

## Managing Wireless Ad-hoc Networks, IP Addressing and Service Delivery

Sinead Cummins, J.P. O'Grady & Fergus O'Reilly

Adaptive Wireless Systems Group

Department of Electronic Engineering

Cork Institute of Technology

E: Mail: scummins@cit.ie , jpogradey@cit.ie & foreilly@cit.ie

### Abstract:

*Service delivery over wireless ad-hoc networks requires, firstly the wireless creation of the ad-hoc networks, and then on these networks the ad-hoc discovery and delivery of services over this temporary infrastructure. On the creation of a wireless network, individual nodes must be identified or addressed correctly, cogniscent of the challenges posed by the ad-hoc nature of the communication. Secondly, services provided may then be provided and used only on a temporary and ad-hoc basis as well, requiring that systems for service discovery and delivery must also be suitable for operating in this environment. This paper examines approaches to Wireless Ad-hoc Address Allocation based on the IP Protocol and from this approaches for Ad-hoc Wireless Service Delivery based on the Jini/Java programming / communication model. Together these can be combined to provide for IP based wireless Service Delivery for Ad-hoc networks.*

### 1. Introduction

Smart Space and Managed Zone research frequently suggests the use case of a suitably equipped person, carrying the requisite intelligent PDA/SmartPhone, walking into a smart environment and then describes the intelligent interactions which can occur to make that person's time/experience as productive and automated as possible. Lighting will adjust accordingly, e-mail, calls will forward correctly, chairs, music will adjust to that persons remotely stored preferences. When two or more such enabled people meet, maybe at a hotel, conference or on the street, their intelligent devices will communicate wirelessly, update contact, check each other's schedules and arrange meetings at mutually agreeable times all with minimal input. Information will flow at local, macro and national level to provide seamless access and services, all transparent and with minimal interaction from the user.

One common characteristic of much of the above communication is that it is ad-hoc. Users and systems may appear/communicate and disappear on an ad-hoc basis, both where and when it suits them. Wireless communication networks will thus need to form, communicate, adjust, re-form adjust and disappear as needed. Over these temporary ad-hoc networks services then will be requested and used, forming re-forming and adjusting the services as required. This paper will examine two key challenges in this research area of wireless ad-hoc networks.

Firstly in the formation of these wireless ad-hoc networks, it will examine addressing schemes which allow unique addresses to be given on a temporary basis to the ad-hoc wireless devices communicating. Correct ad-hoc addressing is required not only for identification but also routing and larger scale information flow.

Once the wireless ad-hoc networks are formed the next challenge is the discovery and delivery of ad-hoc services to the communicating users and devices over these networks. This forms the second part of this paper which will examine Ad-hoc Service delivery based on the Jini sub-set of the Java model.

#### 1.1. The Wireless Ad-hoc Network

“A wireless ad hoc network is a collection of autonomous nodes or terminals that communicate with each other by forming a multi-hop radio network and maintaining connectivity in a decentralised manner”[1]. The network topology of these networks can be highly dynamic as the nodes may be mobile and the links may have to contend with the side effects of radio communication, such as noise, fading and interference. Ad hoc wireless networks can range from wireless mobile networks to sensor networks, allowing short range and long range communication. Nodes in a true ad hoc network communicate directly only with other nodes within their wireless range, however beyond their wireless range they can communicate with

each other using multi-hop routes throughout the network. When nodes enter a network they must discover the ad-hoc network and request an address to join it. A dynamic address assignment protocol then provides an address suitable for use in the network.

## 2. Ad-hoc Networks Network Address Assignment

The Internet Protocol (IP) is the most popular network layer communication protocol used by ad-hoc networks. To allow nodes to communicate using IP, a number of protocol parameters need to be configured; one of these is the IP address, which uniquely identify a node within a network. This is required to ensure that packets sent to a particular node reach their intended destination. If two or more nodes share the same address an address conflict occurs, which can affect the routing process. This can result in a packet sent to a particular node being routed to the wrong destination.

### 2.1. Address Assignment Challenges

The unpredictability and dynamic nature of ad hoc networks has an adverse affect on the assignment process of address within this network. This makes the assignment of IP address in an ad hoc network more complex and introduces a number of issues not found in wired networks.

#### Node Departure

Due to the dynamic topology of ad hoc networks, nodes may depart from an ad hoc network in a random and unpredictable manner. Generally there are two types of node departure, permanent and temporary departure. During permanent network departure a node leaves the network either permanently or for a considerable period of time. Nodes that temporarily depart the network do so only for a short period. The address

assignment protocol should ensure that nodes that permanently depart the network release their address so that another node may use it. Temporary departed nodes should have their addresses protected so that another node cannot configure itself with the temporally departed nodes address, before it returns.

### Network Partitioning/Merger

Another scenario, which can occur due to the dynamic topology of ad-hoc networks, is network partitioning and merger. At any time due to topology change, a network may split into multiple partitions. In this situation any addresses that are not currently in use in the various partitions need to be recovered. A merger occurs when two or more separately configured networks are combined together to form one network. If a merger occurs duplicate addresses may exist between the two networks, as the two networks are separately configured. Network mergers need to be detected and address conflicts between the two networks should be detected and resolved.

### 2.2. IP Address Assignment in Ad Hoc Networks

In the wired environment two methods are used to configure or assign IP addresses, manual static configuration and dynamic configuration. Manual configuration requires the user to manually assign an IP address to an interface; with dynamic configuration the IP addresses are assigned dynamically as needed. For ad-hoc network; a dynamic approach is required since the user cannot be asked to continually re-configure. In the wired network dynamic configuration is achieved though the use of the Dynamic Host Configuration Protocol (DHCP)[2]. DHCP is based on a client server approach whereby a central server maintains a list of available IP addresses and assigns available addresses to new nodes as required in a client server fashion. DHCP however can't be applied to ad-hoc networks because it introduces an element of

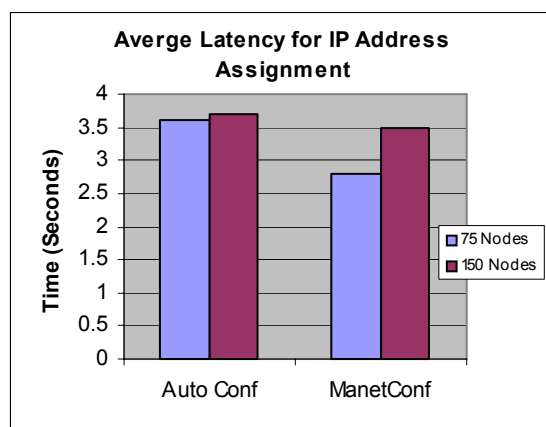


Figure 1: Latency for IP Assignment

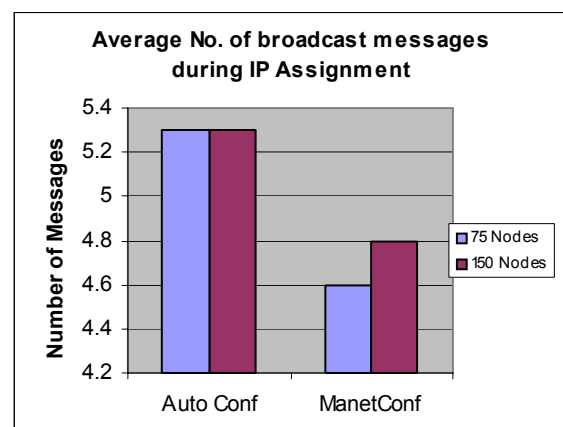


Figure 2: Average Number of Messages

central authority (DHCP Server). The distributed nature of ad-hoc networks requires that a distributed management approach be used.

### Dynamic IP Address Assignment Approaches for Ad Hoc Networks

The process of dynamic assignment of IP addresses in an ad hoc network can be divided into the two steps, which then lead onto two approaches to follow these two steps.

1. The creation of an IP address
2. Duplicate Address Detection (DAD) process. This is a process whereby a node determines whether or not a chosen address is unique within its network. A node already equipped with an IP address may also take part in DAD in order to protect its IP address from being accidentally used by another node.

#### Stateful Dynamic Assignment

In the stateful approach when a node enters into the network and requires an IP address, it chooses one configured node to assist the new node in the creation of an IP address. This configured node may assist the new node in a number of ways, for example it may choose an IP address and perform the DAD process on behalf of the un-configured node. It may also provide information to the un-configured node to assist it in the DAD process.

The ManetConf [3] Stateful approach is based upon a distributed mutual exclusion process. It requires that every node in the network maintain info on the list of allocated addresses in the network. Basically when a new node enters the network, hereafter called a “requester”, it chooses one configured node “initiator” to act on his behalf during the DAD process. The DAD process is

similar to a two phase commit process, the initiator chooses an IP address and then attempts to obtain permission from the other nodes in the network to allocate the address to the requestor. Other nodes in the network respond in the affirmative or negative depending on whether they believe that the address is in use within the network. Dealing with node departure and address reclamation is quite simple, a departure can be detected when a new node enters the network and the departed node fails to respond to an address request from the initiator.

The ManetConf [3] approach also deals with network partition and merger through the use of network ID’s. When the requestor is assigned an IP address by the initiator it is also provided with a network ID that is the same for all nodes within the network. When two nodes come into contact with each other they exchange network ID’s, if the two network ID’s are different obviously two separately configured networks have merged together.

#### Stateless Approach

In a stateless approach when a node enters into the network and requires an IP address, it creates and verifies an IP address on its own without any assistance from the other nodes in the network. Generally the new node chooses an IP address at random and then performs DAD on this address to see whether it is unique within the network.

The Auto Conf [4] proposal from the IETF MANET (Mobile Ad Hoc Network) working group can be classified as a stateless approach. In the approach the procedure for IP address assignment is basically a flooding process, whereby a node chooses a random address and then broadcasts an address request looking for this address. The node expects to receive a unicast address reply to this request when the address is already in use within the network. If no reply is received after three

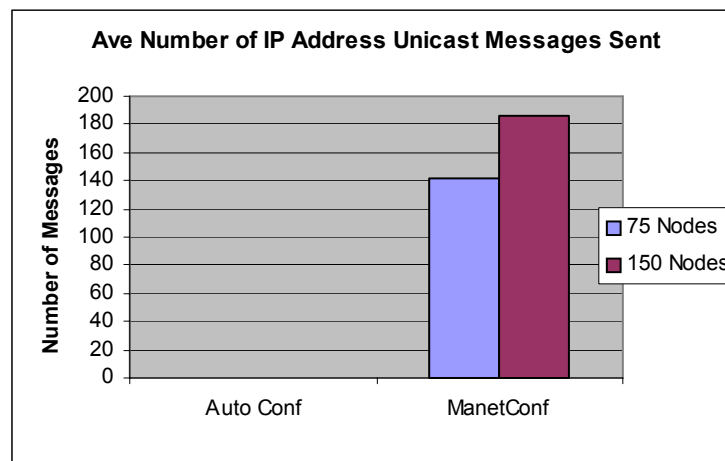
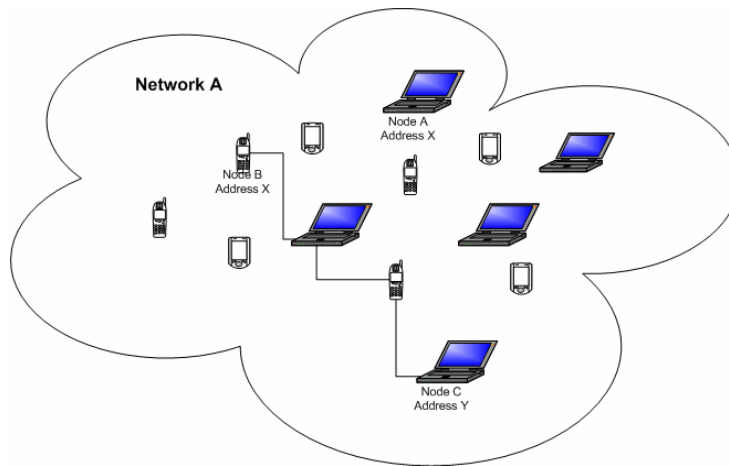


Figure 3: Average Number of Unicast Assignment Sent



**Figure 4:** Duplicate Address Problems with Service Delivery

attempts, the uniqueness check is passed. This method can easily deal with node departure and address reclamation, as departed nodes will not respond to an address request message and so their address will be released for another node to possibly use. The proposal does not however look at the problem of network partitioning and merger.

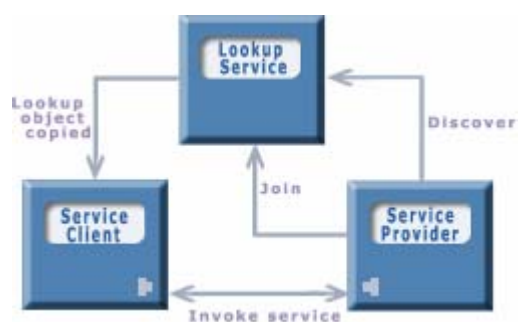
### 2.3. Simulation of Dynamic Addressing

To examine the above two proposals we have simulated them using the Communication Network Class Library (CNCL)[5]. CNCL is an event based C++ class library used for the simulation of communication networks. The primary aim of our simulations is to gather statistics regarding the address assignment latency and the number and type of messages exchanged during assignment.

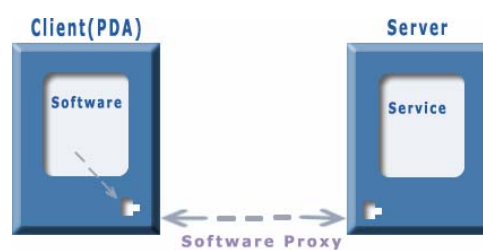
Networks of 75, and 150 nodes were simulated for 10,000s with 75% of the nodes abruptly departing the network for a random period of time. The number of possible addresses in use was set to 2000. Routing and mobility delay depended on the number of nodes in the network, with the end-to-end delay increasing the greater the number of nodes in the network.

Figure 1 shows the simulation results for the average latency involved in an IP address assignment during the simulations. Both methods are able to assign an IP Address in less than 4 seconds. In Figure 2 we can see the results for the average number of IP address assignment related broadcasts sent by each node during the simulation. From the results we can see that the ManetConf proposal is slightly superior to the Auto Conf proposal in that it generates fewer broadcasts during the assignment process.

Figure 3 shows the average number of IP address assignment related unicast messages sent by each node during the simulation. Here we can see that there is a significant difference between the two proposals. Whereas the Auto Conf proposal generates very few unicast messages (0.1 on average), the ManetConf proposal generates a large number of unicast messages. This is because unicast message are only generated in the Auto Conf proposal when a node tries to obtain an address, which is already in use in the network. By using a large pool of potential addresses (2000 in the simulation) the probability of this event occurring is very low. In the simulation this event



**Figure 5:** Joining a Jini Federation



**Figure 6:** Accessing Services via a Proxy

occurred very infrequently and as a result the average number of unicast messages generated by each node was only 0.1. The number of unicast messages generated would obviously increase through a reduction in the pool of available addresses.

#### **2.4. IP Address Assignment for IP Service Delivery**

Successful dynamic assignment of addresses with no duplication is vitally important for Service delivery based in IP networks and especially for Jini. Address duplication could cause problems in the situation, where a node requests services advertised by a particular node, which then might have a duplicate address within the network. This could result in service requests being routed to the wrong node within the network. The risk is illustrated below in Figure 4. Node C requests a service advertised by Node A with address X. A duplicate address exists in the network whereby two nodes are sharing the same address (Address X). As a result the service request by Node C may be misrouted to Node B instead of Node A.

A second potential problem could occur if a node currently involved in a Service delivery session has its address replaced with another address due to a dynamic address re-assignment following a network merge. This could result in the delivery sessions above the network layer being broken as the node has now changed its address during a service delivery session. To resolve this, when seeking address changes to remove duplication after a merge, individual nodes should have the option to veto a change, if it were to impact service current delivery. If they are forced to change they should then inform all client nodes of the impending address change. With Jini even if this is not possible, its automatic self healing mechanism, as discussed in the Section 3 of this paper will allow the automatic recovery and re-discovery of the service in the event of dynamic IP address changes due to network merges and partitions.

#### **2.5. Comparison and Conclusion - Address Assignment**

From the results we can see that overall, there is very small difference between the two proposals simulated for general performance and network traffic. However, some advantages can be found in the ManetConf proposal. Its stateful approach should allow more effective scaling than the stateless Auto Conf proposal. Another problem of the Auto Conf proposal is its inability to deal with network partitioning and merger. From these results we see that the ManetConf proposal is the most appropriate for Service Delivery over ad-hoc networks.

### **3. Ad-Hoc Networks Service Delivery with Jini**

Jini is a technology from Sun Microsystems which is intended to be a base upon which to build robust, truly distributed systems. It provides a means for software and hardware components to be integrated into one network, known as a Jini technology enabled network[6]. Jini allows its network participants both hardware and software to dynamically enter and exit a network without impacting on that network or it's users.[7] It allows developers and manufacturers to create a range of devices that can instantly connect to both wired and wireless networks in order to utilise their services, independent of the operating system or hardware present. In addition, it is possible for a Jini enabled device to access and use any service present on the network as if it were directly attached to it, or the device had been specifically configured to use that service. The Jini network technology creates an abstraction layer, which can hide the transient and unreliable aspects of an ad-hoc wireless network. This provides the auto-configuration necessary for smart devices which are intended to turn on and simply connect and work [8].

#### **3.1. Basic Jini Architecture**

A Jini network consists of a network of many Jini enabled services, which can be combined into groupings know as Federations [9]. A Federation consists of lookup servers, clients and services. Every Federation must have at least one lookup server, where all the client and service information is exchanged. A Jini Service is a software or hardware component, which provides a specific functionality within the network.

A service wishing to share its functionality must find the lookup server in the federation it wishes to join [10], as shown in Figure 5. It advertises itself through a multicast protocol called discovery, after which it registers itself, its attributes and its service proxy. The proxy is a java object, capable of accessing the service's duties and the attributes describe the service and can include a service GUI if required. The lookup server maintains a map between each service and its attributes. The Jini Client can now request from the lookup server a list of Jini services that match the requested attributes and can then select the desired service from the returned list. Once the client has selected the service it wishes to use, the lookup server downloads a copy of the service proxy and the client communicates directly with the service through this proxy, Figure 6. The lookup server no longer participates in the communication process between these two. A Client can at any time leave a federation in order to join another and access a different set of services, modelling the ad-hoc movement of real users. Likewise a Jini service can



Figure 8: Remote File Store Access via Jini

make use of other services within its federation in order to perform its own tasks, thus allowing the Jini service to act as a client to another service, providing service aggregation.

### 3.2. Managing Network Changes in the ad-hoc Environment

A Jini network is self-healing, allowing devices to leave a network without affecting the operation of the remaining devices. This eliminates network failure through a process called leasing. When using a service a client takes out a lease. Clients must periodically renew their leases, which are granted for a fixed period of time. This ensures that a client must show continued interest in a service. If a client leaves an ad-hoc network its lease expires. Similarly a service must receive a lease from a lookup server in order to maintain their availability within the federation. Services that leave the ad-hoc network similarly are removed. The lease may at any time be denied by the lease grantor and in any event expires at a predetermined time. This is a lightweight and distributed system for resource allocation makes it particularly suited to ad-hoc

networks making it virtually maintenance free.

### 3.3. Surrogate Architecture - Lightweight Jini Access

In order for a device/component to join a Jini federation it must be able to participate in the Jini discovery and join protocols and be able to download and execute classes written in the java programming language [11]. It may also need the ability to export classes written in java so that they are available for downloading to a remote entity. Many wireless devices/components may not be able to satisfy all of these requirements and thus, cannot participate directly in a Jini network. The Jini technology surrogate architecture specification provides a solution to this problem by defining a means by which these limited devices can participate in a Jini network, with the aid of a capable third party [12]. This architecture assumes the existence of a Jini capable device (surrogate host), which then connects the Jini limited device into the Jini Federation. Once the limited device has made a connection to the surrogate host, the surrogate host, will partake on its behalf in the

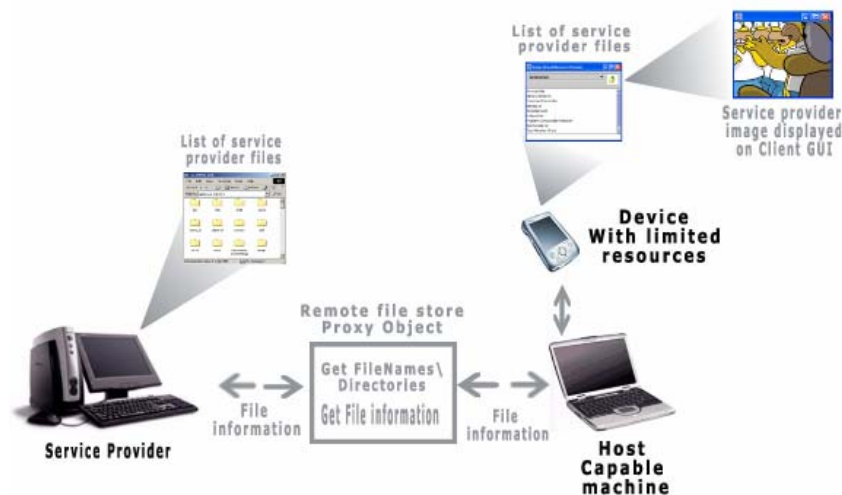


Figure 9 Surrogate Delivery of File Store Services

discovery protocol. Discovery and communication by the limited device of the surrogate host can be done at its communication interconnect protocol. e.g. Bluetooth. This provides a method for service routing in a wireless ad-hoc network of unequal devices, some without Jini. One capable surrogate host, e.g. a laptop PC, can give service providers and developers a standard way to easily and effectively connect different networks both fixed and wireless, e.g. TCP/IP, Bluetooth and Firewire, to the Jini network and the Jini technology infrastructure. Interconnect specifications define how specific connectivity technologies work within the surrogate architecture.

### 3.4. Wireless Service Provision via Jini - Test Services

In order to demonstrate the versatility of Jini within an Ad hoc network we have taken three sample scenarios and tested these over a fixed 100Mbs subnet and a 10Mbs 802.11b wireless LAN.

The first scenario examined, follows a very common Jini model, that of a software service that controls and mediates access to attached hardware, making it accessible through the Jini interfaces. We have chosen a printing service, since this is most appropriate to ad-hoc workers. This offers clients an API for printing data and controlling a number of print parameters. The service uses standard Java printing API's to cause received data to be rendered onto the a printer on whatever host the service is running.

The service proxy communicates file content and print parameters between the interfaces, back to the service back-end, the service back-end then relays this information to the host-attached printer. In operation this service allows a Jini client to render print jobs onto any Jini enabled printer regardless of its make or model or location.

The second scenario, a Remote File Store, Figure 8, is a distributed service that using a client-server architecture to enable the contents of a

remote file store to be accessed and displayed and if required a specific file content to be displayed. The "back-end" file storage service, exports a chunk of the file system via a proxy that it publishes in the Jini lookup service. Clients then use this proxy to access files on the remote system. Here both machines are independently capable of participating within a Jini federation. The Client can access files on any machine running the file store service within its federation. Alternatively if the client possesses the IP address of the lookup server on which the service is advertised that client can access a file store located anywhere in the world.

The final scenario involves a client device with limited resources. Using the surrogate architecture a PDA communicates with a host capable machine over wireless LAN. The host capable machine then communicates, on behalf of the PDA, with the Jini federation. This is the same service as previously mentioned, however it is built using the surrogate architecture. Once the mobile device establishes the connection to the host capable machine, information is passed between the host machine and the service through the service proxy. This information is then passed by the host machine to the PDA over WLAN using the IP interconnect specification.

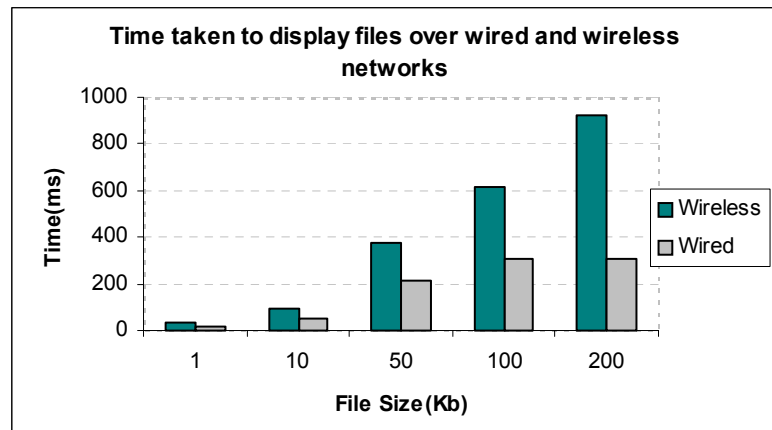
The power of this architecture is due to the flexibility allowed in the means by which the device communicates with the host capable machine. One example of this would be where the limited device is GPRS enabled, e.g a phone and can thus communicate with its host machine from any location and allow the host to communicate with the Jini network. This would allow a user to access files on their office PC, from their PDA or smartphone, while travelling on a train or in a car.

### 3.5. Results of Jini Service Delivery

Tests have shown that services behaved reliably and proved to be robust in the face of multiple client requests. While service registration can be achieved on a wired network within an average of 735ms on a 100 Mbps LAN switched Ethernet

	10K		100K	
	PC	PDA	PC	PDA
Receive Time (ms)	78	4,465	344	53,685
Display Time (ms)	184	1,612	238	1,856
Total Time (ms)	262	6,077	582	55,541
%Transfer	29.7%	73.5%	59.1%	96.7%
%Display	70.3%	26.5%	40.9%	3.3%

**Table 1:** Breakdown of Transfer Times for File Store Viewer



**Figure 10: Display times for files over Wired & Wireless Networks**

wired network, this extends to 15.23s on an 802.11b wireless network in infrastructure mode with low traffic. The large number of messages transmitted between both parties during the registration process, has lead to resource contention in the wireless Ethernet 802.11b.

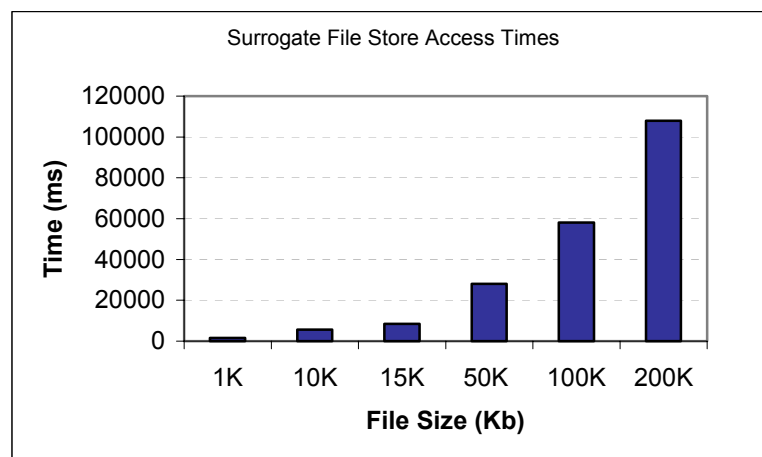
The print service succeeded in initiating printing within an average of 1.7sec when requests were sent from a client on a wired network, and within 15sec via 802.11b. Sending multiple requests resulted in jobs being queued in the order they were received. These were dealt with within an average of 2.1sec of each other.

Results from the remote file store show that the service’s directory can be navigated from the client GUI in the order of 50ms over a wired network compared to an average of 91ms over a wireless network. When testing the time taken to display various file sizes, the following results were achieved.

These results show a significant difference in the time taken to display large files over a wired

network compared to that of a wireless network. The wireless network being almost three times slower for a file 200Kb, however the scaling indicates a predictable scaling in performance. It was also observed that when multiple clients accessed the same 200 Kb file at the same time it could be displayed within 300ms on each of their displays. This shows that the service provision model scales well providing support for multiple client requests.

Using the surrogate File Store Architecture showed that connecting to the host capable machine took an average of 989ms. The list of available services was received after 158 ms and one of these services could be activated from the device in 3.4 s. Once the service was active and the GUI File store displayed, navigating through the directories took an average of 2.1s. Figure 10 illustrates the time taken to transfer and render a number of different sized files onto the user display. Table 1 compares these surrogate based PDA times with the PC based Jini times.



**Figure 11: Surrogate File Store Access Times**

From these results, we see that the time taken to display the files over the surrogate service to a PDA, as shown in Figure 11 and Table 1, is much longer than the time taken with the straight Jini architecture to the time with a surrogate, varying from 23 times for a 10K file to factor of 95 times longer for the 100KB file. The added layer of communication between the device and the host machine, and the processing limitation of the PDA severely impacted on the transfer times of the Java byte array, containing the file information.

### 3.6. Jini System Requirements

The results above and the experimentation with the surrogate architecture highlight the one of the main limitations with the Jini architecture, in that it is device demanding. Jini runs on top of Java 2 Standard Edition (J2SE), minimum version 1.2. In fact J2SE v1.3 or v1.4 is recommended. On a PC running the lookup server, the Java environment consumed 24MB of RAM. This is aside of the operating system. The standard Jeode JVM shipped with PocketPC PDAs is unable to run Jini as it does not match the full J2SE 1.2 specifications. We have used Jini on top of port of J2SE 1.31 on Sharp Zaurus PDAs and when executing it is just able to fit it into the RAM space of 32MB with the rest of the Linux OS. The results for the surrogate architecture based on J2ME are important since this is the only route that many wireless intelligent devices will be able to use to run Jini.

### 4. Conclusions

This paper has examined two basic infrastructure services needed to provide for wireless ad-hoc networking. These are dynamic IP Address allocation and service discovery and provision. Under dynamic IP address allocation the AutoConf and ManetConf protocols were examined. It was found that the ManetConf was superior both in generating less traffic and allowing for ad-hoc network merging and partitioning. For the service discovery and delivery the Jini framework has been tested and sample uses built. Here it was found that Jini provides the robust framework necessary for Services in the temporary ad-hoc environment. Its built-in resource management method of leasing, automatically makes it very suited to wireless ad-hoc applications. Its main difficulty is the overhead it can impose, this was shown in the registration times involved on 802.11b and the memory requirements of the full Jini architecture. For limited capability devices, this rules out Jini implementation except through the J2ME based surrogate architectures which have been developed to allow for simpler devices to join in a Jini

federation. This is the focus now for embedded devices.

Overall this work shows that industry strength networking based on IP and Java can work successfully in the wireless ad-hoc environment, but the devices still need to step up to the CPU/memory requirements. With the continued development of user devices, this should allow the leveraging of industry experience in service delivery via IP and Java, when developing/moving services to the wireless ad-hoc user.

### References

- [1] Advanced Network Technologies Division, NIST.  
[http://w3.antd.nist.gov/wahn\\_bkgnd.shtml](http://w3.antd.nist.gov/wahn_bkgnd.shtml)
- [2] R. Droms 1997 "Dynamic Host Configuration Protocol". Network Working Group, (Draft Standard) 2131.
- [3] Sanet Nesargi, Ravi Prakash. 2002 "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network" in Proceedings of IEEE Infocom 2002.
- [4] Perkins et al. (Nov 2001) Internet Draft "IP Address Auto-configuration for Ad Hoc Networks, Technical Report. Internet Engineering Task Force, MANET Working Group.
- [5] Communication Network Class Library (CNCL). Web Site, <http://www.comnets.rwthachen.de/doc/cncl.html>
- [6] Wong H. "Developing Jini applications using J2ME technology", Addison-Wesley, 2002, ISBN: 0201702444
- [7] Edwards W.K (2001), "Core Jini, Second Edition", Prentice Hall PTR, ISBN: 0-13-089408-7
- [8] Clark D. (1999) "Network nirvana and the intelligent device", IEEE Parallel & Distributed Technology
- [9] Newmarch J. "A Programmer's guide to Jini Technology", APress, Nov 2000. ISBN: 1893115801
- [10] Kumaran S.I, "JINI Technology: An Overview", Prentice Hall PTR, 2001, ISBN: 0130333859
- [11] "Jini Technology Surrogate Architecture Specification, Version 1.0", Draft Standard July 2001, <http://surrogate.jini.org/>.
- [12] "Jini Technology Community Surrogate Architecture Broadens Access to Jini Networks", <http://www.sun.com/smi/Press/sunflash/2001-05/sunflash.20010530.5.html>