Int. Conf. on Communication System Software and Middleware*, page to appear, New Delhi, India, Jan 2006. IEEE Computer Society Press.