The Performance of a Second Generation Service Discovery Protocol In Response to Message Loss

V.Sundramoorthy, G.J.van de Glind, P.H.Hartel, and J.Scholten

University of Twente
Enschede, The Netherlands
vasughi.sundramoorthy@cs.utwente.nl

***Abstract*— We analyze the behavior of FRODO, a second generation service discovery protocol, in response to message loss in the network. First generation protocols, like UPnP and Jini rely on underlying network layers to enhance their failure recovery. A comparison with UPnP and Jini shows that FRODO performs more efficiently in maintaining consistency, with shorter latency, not relying on lower network layers for robustness and therefore functions correctly on a simple lightweight protocol stack.**

Keywords: service discovery, self-healing networks

I. Introduction

Service discovery protocols allow devices to discover their environment, detect and adapt to topology changes, establish communication with each other and share services. One way to classify service discovery protocols is according to the size of the network: local area network (Jini [1], UPnP [2], SLP [3], [4], FRODO [5]) and wide area network (SSDS [6], INS/Twine [7]). FRODO was built for the home environment, thus we focus on service discovery for small, local area networks (LAN) in this paper.

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) and *peer-to-peer* (e.g. UPnP). 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. In the peer-to-peer architecture there are 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, FRODO).

Contribution: This paper shows that by implementing a robust service discovery architecture, FRODO proves more responsive, effective, efficient and has the shortest latency in maintaining consistency compared to UPnP and Jini, up to 45% message loss rate. We also show that reliable unicast transmission has little positive impact on consistency maintenance performance at lower than 60% loss rate.

Section II of this paper presents related work. Section III provides a brief overview of FRODO. Section IV describes the fundamentals of consistency maintenance in service discovery systems. Section V presents the modelling methodology and the experiment description. Section VI benchmarks the performance of FRODO against UPnP and Jini. Section VII concludes.

II. Related Work

This work is inspired by the work done by Dabrowski and Mills from NIST in evaluating the consistency maintenance mechanisms of UPnP and Jini. To the best of our knowledge, only Dabrowski and Mills measure the consistency maintenance performance in service discovery. They use an architectural-based approach using an ADL to analyze service discovery systems [8]. They develop a set of metrices to benchmark the performance of service discovery systems in a dynamically changing environment, with increasing message loss [9] and interface failure [10] as the communication failure models. In these work, they also mention that consistency maintenance through notification for UPnP and Jini rely on reliable unicast transmission to enhance recovery from communication failure. They propose a generic model encompassing the design of first-generation service discovery systems [11]. In this work, they report first-generation service discovery systems do not provide guarantees of behavior. FRODO is a second-generation service discovery protocol that provides guarantees [5], implements zero-configuration and increase robustness at the service discovery layer. We use the same scenarios and metrics as used by Dabrawski and Mills for UPnP and Jini to analyze FRODO and to make a comparison of the three protocols.

III. 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 device class - simple devices with restricted resources (e.g. smart dust). Nodes in this class are only Managers. (2) 3D 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 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 name the Registry as *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 as the Central in case of Central failure.

FRODO classifies the tasks of service discovery into four major functions, which are implemented according to the device classes:

1. Configuration Discovery: FRODO provides zero-configuration as follows: 300D nodes elect the Central which becomes the Registry. 3D nodes discover the Central by sending multicast announcements (*active* discovery), while 3C nodes discover the Central through the Central's periodic announcements (*lazy* discovery), since they do not have the capability to do periodic active discovery.

2. Service Registration: By definition, a Manager has at least one service to offer. The service is registered with the Central. There are 2 types of registrations, *solicited* and *unsolicited* registrations. Solicited registration allows the Central to initiate the registration process when it discovers an unrecognized Manager, while unsolicited registration allows nodes to register with an unrecognized Central.

3. Service Discovery: When a User needs a service, it sends its service requirements to the Central. There are 2 types of mechanisms to search for services: with Central, through the *directed search* scheme, and without Central through the *peer search* scheme. Directed search is done when a User knows the existence of the Central, while peer search is used when the User is not able to discover a Central. The Central can also notify the User of future matching services, if there are no such services currently registered.

4. Configuration Management: FRODO avoids single point of failure by appointing a Backup for the Central. The Central and Backup poll each other, so that if the Central fails, the Backup takes over, or if the Backup fails, another node is appointed. The registration and subscription data are stored in the Backup as well as in the Central for seamless recovery. The leader election process is restarted by 300D nodes when the Central does not respond to their messages. To detect and purge defunct services, FRODO implements garbage collection by requiring 300D Managers to renew their registration periodically. 3D/3C Managers are polled by the Central periodically. If the lease is not renewed, the node is purged. Users can also notify the Central of non-responding Managers, so that they can be probed by the Central and purged if necessary. To heal network partitions where multiple Centrals may exist, a negotiation process among the Centrals maintains only one Central in the system. Configuration Management also includes consistency maintenance mechanisms, so that a User can subscribe to a Manager to receive updates on a discovered service.

Unlike first generation service discovery systems, FRODO does not depend on the recovery abilities of lower layer protocols 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.

TABLE I

CONSISTENCY MAINTENANCE IN STATE OF THE ART SYSTEMS. FRODO IMPLEMENTS BOTH 2-PARTY AND 3-PARTY SUBSCRIPTIONS, ACCORDING TO THE DEVICE CLASS OF THE MANAGER

Consistency maintenance attributes	**UPnP**	**Jini**	**FRODO**
Update type	Service description update, event notification	Service description update	Service description update, event notification, supports essential update
Consistency maintenance mechanism	Polling, Notification	Polling, Notification	Polling, Notification
Subscription type	2-party	3-party	3-party(3C/3D Manager), 2-party (300D Manager)

IV. FUNDAMENTALS OF CONSISTENCY MAINTENANCE

"Consistency" is the state where the User obtains the correct service information after the service changes. Users become consistent with the Manager when they successfully receive update information from the Manager. The mechanism used to achieve consistency is known as "consistency maintenance".

A. Types of Update Information

After a service is discovered, it can undergo 2 types of change: (1) the change in the capability of the service, which is described in the service description and (2) the change in the service after an event occurs. To illustrate the difference between the two types of change, we use an example of a Manager which offers a printing service. The service description format is according to the attribute-value pair structure. The event variable "PaperTray" reflects the status of the printer paper tray when it empties:

```
SD = {DeviceType=Printer,
      ServiceType=ColorPrinter,
      AttributeList{PaperSize=A4,
             EventVar{PaperTray}}}
```

When the value of "DeviceType", "ServiceType" or "PaperSize" changes, the service description changes. For example, the "ServiceType" changes from "ColorPrinter" to "Black&WhitePrinter". The printer propagates this change to the Registry and the interested User as a *service description update*. The previous service description registered in the Registry and obtained by the User is overwritten. Service description change does not occur frequently. However, the printer runs out of paper quite frequently, triggering the change in the "PaperTray" variable. The printer propagates the change to the User as an *event variable update*.

In FRODO, if an update is considered *essential*, it is propagated until acknowledged (as long as the subscription is active). A sequence number for every event variable update message also allows the User to monitor for missed events. The User can request a missed event variable update if necessary.

Among first-generation service discovery protocols, only UPnP [11] supports event variable update. To the best of our knowledge, FRODO is the first registry-based protocol that defines and supports both service description update and the event variable update. However, in this paper, we only focus on service description update because Jini does not include event variable update.

B. Consistency maintenance mechanisms

To maintain consistency, the User has to first *subscribe* either directly to the Manager (2-party subscription) or to a Registry (3-party subscription) to receive updates. There are 2 methods for Users to achieve consistency with the Managers.

(i) *Notification* - In this method, a User receives an update when the service description changes. In a registry-based architecture, the Manager notifies the Registry which then propagates the update to subscribed Users. In a peer-to-peer architecture, the Manager notifies subscribed Users directly.

(ii) *Polling* - In this method, the User is responsible to discover the update. In a registry-based architecture, the Manager updates the Registry by re-registring its services. In both service discovery architectures, periodic queries from the Users will eventually retrieve the updated service description.

Consistency maintenance in registry-based architectures relies on the successful communication between the Manager and the Registry, *and* between the Registry and the User (*3-party subscription*). Peer-to-peer architectures rely only on the successful communication between the User and the Manager (*2-party subscription*).

Table I summarizes consistency maintenance in UPnP, Jini and FRODO. Although polling is available in FRODO, we do not model it in this work for 2 reasons: (1) Polling is driven by the application layer (periodic query for a service), and thus does not reflect the recovery ability of the service discovery layer. (2) Polling is not resource-efficient because Users are required to persistently query the service, even when the service rarely changes. This does not fit in the resource-aware context of FRODO.

C. Consistency maintenance through notification in UPnP, Jini and FRODO during message loss

The initial steps in 3-party and 2-party subscriptions during consistency maintenance are similar for all systems. After discovering a service, the User subscribes to the Registry (3-party subscription) or directly to the Manager (2-party subscription), and renews the subscription lease periodically. When the Manager has an update, it notifies the Registry (in 3-party subscription, the Registry notifies the Users) or the User directly.

Jini uses 3-party subscription while UPnP uses 2-party subscription. In Jini, the Manager sends an update to the Registry, and receives an acknowledgement. The Registry propagates the update to the subscribed Users. Thus, in a sample topology (Section V) of 1 Registry, 1 Manager and 5 Users, the number of update messages is minimum 7 (without retransmissions). In UPnP, the Manager sends notifications to the subscribed Users. The notification only indicates that the service has changed. A User receives the actual update only after it requests the change. Thus in a topology of 1 Manager and 5 Users, a minimum of 15 messages (without retransmission) are propagated for consistency maintenance.

In both systems, a message is only sent if the reliable transmission using TCP successfully sets up a connection between the sender and the receiver. Messages for setting up the connection and sending the update are acknowledged and retransmitted, as part of the TCP behavior.

In FRODO, the subscription for the 300D Manager uses the 2-party scheme, while the subscription for the 3D/3C Manager uses the 3-party scheme. 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. Thus in the 3-party scheme, with 1 300D node, 1 Manager and 5 Users, a minimum of 6 update messages is needed for the system to reach consistency. In the 2-party subscription topology of 8 300D nodes with 1 Central, 1 Backup, 1 Manager and 5 Users, a minimum of 7 messages is needed for consistency. This includes an upate to the Central, and an update from the Central to the Backup. In both subscription types, every update message requires an acknowledgement, but this is a smaller overhead compared to that incurred by reliable unicast transmission used by Jini and UPnP.

The subscription for all three systems may still remain active when the TCP retransmissions during consistency maintenance for Jini and UPnP fail, or the Manager in FRODO gives up transmitting the update. This can cause the User never to regain consistency. However, future communication failures may cause the User to purge the service description of the Manager and restart the discovery process. Through this, the User may rediscover the Manager, along with the updated service description. In FRODO, when the Central and Manager receive a subscription renewal message for an inactive subscription, they request the User to resubscribe. This is another opportunity for the User to obtain the update and regain consistency.

Consistency maintenance can also be facilitated by the application layer. If the communication between the User and Manager fails during an application response, the User can purge and rediscover the Manager, instead of waiting for the subscription or registration lease to expire. Thus the chances for retrieving an updated service description is increased. However, we do not explore this point further in this paper, because we only focus on the service discovery response.

V. Modelling Methodology

We use *Rapide* [12], 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 an event-based execution for distributed, time-sensitive systems.

The following steps describe the modelling methodology.

Step 1: Modifying the Jini model from NIST: We study the Jini model from NIST to understand the methods for modeling a registry-based service discovery in Rapide. We then drop Jini-specific functions and reuse with some modification, the functions that support and specify the simulation environment. We also make the simulation environment more flexible by removing timing and retransmission policies from the service discovery model, and implementing them as inputs for a more scenario-driven approach.

Step 2: Constructing components of FRODO: The class diagrams in Figure 1 and 2 show the difference in the structure of our FRODO model against the Jini model by Dabrowski and Mills [8]. The main challenge in modelling 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 experiment, we do not include 3C Managers because they behave exactly the same as 3D Managers.

TABLE II

NETWORK CHARACTERISTICS. UPNP AND JINI RELY ON NOTIFICATIONS FROM THE TRANSPORT LAYER TO DETECT TRANSMISSION FAILURES. FRODO DOES NOT RELY ON LOWER LAYERS TO DETECT FAILURES. REDUNDANT MULTICAST TRANSMISSIONS ALSO DO NOT OCCUR IN FRODO BECAUSE IT DOES NOT FIT THE RESOURCE-AWARE CONTEXT.

Network behavior and failure response	**UPnP and Jini**	**FRODO**
Multicast	UDP	UDP
Unicast	TCP	UDP
Transmission delay	0.14s-0.42s	0.14s-0.42s
Unreliable protocol (UDP) response to message loss	Message discarded. No retransmission. Redundant 6 times transmission for all messages	Message discarded. No retransmission.
Reliable protocol (TCP) to message loss	Connection setup: 4 retransmission attempts with delays 6s, 24s, 24s, 24s, then REX if unsuccessful. Data transfer: retransmit until success, increasing, timeout by 25% on each retry (first time-out is round trip time)	-

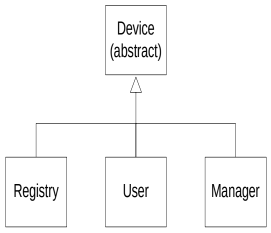

Fig. 1. In the Jini model from NIST, a device can instantiate as a User, a Manager, or a Registry. The Registry component is only relevant for Jini).

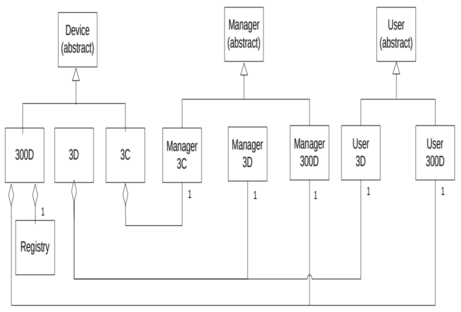

Fig. 2. In the FRODO model, a device can instantiate as a 3C, 3D or 300D class. A 300D node has a Registry component which performs leader election, and is triggered when it is elected as the Central. 300D and 3D nodes can instantiate as a User and a Manager, while 3C nodes are only Managers. The User and Manager behaviors are tailored according to the device class limitations.

Step 3: Constructing the failure response of transmission protocols: All three models 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 retrying and waiting, as explained further in Table II.

Step 4: Constructing the service discovery behavior and failure response: The period for a service to remain valid in the cache of the Registry or User is called *Time to Live (TTL)*. When the service TTL expires, the User or the Registry purges the service description. The User then restarts the discovery process. In Jini and FRODO, the User may also purge the Registry if there were no response to its query. In this case, nodes rediscover the Registry by monitoring the Registry's periodic multicast announcements (both Jini and FRODO implement lazy discovery). In FRODO, nodes also do active discovery where 300D nodes restart leader election, while 3D nodes periodically send multicast announcements. Peer search also allows the Users in FRODO to discover the Manager when the Central is not responding, allowing another opportunity to regain consistency. The failure to receive acknowledgement for subscription requests in FRODO allows the User to restart the discovery process. Furthermore, if the Central (3-party subscription) or the Manager (2-party subscription) receive unidentified subscription renewals (subscription purged due to communication failure), they request the subscriber to resubscribe, thus allowing the User to regain consistency.

Table III compares the recovery mechanisms in the three systems, along with related experiment parameters.

Step 5: Experiment design: We use the same application scenarios and parameters as used by the UPnP and Jini models at NIST [9] for fair comparison. A simulation run lasts for 5400s. The run time is based on the UPnP recommended TTL of 1800s. All three systems use this period for service TTL. Thus, 3 announcements provides a reasonable opportunity for a system to regain consistency. 5 Users discover the Manager and obtain the service description. This process occurs within the first 100s without message loss. At a random time between 100s to 2700s, the

TABLE III

EXPERIMENT PARAMETERS, AND COMPARISON OF SERVICE DISCOVERY FAILURE RESPONSE

Experiment parameters	**UPnP**	**Jini**	**FRODO**
Topology	1 Manager, 5 Users	1 Registry, 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
Service announcement by Manager	The Manager multicasts 6 announcement messages every 1800s.	-	-
Registry announcements	-	The Registry multicasts 6 announcements every 120s	The Central multicasts announcements every 1200s.
User/Manager discovery of Registry	-	within a 5s interval, 7 announcements are sent at startup. When a node purges the Registry, it listens for the periodic announcements to rediscover the Registry	300D: Leader election to configure the Registry. 3D: exponential back-off announcements starting from 10s to 800s, remains at 800s until Central discovered. For rediscovering the Central, the same steps are carried out
TTL	1800s	1800s	1800s
Service rediscovery by User after TTL expires	User multicasts queries every 120s	The User unicasts a query to the Registry once. If the service is unavailable, the User asks to be notified when the service becomes available	The User unicasts a query to the Central. If the service is unavailable, the User asks to be notified when the service becomes available. If there is no response (either positive, or negative) to its query, the User retries every 180s up to 6 times
2 and 3-party subscription: Subscription TTL	1800s	1800s	1800s (2 x 900s)
Acknowledgements	Not as part of service discovery, but using TCP	Not as part of service discovery, but using TCP	Critical messages, including subscription messages are acknowledged
Retransmission	Reliable unicast transmission protocol, redundant multicast transmission	Reliable unicast transmission protocol, redundant multicast transmission	Critical messages are retransmitted once. Update messages are retransmitted 4 times
Other features	No single point of failure because of peer-to-peer architecture	Adding more Registries to increase robustness	Peer search, leader election and Backup reduce single point of failure problem, notification can be implemented until update is successful

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.

This experiment benchmarks 4 models: UPnP, Jini, FRODO with 2-party subscription and FRODO with 3-party subscription. For FRODO, the 3-party subscription model focuses on the behavior of resource lean nodes, therefore the topology consists of only 1 300D node, while the rest are 3D nodes. The 2-party subscription model consists of only 300D nodes. This model is comparable to Jini with a single Registry topology, while behaving as UPnP when propagating service updates.

A summary of the topologies and experiment parameters is given in Table III.

Step 6: Message loss: Messages are discarded randomly, at a loss rate λ, varying from 0.00 to 0.95, in increments of 0.05.

Step 7: Metrics: The metrics from NIST [9] measure the latency, success probability and efficiency of consistency maintenance of service discovery systems against a particular failure rate, λ.

(i) Update Responsiveness, R_λ. Measures the ratio of the available time 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 (in this experiment, D is the simulation run time of 5400s).
Let X be the number of runs repeated in the experiment, N the number of Users in the system, $t_{C(i)} (< D)$ the time when the service changes, and $t_{U(ij)}$ 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_{ij} is:
$L_{ij} = (t_{U(ij)} - t_{C(i)})/(D - t_{C(i)})$.
Update responsiveness R_λ is the median of $(1 - L_{ij})_\lambda$. Median calculation rules out extreme scenarios where only messages from the Manager or the Registry are lost (outliers).

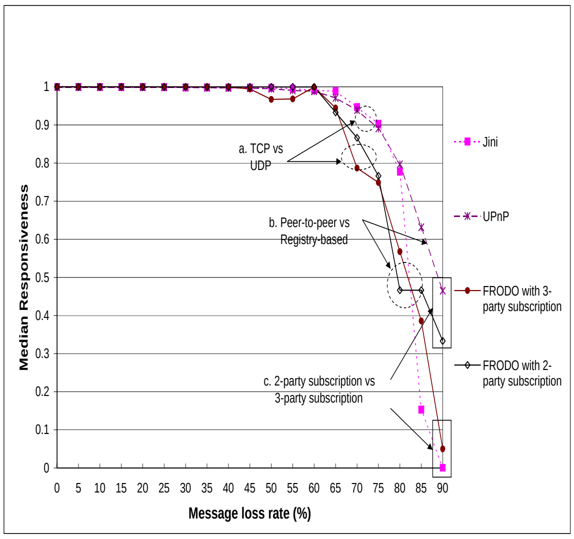

Fig. 3. Update Responsiveness. (a) Reliable unicast transmission, TCP allows Jini and UPnP to be more responsive than FRODO after 60% loss rate. (b) UPnP's peer-to-peer architecture is the most responsive after 80% loss rate. (c) 2-party subscription is more responsive then 3-party subscription after 80% loss rate.

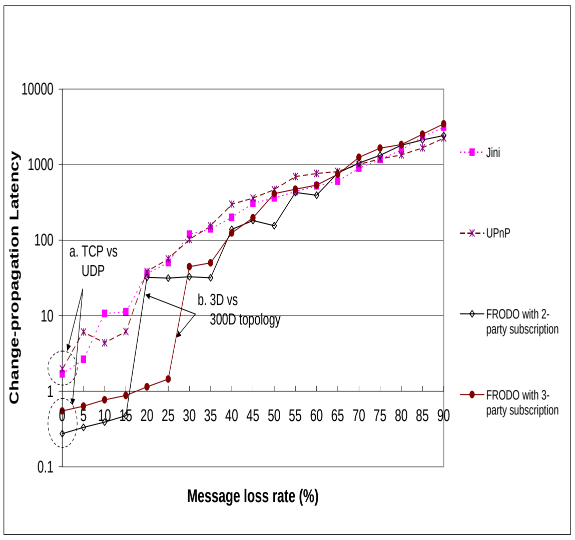

Fig. 4. Change-propagation latency, L'. Measures the delay for a User to regain consistency. (a) Unreliable unicast transmission, UDP allows faster update propagation up until 55% loss rate. TCP and the use of higher number of messages cause UPnP and Jini to have higher latency (b) The topology with 300D nodes incur more latency after 15% loss rate because 300D nodes perform more number of tasks then 3D nodes.

(ii) Update effectiveness, U_λ. Measures the probability of success for a User to reach consistency.

Define $U_\lambda = \dfrac{\sum_{i=1}^{X}\sum_{j=1}^{N} chg_{i,j}}{X.N}$

where $chg_{i,j} = 1$ if $t_{U(i,j)} < D$ and $chg_{i,j} = 0$ if otherwise.

(iii) Update Efficiency, E_λ. Measures the effort required to maintain consistency. Let m be the minimum number of messages across all systems to propagate a change to the Users. In this experiment, $m = 6$ based on FRODO 3-party subscription. Let y be the number of messages sent during the time the Users try to reach consistency in a system. Thus update efficiency is the ratio of m to y.

$$E_\lambda = \frac{\sum_{i=1}^{X}(m/y_i)}{X}$$

VI. Results and Discussion

In this section, we present the results of our experiment showing the relative performance of Jini, UPnP and FRODO. We find that FRODO is the most responsive and effective protocol up to 45% loss rate, and has the highest efficiency and lowest latency for consistency maintenance. We also find that reliable unicast transmissions and redundant multicast transmissions in Jini and UPnP do not provide any positive impact below 60% loss rate. Figures 3, 5 and 6 compare Update Responsiveness, Update Effectiveness and Update Efficiency against increasing message loss. The results can be analyzed according to the following classification:

1. TCP versus UDP transmission: At lower failure rates, the different types of transmission policies do not show any positive influence on Update Responsiveness and Update Effectiveness. Recall that we do not model aggressive transmission policy in FRODO, as was done for Jini and UPnP. If we implement aggressive transmission policy, the performance of FRODO will improve, but will not satisfy the resource-aware context of FRODO.

Nevertheless, after 60% loss rate, the more aggressive transmissions allow Jini and UPnP to be more responsiveness and effective than both FRODO systems, as shown in Figure 3(a) and Figure 5(a). However, failure of TCP retransmissions in Jini cause Update Responsiveness and Update Effectiveness to decline steeply at 80% and 70% respectively. Unlike Jini, the decline of the responsiveness and effectiveness of UPnP is more gradual because of the inherent robustness of its peer-to-peer architecture, discussed in (2).

When the lower layer protocol stacks fail to recover consistency, all three protocols depend on the recovery mechanisms in the service discovery layer, which includes rediscovering the Manager, as explained in Section IV-C

We further investigate the impact of TCP and UDP by comparing the absolute latency for Users in each protocol to regain consistency, We remove the denominator from the relative change-propagation, L_λ, so that

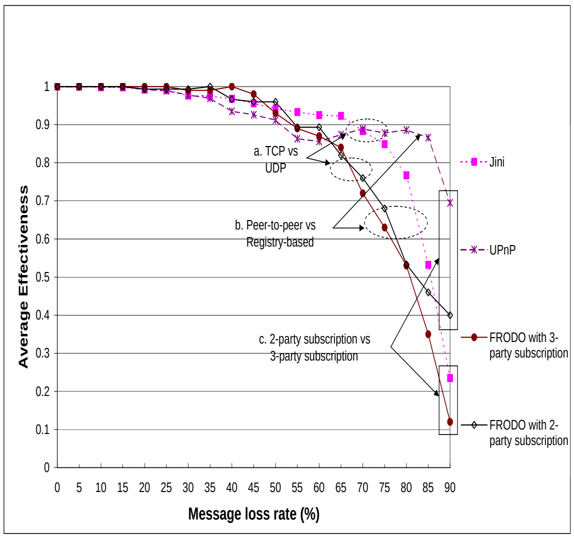

Fig. 5. Update Effectiveness. (a) Reliable unicast transmission, TCP allows Jini and UPnP to be more effective than FRODO after 60% loss rate. (b) UPnP's peer-to-peer architecture is the most effective after 75% loss rate. (c) 2-party subscription is more responsive then 3-party subscription after 70% loss rate.

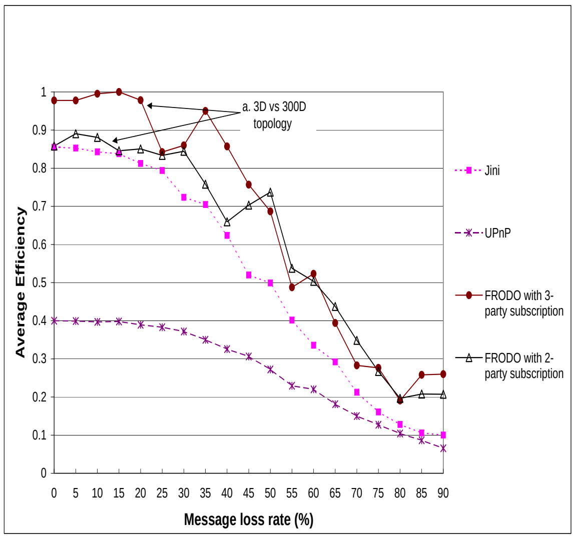

Fig. 6. Update Efficiency. (a) FRODO with 3-party subscription has 1 Central and 6 3D nodes, and use the least number of messages to maintain consistency. Therefore it is the most efficient. 300D nodes in FRODO with 2-party subscription use more messages, because they also perform more number of tasks

the absolute change-propagation latency for a message loss rate, L'_λ is:
$L'_\lambda = t_{U(ij)} - t_{C(i)}$.
The result of this investigation is shown in Figure 4. TCP causes longer change-propagation latency in UPnP and Jini, compared to UDP, as shown in Figure 4(a). TCP uses additional time for setting up the connection before actually sending the update message. UDP does not require connection setup.

2. Peer-to-peer versus registry-based architecture: UPnP is a peer-to-peer architecture while Jini and FRODO are registry-based architectures. The difference in the performance of the two types of architectures are only seen at high failure rates. In Update Responsiveness, UPnP performs better than Jini after 80% loss rate, as shown in Figure 3(b) while in Update Effectiveness, UPnP performs better than Jini after 70% failure rate, as shown in Figure 5(b). This is because the peer-to-peer architecture does not have a single point of failure problem as faced by the registry-based architecture of Jini and FRODO. At high failure rates, consistency maintenance depends on the failure recovery strategy of the service discovery layer because Users will end up purging the Manager. To regain consistency, they have to rediscover the Manager. UPnP does not require a Registry for the rediscovery process, thus increasing its chances of success.

However, in Update Efficiency, UPnP has the worst performance. This is because the notification mechanism in UPnP uses three steps; notification from Manager to the User to indicate an update has occurred, request from the User for the update, and finally the update reply from the Manager. For the system to regain consistency at 0% loss rate, UPnP uses 15 messages, Jini and FRODO with 2-party subscription use 7 messages and FRODO with 3-party subscription uses 6 messages. The UPnP and Jini models from NIST do not take into account the messages used by transmission layers for the Update Efficiency metric. Therefore, the true Update Efficiency for Jini and UPnP is even lower than shown in Figure 6.

The 3-step notification in UPnP also impacts the effectiveness of UPnP at lower failure rates, from 35% to 60%. The chances for failure is higher for UPnP because if TCP fails to transmit the multiple notification messages, the subscription may continue to remain active at such failure rates, providing no opportunity for UPnP to fall back on the recovery process in the service discovery layer.

3. 2-party subscription versus 3-party subscription : Even though FRODO with 2-party subscription is a registry-based architecture, at high failure rates. FRODO with 2-party subscription is more responsiveness (Figure 3(c)) and effectiveness (Figure 5(c)) beyond 80% loss rate, than Jini and FRODO with 3-party subscription counterpart. This is because after discovering the Manager, nodes in the 2-party subscription model do not depend on a Registry to propagate the update. The direct communication between the Manager and the User reduces the chances for failure at very high message loss rates. The resubscription process for regaining consistency in FRODO is more effective in 2-party subscription than in 3-party subscription, which depends on the Central for maintaining subscription.

The direct communication between the User and the Manager also reduces the change-propagation latency in the 2-party subscription, as shown in Figure 4. However, from 15% and 30% failure rates, we see that FRODO with 3-party subscription model has lower latency than its 2-party counterpart. The 2-party subscription model of FRODO consists of only 300D nodes. 300D Users have more tasks than 3D Users, which includes becoming the Backup if necessary (the Central may purge the Backup due to message loss).

The 2-party subscription of FRODO has lower responsiveness and effectiveness than UPnP because of the difference in the recovery process in service discovery. At high failure rates, when the User purges the Manager, FRODO with 2-party subscription still depends on the Central to rediscover the Manager and regain consistency. This is the disadvantage of the registry-based architecture, as explained in (b).

The worst-case scenario for message loss in IEEE 802.11b, ad-hoc mode wireless links with no error control is around 40% for the office environment, measured by Hoene et al [13] for 2 nodes at a critical distance of 18m. In base station mode, with error control, the measurements show that the typical loss rate does not exceed 5%. The loss rates also apply for the home environment. Since there is no agreement in the research community for the message loss rates in a typical home environment, we use 20% (within the measured 5% to 40% range) as the reasonable loss rate to compare the performance of the three protocols in Figure 7. FRODO with 3-party subscription performs the best, followed by FRODO with 2-party subscription. Note that the change-propagation latency for FRODO with 3-party subscription is low compared to the other models. As explained earlier in Section VI, TCP in Jini and UPnP losses some time setting up the connection, and retransmitting the connection requests and the update messages. FRODO only incurs one-time transmission delay to send an update, and very few retransmissions at this failure rate. However, FRODO with 2-party subscription (topology of 300D nodes) have higher latency than the more lightweight 3-party subscription topology for reasons discussed in Section VI.

Therefore, even though FRODO implements zero-configuration, *adds more robust features by removing dependency on the recovery of lower layer protocol stacks*, and limits behavior of resource-lean nodes, its performance satisfies the requirement for wired and wireless (including ad-hoc), home networks. The performance only deteriorates significantly at failure rates above 60% which is expected because of unreliable transmission protocol and low retransmissions. Although the performance can be increased by adding redundant transmissions for both unicast and multicast messages, we chose not to do so to satisfy the resource-aware context of FRODO.

System	Consistency maintenance performance at 20% loss rate			
	Median Responsiveness	Average change-latency (s)	Average Effectiveness	Average Efficiency
FRODO with 2-party subscription	1.000	32.145	0.993	0.851
FRODO with 3-party subscription	1.000	1.141	1.000	1.000
Jini	1.000	36.192	0.992	0.812
UPnP	0.912	38.537	0.992	0.389

Fig. 7. Comparison of consistency maintenance performance for UPnP, Jini and FRODO at 20% message loss rate. The shaded values show the best performance for a particular metric. FRODO with 3-party subscription has the highest, overall performance for this message loss rate

VII. Conclusion

Consistency maintenance in service discovery ensures that Users have the correct view of the discovered services. We show that by incorporating robustness against communication failures such as message loss, FRODO is more effective and efficient than Jini and UPnP, with the lowest update latency for message loss rates under 45%. The performance of FRODO remains high up until 60% loss rate. This proves that FRODO can be deployed over leaner, less powerful protocol stacks with limited failure recovery. We also show that a reliable transmission protocol is only beneficial at high message loss rates (higher than 60%). In conclusion, FRODO satisfies its objectives for resource-awareness and robustness for small, local area networks. On-going work in this area includes validating the simulation measurements against a real-life scenario, measuring the consistency maintenance of FRODO during interface failure and measuring the performance of its discovery process.

VIII. Acknowledgement

This research is 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] Sun Microsystems, *The Jini Architecture Specification, version 2.0*, June 2003.

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

[3] 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*, September 2000.

[4] E. Guttman, C. Perkins, J. C. Veizades, and M. Day, *Service Location Protocol, V.2*, December 2003.

[5] V. Sundramoorthy, M. D. Speelziek, G. J. van de Glind, and J. Scholten, "Service discovery with FRODO," in *12th IEEE Int. Conf. on Network Protocols (ICNP)*. Berlin, Germany: Computer Science Reports, BTU Cottbus, Oct 2004, pp. 24–27.

[6] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service," in *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM '99)*. ACM Press, New York, NY, USA, August 1999.

[7] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," in *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*. ACM Press, New York, NY, USA, December 1999, pp. 186–201.

[8] C. Dabrowski and K. Mills, "Analyzing properties and behavior of service discovery protocols using an architecture-based approach," in *Proceedings of Working Conference on Complex and Dynamic Systems Architecture (CDSA)*. Distributed Systems Technology Centre, December 2001.

[9] C. Dabrowski, K. Mills, and J.Elder, "Understanding consistency maintenance in service discovery architectures in response to message loss," in *Proceedings of the 4th International Workshop on Active Middleware Services*. IEEE Computer Society, July 2002, pp. 51–60.

[10] 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*. ACM Press, July 2002, pp. 168–178.

[11] C. Dabrowski, K. Mills, and S. Quirolgico, *A Model-based Analysis of First-Generation Service Discovery Systems*. Special Publication 500-260, National Institute of Standards and Technology, 2005.

[12] 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*, ser. Lecture Notes in Computer Science, Y. Masunaga, T. Katayama, and M. Tsukamoto, Eds., vol. 1368. Springer-Verlag, March 1998, pp. 88 – 96.

[13] C. Hoene, A. Gunther, and A. Wolisz, "Measuring the impact of slow user motion on packet loss and delay over ieee 802.11b wireless links." in *28th Annual IEEE Conference on Local Computer Networks (LCN 2003), The Conference on Leading Edge and Practical Computer Networking*. IEEE Computer Society, 2003, pp. 652–662.