R2D2: Rendezvous Regions for Data Discovery

Karim Seada[1], Ahmed Helmy[2]
[1] Nokia Research Center, Palo Alto
[2] Computer and Information Science and Engineering Department, University of Florida, Gainesville
kseada@alumni.usc.edu, helmy@ufl.edu

Abstract: *In large-scale wireless networks such as mobile ad hoc and sensor networks, efficient and robust service discovery and data-access mechanisms are both essential and challenging. Rendezvous-based mechanisms provide a valuable solution for provisioning a wide range of services. In this paper, we describe R2D2* **-** *a novel scalable rendezvous-based architecture for wireless networks. R2D2 is a general architecture proposed for service location and bootstrapping in ad hoc networks, in addition to data-centric storage, configuration, and task assignment in sensor networks. In R2D2 the network topology is divided into geographical regions, where each region is responsible for a set of keys representing the services or data of interest. Each key is mapped to a region based on a hash-table-like mapping scheme. A few elected nodes inside each region are responsible for maintaining the mapped information. The service or data provider stores the information in the corresponding region and the seekers retrieve it from there. We run extensive detailed simulations, and high-level simulations and analysis, to investigate the design space, and study the architecture in various environments with different characteristics. We evaluate it against other approaches to identify its merits and limitations. The results show high success rate and low overhead even with high dynamics and failures. R2D2 scales to large number of nodes and is highly robust and efficient to node failures. It is also robust to node mobility and location inaccuracy with a significant advantage over point-based rendezvous mechanisms. In addition, in networks with heterogeneous nodes, R2D2 can select nodes with specific capabilities as servers for key storage.*

Keywords: Network architecture, algorithm/protocol design and analysis, wireless multi-hop-networks, mesh networks, mobile ad hoc networks, sensor networks, resource discovery, service discovery.

[*] This work has been supported in part by grants from NSF CAREER, Intel and Pratt&Whitney.

[+] This is a significantly enhanced version of preliminary work that was presented at WMAN 2004 [20]. This work has gone major revisions and additions since then. The journal paper submission has the following additional contributions:

- The design is presented in detail with extra design challenges covered. Extensive evaluations of the architecture have been performed to cover additional design and environmental parameters, and assess the robustness of the architecture.
- Section 6 discusses new issues such as region gaps, dynamic region resizing, and empty regions.
- The performance evaluation section of the journal submission has significantly more results than the workshop paper, such as the effect of the number of servers and the number of regions on insertion, lookup, and refreshment packet overhead. It includes also the evaluation of the storage overhead and the evaluation of node failures which was not in the workshop paper.
- The journal submission includes coverage of the event model which was not included in the workshop paper.

1 Introduction

Recent research in infrastructure-less wireless networks is focused into two main categories: mobile ad hoc networks and sensor networks. There are many similarities between the two categories, but the major challenges are typically different. For efficient service provisioning, the challenges in ad hoc networks are the lack of infrastructure and the highly dynamic nature of nodes and their unpredicted mobility patterns. In sensor networks the challenges are mainly the limited resources and the extremely large number of nodes. In addition to these two types of networks, there has been recently an emerging related area of commercial importance, called mesh networks. Mesh networks share many characteristics with the previous two types of multi-hop wireless networks, but they are also characterized by the availability of static nodes, with special capabilities, that frequently act as gateways. Typical approaches for locating resources and data items in wireless ad hoc and sensor networks rely on either flooding or centralized storage. Both could suffer from scalability and efficiency problems. In this paper, we describe *R2D2* (Rendezvous Regions for Data Discovery) - a novel self-configuring, scalable, efficient and robust rendezvous-based architecture.

In infrastructure-based networks (e.g. the Internet), the rendezvous could be a logical address such as a hostname or an IP address. Logical addresses are not a feasible rendezvous in infrastructure-less dynamic wireless networks, either because they do not exist, or because their mapping to physical locations will change frequently causing high overhead. Geographic addresses provide a natural rendezvous in wireless networks, and due to the correspondence between the geographic locations of nodes and their network topology, no additional infrastructure is required other than nodes aware of their geographic locations. In our architecture, we use geographic regions as the rendezvous for the providers and seekers of information. Another alternative was to use geographic points instead of geographic regions. We chose to use geographic regions, because regions relax the requirements for location accuracy and are more robust to the topology changes caused by dynamics and mobility.

In our architecture, the network topology space is divided into rectangular geographical regions, where each region is responsible for a set of keys representing the data or resources of interest. A key, k_i, is mapped to a rendezvous region, RR_j, by using a hash-table-like mapping function, $h(k_i)=RR_j$. The mapping is known by all nodes and is used during the insertion and lookup operations. A node wishing to insert or lookup a key obtains the region responsible for that key through the mapping, then uses geographic-aided routing to send a message to the region. Inside a region, a simple local election mechanism dynamically promotes nodes to be servers responsible for maintaining the mapped information. Static nodes with desired capabilities have a higher probability for being chosen as servers. Replication between servers in the region is used to reduce the effects of failures and movements. By using regions instead of points, our scheme requires only approximate location information and accordingly is more robust to errors and imprecision in location measurement and estimation than schemes depending on exact location information. Regions also provide a dampening factor in reducing the effects of topology changes, since no server changes are required as long as current servers are inside their region and hence the overhead due to connectivity updates is quite manageable.

We run extensive detailed simulations to investigate the design space, and study the architecture in various environments including node dynamics and failures. In addition, we perform high-level simulations and analysis of R2D2 scalability and evaluate it against other approaches; flooding, centralized storage and GHT [18], to identify its merits and limitations. The results show that R2D2 is scalable to large number of nodes and is highly efficient and robust to node mobility and failures. It is robust to location inaccuracies and errors, and it relaxes the requirements for the geographic accuracy of node positions and network boundaries. It also has a low hotspot overhead and it has the flexibility to select nodes with desired capabilities as severs for key storage, which is valuable in networks with heterogeneous nodes such as mesh networks. Our goal from this comparison is to show the strengths and limitations of different approaches and under which conditions and environments each is preferable.

The rest of the paper is outlined as follows. In Section 2 we discuss the related work. In Section 3 we provide the context and assumptions under which our architecture operates. Section 4 explains the design and Section 5 contains the detailed evaluation of the architecture. Additional design issues are presented in Section 6. Conclusions are presented in Section 7.

2 Related Work

In wireless networks, the simplest form of data dissemination or resource discovery is global flooding. Flooding does not scale well. Other approaches that address scalability employ hierarchical schemes based on cluster-heads or landmarks [13]. These architectures, however, require complex coordination between nodes, and are susceptible to major re-configuration (e.g., adoption, re-election schemes) due to wireless connectivity changes or failure of the cluster-head or landmark, incurring significant overhead. GLS [14] provides a scalable location service by using a predefined geographic hierarchy and a predefined ordering of node identifiers to map nodes to their locations. GLS is presented for locating nodes and assumes that node identifiers are known. It is not clear that GLS could be extended efficiently to provide a general rendezvous-based mechanism. One way is to map keys to identifiers in the same space and let the insertion and lookup use GLS to reach the node with the closest node identifier for storage and retrieval respectively. In this case, an extra replication mechanism will be required to do reliable replication at multiple nodes. In addition, the path could be significantly long, since the insertion or lookup has to follow the sequence of location servers in order to get to the storage node. Another possibility is to use the key identifier itself to perform storage of the key-value pair in GLS servers similar to how node locations are stored. Since, in GLS the servers of a node are determined based on the node's location, the servers of a key will be determined based on the inserter location. This will create inconsistencies if multiple insertions for the same key happen.

In sensor networks communication is identified as data-centric based on the content of data rather than node identities. A data-centric routing scheme presented is directed diffusion [9]. Directed diffusion uses flooding to advertise the interests from sinks to sources throughout the network. We believe that our rendezvous-based architecture can be integrated with directed diffusion to discover resources in a scalable manner instead of using flooding mechanisms.

The original R2D2 idea borrowed from our earlier work on PIM-SM rendezvous mechanism [5] that uses consistent mapping to locate the rendezvous point (RP). However, a rendezvous *point* is insufficient in a highly dynamic environment as wireless networks. We first hinted at the R2D2 idea in [7], in the context of bootstrapping multicast routing in large-scale ad hoc networks, with no protocol details or evaluations. In the current work, we present a detailed architecture for R2D2, with full description of the design and the mechanisms to deal with dynamics, failures, and inaccuracies, and generalizing it to deal with resource discovery and data-centric architectures.

Our architecture requires nodes to know their approximate locations. Location-awareness is essential for many wireless network applications, so it is expected that wireless nodes will be equipped with localization techniques. In general, many localization systems have been proposed in the literature: GPS, infrastructure-based localization systems [25][17], and ad-hoc localization systems [4][19]. For an extensive survey of localization refer to Hightower *et al.* [8]. In all these localization systems an estimation error is incurred that depends on the system and the environment in which it is used. GPS is relatively accurate, but it requires visibility to its satellites and so is ineffective indoors or under cover. In addition, the high cost, size, and power requirements make it impractical to deploy on all nodes. Infrastructure-based localization systems are mostly designed to work inside buildings and they either have a coarse-granularity of several meters or require a costly infrastructure. Ad-hoc localization systems can have high localization errors due to environmental factors affecting the location measurements. In our design we attempt to provide an architecture that requires only approximate location information.

Several geographic routing protocols (e.g. [2][10][12]) have been proposed for wireless networks that use two modes of operation: greedy mode and perimeter mode. In greedy mode, each node moves the packet closer to the destination at each hop by forwarding to the neighbor closest to the destination. Greedy forwarding fails when reaching a dead-end (local maximum), a node that has no neighbor closer to the destination. Perimeter routing (face routing) is used to route around dead-ends until closer nodes to the destination are found. In perimeter mode, a packet is forwarded using the right-hand rule in a planar embedding of the network graph. Since wireless network connectivity is in general non-planar, each node runs a local planarization algorithm such as GG or RNG, to discard a subset of the physical links during perimeter routing, so that the resulting graph is planar.

As mentioned earlier, we presented a high-level description of the original R2D2 idea in [7]. Later on, in GHT [18] a related idea was proposed for data-centric storage in sensor networks. GHT is a geographic hash table system that hashes keys into geographic *points*, and stores the key-value pair at the sensor node closest to the hash of the key. GHT requires nodes to know their exact geographic location and uses the GPSR [10] protocol, to reach the destination. GHT uses GPSR perimeter routing to identify a packet home node (the node closest to the geographic destination). Packets enter perimeter mode at the home node (since no neighbor could be closer to destination), and traverse the perimeter that enclose the destination (home perimeter) before returning back to home node. GHT uses a perimeter refresh protocol to replicate keys at nodes in the home perimeter. The perimeter refresh protocol uses perimeter routing to refresh keys periodically, in order to detect topology changes after failures or mobility.

3 Context

In this section, we present the data models we consider in our study. We introduce two models, the *service model* and the *event model.* We then give a brief statement about the geographic requirements of Rendezvous Regions and present our assumptions.

3.1 Data Model

R2D2 provides a general architecture for data-centric storage and resource discovery. Data operations are viewed as general insertions and lookups of keys. Different applications differ in the number and characteristics of their insertions and lookups. The ratio between lookups and insertions affects the performance of our architecture; we call it *LIR* (Lookup-to-Insertion Ratio). We will define two models and identify data services and applications suitable under these models.

3.1.1 Service Model

In this model, the number of lookups is much larger than the number of insertions. Hence, this model has large LIR. Stored data is long-lived and queried continuously by large number of nodes. Any node in the network can perform insertion or lookup. This model is viewed as a general service location model and several applications could be mapped to it:

- Service location & bootstrapping: In ad hoc networks, nodes can use R2D2 to identify and find new services. The service provider maps its service to a region ($RR_i = hash(service)$) and stores information about the service (e.g. location) there. Nodes interested in the service use the same hash function to get the information from that region.

- Users and object tracking: Users (or tracked objects) in the ad hoc network map their identities to the corresponding region and store their location there. Other users use the same mapping to retrieve the location from that region.

- Configuration & task assignment in sensor networks: During the startup of the network operation, it is assumed that nodes' configuration and tasks are flooded to the entire network. Global flooding is inefficient if nodes are not joining simultaneously, new nodes are added, or tasks of different nodes are changing with time. R2D2 provides a flexible way for nodes to get this information. Each node will get its task from a rendezvous region computed as a hash of the node location or other properties of the node environment (e.g. RR_i = *hash(location or properties)*). Operators assign tasks to nodes by sending them to RR_i using the same hash function.

- Data-centric storage for events that do not change frequently, such that the number of detected events is lower than the number of queries sent by nodes. For example, detecting a certain phenomenon in an area, which is required by other sensors to perform their computations, or required by actuating devices in order to decide on certain actions.

- Providing global properties of the network or environment that are of interest to many nodes.

3.1.2 Event-based Model

This model has large number of insertions with a lower number of lookups. An example is data-centric storage with many events detected from a set of event types. A query looks for the events detected of a certain type. Each event type is hashed into a rendezvous region and all events detected of that type are forwarded there. Events detected cause insertions, while queries for events are the lookups. Aggregation could be valuable in this model, where a query for a certain type returns an aggregation of all events detected for that type. In this model any node in the network detects and inserts events, while queries are normally sent by external nodes through specific access points. Examples of some applications:

- Data-centric storage for events that do change frequently, such that the number of updates is much higher than the number of queries sent by nodes.

- Database querying: Much ongoing research views the sensor network as a database and using declarative SQL-like queries for retrieving information [6][1]. R2D2 can provide an in-network storage component in these schemes. For example, temperature sensors send the temperatures of their regions to 'RR_i = *hash(temperature property)*' (the region responsible for storing temperature information) and a node interested in the max temperature sends a query 'Select MAX temperature' that retrieves this information from RR_i.

3.2 Geographic Requirements

A main objective of our architecture is to *relax the requirements for exact geographic information*. Nodes need only to know their regions; exact location is not required. We design our system as an *overlay* that can run over different routing protocols. The routing protocol need not be a geographic routing protocol that requires the exact geographic positions of nodes. It can be any routing protocol augmented to provide approximate routes towards regions. Relaxation of the dependence on geographic information is valuable for many reasons: Geographic information is often unavailable and in most cases it is inaccurate. Obstacles and non-ideal radio ranges can cause many problems to current geographic routing schemes. In addition, mobility increases the inaccuracy of location updates.

In our previous studies, we show how location errors, caused by localization systems and inaccuracy [22], inconsistency of location dissemination [11], or node mobility [24], result in severe performance degradation and correctness problems in geographic routing protocols. It is therefore crucial to provide alternative architectures that relax the assumptions of availability of accurate location information. Our Rendezvous Regions approach provides one such architecture and is more robust to errors and imprecision in location measurement and estimation than schemes depending on exact geographic location. In the results section, we show the effect of location inaccuracy on R2D2 and GHT.

In [23] we studied geographic routing under more realistic wireless models and presented efficient strategies for forwarding on lossy wireless links. These strategies could be used here as well to improve the robustness of geographic routing in real-world environments that violate the assumption of the perfect wireless radio ranges commonly used in simulation tools.

3.3 Assumptions

The network space is divided into rectangular equal-sized regions (see Figure 1), where the size of the region is set based on the radio range and how many hops we want the region to cover. The region size we use is covering a few radio hops, to provide adequate relaxation of the inaccuracy and mobility effects, while keeping the region flooding overhead and server load reasonable. In the simulations we study the effect of the region size in more detail. We assume that the geographic space and boundaries of the network are known and that each node has a localization mechanism to detect its approximate geographic location and accordingly its region. Our design also allows us to relax the requirements for exact boundaries by having boundary regions instead of boundary points. Since nodes know the network geographic space boundaries and the region size, they can determine their regions within the space. Using appropriate density, we assume that the network is connected and that each region has nodes in it, which is a valid assumption under a reasonable density. In case of partitions and empty regions, multiple hash functions can also provide substitute regions, when the original region is not available. In Section 6 we will discuss empty regions, gaps, and the region size in more detail.

4 Design Overview

The network topology space is divided into geographical regions (*RRs*), where each region (e.g., RR_j) is responsible for a set of resources. The resource key space is divided among these regions, such that each resource key (K_i) is mapped to a region. The key-set to *RR* mapping ($KSet_i \leftrightarrow RR_j$) is known by all nodes. For example, a service provider will send the key of its service towards the mapped region and a service seeker will use the same mapping to locate the service. Inside each region, a set of elected servers is responsible for holding these resources' information. We introduce additional mechanisms to maintain consistency in case of mobility or failures.

The Rendezvous Regions scheme can be built on top of any routing protocol that can route packets towards geographic regions. The only requirement of the routing protocol is to maintain approximate geographic information, such that given an insertion or lookup to a certain region, it should be able to obtain enough information to route the packet towards that region. Given that the packet is able to reach the region, there are several design options inside the region itself. These design options affect the operation of insertions, lookups, server election, and replication inside the regions. They also affect the consistency operations for mobility and failures. In our experience, it is usually the case that no one design option gives best performance under all operating conditions. The design choices depend on many factors such as the environment, application, and the pattern of insertions and lookups. In this work we attempt to investigate and study several design options to identify their strengths and weaknesses. There are mainly three options for forwarding packets inside the region: (a) Geocast, (b) Anycast, and (c) Unicast.

(a) *Geocast*: By geocast we mean sending the packet to all nodes in a geographical area. Geocasts suffer from a high overhead, but are practical when we want to send the packet to several non-determined nodes (in our case to the servers) within a specific geographic region. It is also robust in the face of dynamics and does not depend on the underlying routing protocol. In Section 6.1 we explain the geocasting protocol used in more detail. For a study of efficient geocasting protocols see [21].

(b) *Anycast*: Anycasts are used when it is sufficient to reach any node of a set of nodes (any server). It has similar advantages to geocasts and can be implemented by using expanding ring search techniques or by caching previously known servers.

(c) *Unicast* to servers: Direct unicasts can be used when the locations of servers are well-known, either because the servers are fixed or because they coordinate to keep their information up-to-date. This has the advantage of low overhead during insertions and lookups, but if the servers are dynamic, extra periodic overhead for maintaining the servers' information is required. Unicasting may also depend on the underlying routing protocol to keep track of nodes' locations. The additional periodic overhead can be acceptable if the insertion or lookup rates are high.

In our current design we use geocasts for insertions and anycasts for lookups. These design choices are simple to implement, robust to dynamics, and do not require tracking of nodes' locations. In the following we describe the main components of our architecture.

4.1 Region Detection

Using a localization mechanism [8], each node detects its location and accordingly its geographic region. When the node moves, it detects its new location and so it keeps track of its current region. The node uses this information to forward packets towards regions, detect packets forwarded to its region, and potentially participate in server election in its region (if and when needed) (Figure 1).

4.2 Server Election

A simple local election mechanism is used inside the region to dynamically promote the servers. The number of servers required in the region is called *S*. As *S* increases, the robustness to mobility and failures increases, but also the storage overhead increases. In the simulations we study the effect of *S* in more detail. Servers are elected on-demand during insertions. When a data insertion operation is issued, the first node in the region that receives the insertion[1], known as the *flooder*, geocasts the insertion inside the region. Each server receiving the insertion geocast sends an Ack back to the flooder. The flooder keeps track of the servers and if it does not get enough Acks (the minimum number of servers required), it geocasts again and includes a self-election probability, *p*, in the goecast message. Each node receiving the geocast, elects itself with probability $p*c$, where *c* is a factor determined based on certain node capabilities such as stability. If the node becomes a server, it replies to the flooder. If not enough Acks are received, the flooder increases *p* based on a back-off mechanism until the required number of servers reply or *p* reaches 1. When servers move out of the region or fail, new servers are elected in the same way. After the new servers are elected, they retrieve the stored keys from other servers.

4.3 Insertion

A node inserts a key, *K*, by first mapping the key to a rendezvous region, RR_i, where $K \in KSet_i \leftrightarrow RR_i$. The node generates a packet containing the region identifier, RR_i, in its header. Nodes routing the packet toward the region, check the region identifier to determine whether they are in or out of region. The first node inside RR_i to receive the packet, the *flooder*, geocasts the packet inside the region. Servers inside the region receive the geocast, store the key and data, then send Acks back to the flooder (Figure 2). The flooder collects the Acks and sends an Ack back to original sender. If no Ack is received by the sender, it timeouts and retransmits the insertion up to a fixed number of times.

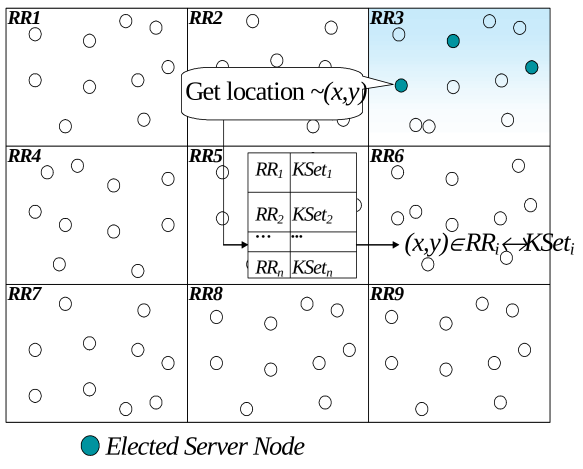


Figure 1: By knowing their approximate location ~(*x*,*y*), and henc region (*RR*), nodes in every region (shown in *RR3*) conduct loca elections. Only a few nodes (the elected servers) maintai information about the resource keys (*KSet*$_3$) mapping to *RR3*.

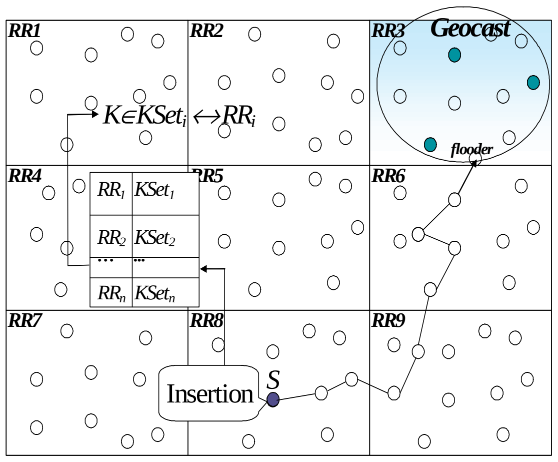


Figure 2: Insertion (step I): Node *S* wishing to insert (or store) resource key *K* that belongs to *KSet*$_i$ gets the corresponding *RR* (in this case *RR3*) through the mapping ($KSet_i \rightarrow RR_i$). (step II): Node *S* sends the resource information towards *RR3*, where it is geocast by the flooder and stored by the servers.

4.4 Lookup

Lookups are similar to insertions except that nodes and previous flooders inside a region cache locations of the recent servers they hear from, and send the lookups directly to any of the servers (anycast). The server replies to the flooder and the flooder replies back to original sender (Figure 3). If the flooder receives no reply or if it has no cached servers, it geocasts the lookup inside the region.

4.5 Replication

Replication is inherent in this architecture, since several servers inside the region store the key and data. This adds extra robustness to failures and mobility. For additional robustness against severe dynamics such as group failures and partitions, multiple hash functions may be used to hash a key to multiple regions.

4.6 Mobility

Local movements of nodes and servers have negligible effect and overhead on our architecture as long as servers stay within their regions. The only condition we need to consider is when a server moves out of its region. The server checks its location periodically to detect when it gets out of its region, in order to send an insertion packet containing its stored information toward that region so that new servers are elected. The server then deletes its stored keys and is not a server anymore. It may or may not get elected again later in a new region.

[1] A node can identify that it is the first node in the region to receive the packet by a simple flag set in the packet header.

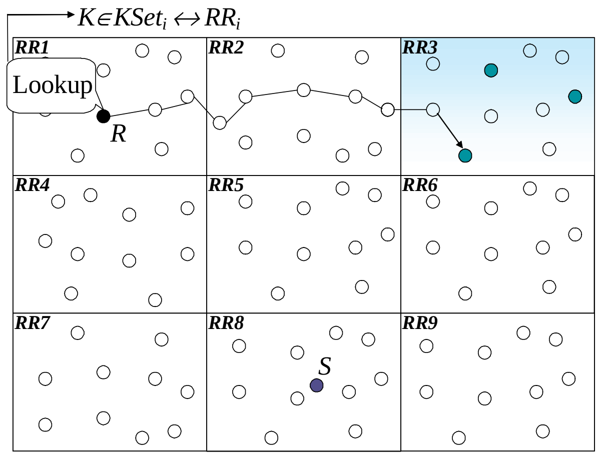
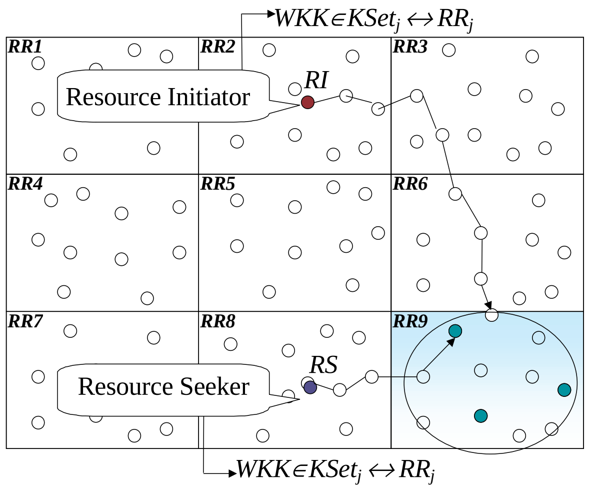


Figure 3: Lookup: Node *R* looking for a resource with key *K* that belongs to $KSet_i$ gets the corresponding *RR* (in this case *RR3*) through the mapping ($KSet_i \rightarrow RR_i$). Then sends the resource lookup towards *RR3*, where it is anycast to any server holding the information.



$WKK \in KSet_j \leftrightarrow RR_j$
RR1
RR2
RR3
RR4
RR5
RR6
RR7
RR8
RR9
Resource Initiator
RI
Resource Seeker
RS
$WKK \in KSet_j \leftrightarrow RR_j$

Figure 4: Using the same mechanism for obtaining the mapping function. A bootstrap mechanism for initiating (and obtaining) new resource mappings using *well-known key(s) (WKK)*.

4.7 Failures

Since each region contains several servers, and insertions and mobility may invoke new server elections, it is unlikely that independent reasonable failures will cause all servers to vanish. In order to avoid this case, servers use a low-frequency periodic soft-state mechanism during silent (low traffic) periods, to detect failing servers and promote new servers. Each server runs a low-frequency timer, which is reset each time an insertion geocast is received. When the server times out, it geocasts a packet checking for other servers. Other servers reset their timers upon receiving this check and reply back demonstrating their existence. If not enough servers reply back, server election is triggered.

4.8 Bootstrap

One question remaining is how the mapping function is obtained. Similar systems normally assume that it is pre-known or provided by out-of-band mechanisms. In our architecture, using the same rendezvous mechanism, we provide a bootstrap overlay to publish dynamic mappings. Using the mapping for a *well-known key*, a node sends request to a well-known region to obtain the mapping function of a set of services (Figure 4). These mappings however are not expected to change frequently. This introduces more flexibility for providing different mappings for different type of services and changing them when required. It also allows for incremental expansion of the services provided.

5 Performance Evaluation

In this section, we evaluate our architecture using detailed NS-2 [16] simulations with detailed models of the wireless MAC and physical layers. We run extensive simulations to investigate the design space, and study the architecture in various environments including node mobility and failures. We study also the effect of inaccurate location. R2D2 is running as an overlay over the routing layer and we are using GPSR [10] as the wireless routing protocol. We modified GPSR to route to regions instead of specific destinations by forwarding the packet toward the center of the region and using geocast or anycast inside the region. We verify the correct operation of R2D2 and evaluate its performance under different scenarios. In addition, we run similar detailed experiments for GHT [18] in order to compare the performance of both systems and identify their characteristics and limitations.

We conducted simulations with up to 400 nodes using the detailed model. To study the scalability of the architecture to higher number of nodes, we perform also higher-level simulations (without the MAC and physical layers) for networks with up to 100,000 nodes and compare it to GHT, flooding (local storage), and centralized storage (external storage).

We evaluate two models, the service model with high lookup-to-insertion ratio (*LIR*) and the event model with low *LIR*. There is a wide range of parameters we consider during the evaluation. The environment parameters include the network dimensions, number of nodes (density), transmission range, and the expected number of lookups per second. The design parameters that we study are mainly the number of regions and the number of servers per region. The performance metrics are the success rate of lookups, message overhead per insertion, message overhead per lookup, and total storage/insertion, in addition to the maximum node overhead of these metrics. We also study the overhead due to mobility and failures.

5.1 Detailed Simulation Results

We implemented R2D2 in NS-2 as an overlay that runs over any wireless routing protocol that can route towards geographic regions. Currently, we are using GPSR, modified to route to a region, with detailed 802.11 MAC and physical layers. The GPSR beacon interval is 1 sec and the beacon expiration is 4.5 sec. The density is fixed to $1/1024m^2$ and the number of nodes is 100, 200, or 400. We explored several transmission ranges between 60m and 120m. For convenience (and space limits), in the results shown, we focus on 100-200 nodes topologies and 80m transmission range. The rate of lookups is 2 per second and the number of retransmissions for both insertions and lookups is 3. The periodic failure check interval is 20 seconds. Keys are uniformly distributed at random over the space. The results are the average of 5 random runs over 5 different random topologies. GHT was already implemented in NS-2. The refresh interval for GHT is set to 10 seconds. A common problem in GHT that affects its performance, is when a key hashes to a point outside the external perimeter. In this case, perimeter routing may move around the whole external perimeter during insertions, lookups, or refreshes. To reduce the effect of this problem during evaluation, we modified GHT to avoid mapping the keys to points close to the space boundary. We do that by excluding 10% of each side (left, right, top, bottom) of the space during hashing. We will refer to the modified GHT by GHT*.

5.1.1 Number of Servers in Region

First, we study the effect of the number of servers in a region. In the simulations, a value is set for the minimum number of servers, below which flooders ask for new servers. Since the server self-election is probabilistic, the actual number of servers in a region could exceed this value. The experiment is run over a 100 node static topology with 4 regions and a LIR equal to 10. The number of insertions varies between 10, 30, and 50. The results obtained are the message overhead/insertion, message overhead/lookup and storage overhead/insertion with different number of servers. From these results we computed the normalized message overhead as *Norm = Ins + Lookup * LIR*, where Ins and Lookup are the overhead/insertion and overhead/lookup respectively. Since, the optimum number of servers depends on LIR (the ratio of lookups to insertions), we show in Figure 5, the normalized overhead for different values of LIR. With low LIR, low number of servers causes lower overhead, while high LIR favors more servers. As we notice, 3-4 servers give a good compromise. Another tradeoff is that the total storage overhead per insertion increases with the number of servers, but also more servers lead to higher robustness in case of mobility or failures.

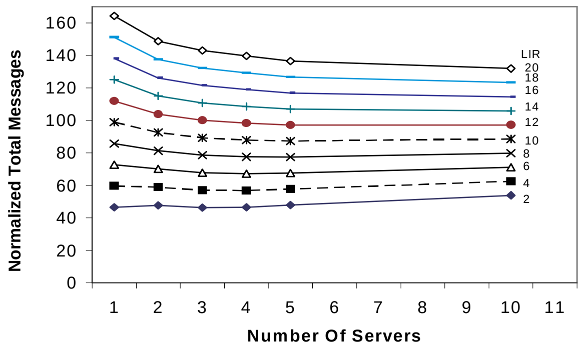


Figure 5: Normalized overhead for different number of servers

In the coming detailed experiments, the number of servers is set to 3. The simulation run is 200 seconds with 30 insertions at the beginning and 300 lookups at a rate of 2 new lookups per second. Nodes are chosen at random for insertions and lookups. The number of regions varies from 4, 9, 16, to 25. We will start with static networks, then look at failures and mobility.

5.1.2 Number of Regions

The number of regions in the space is an important design parameter that indicates the region size and the average number of nodes in region. Ideally we like to have regions with small size to reduce the geocast overhead and at the same time to have the region size large enough to relax the geographic accuracy and reduce the effects of mobility. In static networks, the success rate for both R2D2 and GHT is almost 100%, so we consider only the overheads in this section. Figure 6 shows the insertion overhead of R2D2 compared to GHT and GHT*. R2D2 overhead decreases by increasing the number of regions (i.e., reducing the region size, since the network size is fixed), due to the reduction in the region geocast overhead. At low number of regions, GHT has lower insertion overhead, but they become closer as we increase the number of regions[2]. In Figure 7, the region size has less effect on R2D2 lookup overhead, since the main factor here in reducing the overhead is caching the servers and anycasting them. Both GHT and GHT* have a significantly higher lookup overhead than R2D2. GHT* has a lower overhead than GHT, since it excludes hashing points close to the network boundary, which can cause the external perimeter traversal, but it is still higher than R2D2. Insertion and lookup overhead in GHT are similar, since they are using the same mechanism.

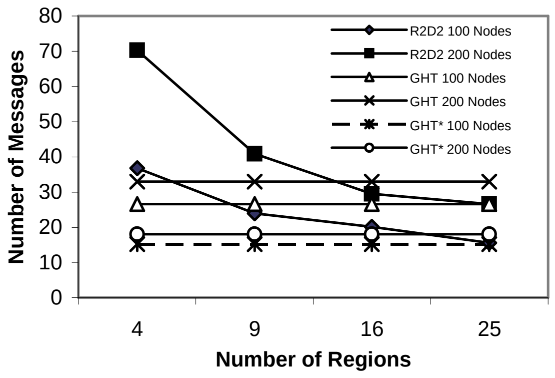


Figure 6: Insertion overhead for R2D2 vs. GHT & GHT*

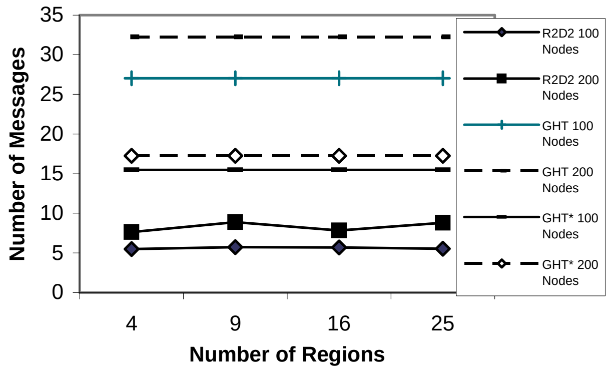


Figure 7: Lookup overhead for R2D2 vs. GHT & GHT*

The high overhead of GHT is due to the home perimeter routing. Each packet in GHT (insertion, lookup, or refreshment) goes to the home node (closest node to destination point), then it traverses the entire perimeter (home perimeter) that encloses the destination, before returning to the home node. The average perimeter length is somewhat long, which can be noticed also from Figure 8, where the average storage overhead for 100 Nodes is around 8 in GHT* (18 in GHT), which means that on average 8 nodes are in the home perimeter and store the packet. This is consistent with the insertion and lookup overheads, which are around 15 messages (including the forwards to home node and back replies). By looking at the randomly generated topology in Figure 9 we can understand why average perimeters are so long: 1) even with avoiding the surrounding boundary points in GHT* and using only the internal 64% (.8*.8) of the space for hashing, there are still many areas where a hash point there could cause external perimeter traversal, 2) even with random uniform distributions there are still somewhat large internal areas without nodes. The keys that hash in these areas have long home perimeters, 3) this may be the most interesting, which is the effect of planarization. Planarization (in this case Gabriel Graph)

[2] Notice that we are using simple direct flooding, where each node inside the region broadcasts the packet once. Alternatively we can use smart flooding

excludes many links from perimeter routing, such that even in higher dense networks, packets still have to traverse long perimeters because shortcuts between the perimeter nodes are not included in the planar graph[3].

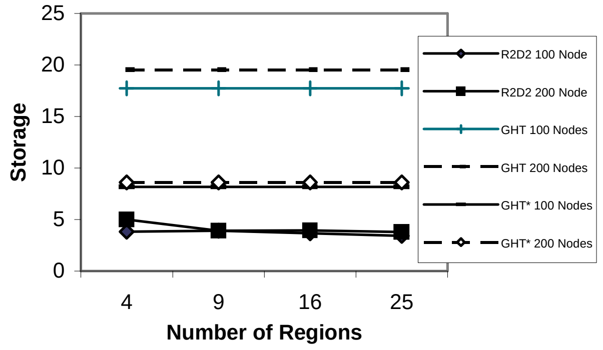


Figure 8: Storage overhead (per insertion) for R2D2 vs. GHT & GHT*

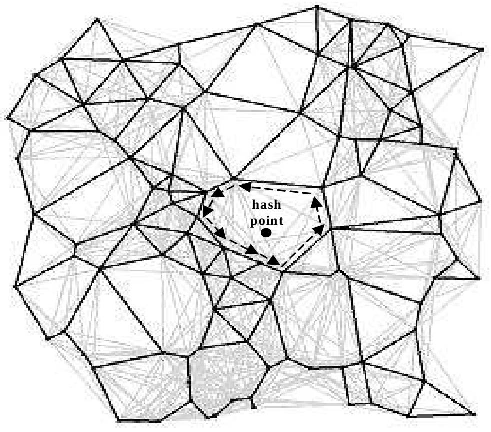


Figure 9: Example of a random uniform topology where perimeter traversing (in a GG planarized graph) takes many hops. Gray edges are the physical links not included in the planar graph

Figure 10 shows the periodic failure check overhead in R2D2 compared to the refresh overhead in GHT. In R2D2 it is close to the number of nodes, since in each region, where there are keys stored, one of the servers geocast and so there is only a single geocast in each region independent of the number of keys stored there. In GHT the periodic overhead is much higher, because the refresh is sent for every key around its home perimeter (every key has a different hash point), so the overhead increases with the number of keys stored. For example, in this case we have 30 keys with an average perimeter of 8 nodes in GHT* (18 in GHT) for 100 node topologies, which gives around 240 messages as shown in the figure (GHT* around 250 and GHT around 500).

These results show that the total overhead depends on the data model and the LIR. For example in the service model, R2D2 has a lower overhead due to the large LIR, while in the event model with larger number of insertions, GHT may have lower overhead. In order to evaluate the total overhead taking into account the insertions, lookup, and periodic overhead, we compute the normalized total overhead as

$$Norm = \frac{Ins}{LIR} * iRate + Lookup * lRate + \frac{Per}{len}$$

where *Norm* is the normalized total overhead per second, *Ins* is the overhead/insertion, *Lookup* is the overhead/lookup, *iRate* and *lRate* are the insertion and lookup rates respectively, *Per* is periodic overhead in an interval, and *len* is the length of the interval. Plotting this equation at different LIRs, we show in Figure 11 the point where R2D2 and GHT* overhead intersect. This is beneficial in deriving which system is more efficient in a certain situation. The overhead ratio between R2D2 and both GHT and GHT* is shown in Figure 12, for

techniques [15] which will reduce the geocast overhead significantly.

[3] We are using a density of 1/1024m^2 with an 80m transmission range; in GHT [18] the density is 1/256m^2 with a 40m transmission range, which gives the same connectivity. In order to verify that, we run the same experiment with the higher density and lower transmission range of [18] and we got the same results.

different lookup rates (LR). For *LIR>0.1*, R2D2 incurs less overhead than GHT*, with the savings approaching 80% for *LIR>10*.

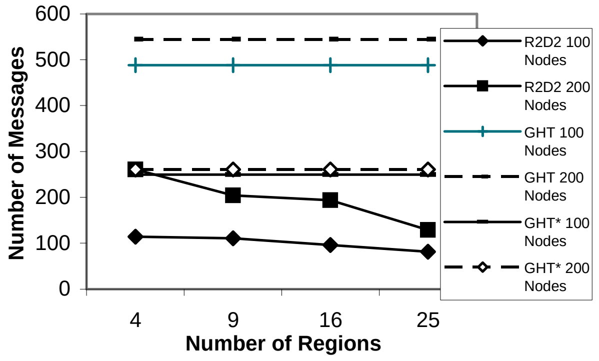


Figure 10: Periodic (refresh, failure check) overhead for R2D2 vs. GHT & GHT*

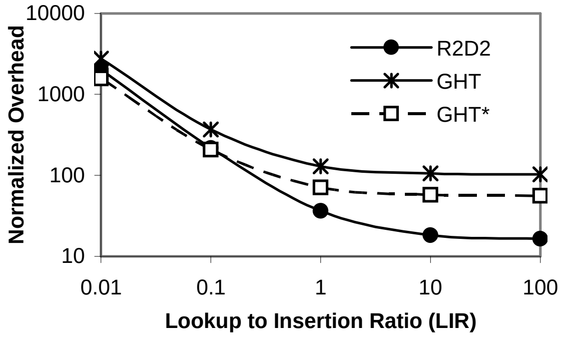


Figure 11: Normalized total overhead per second for different LIR

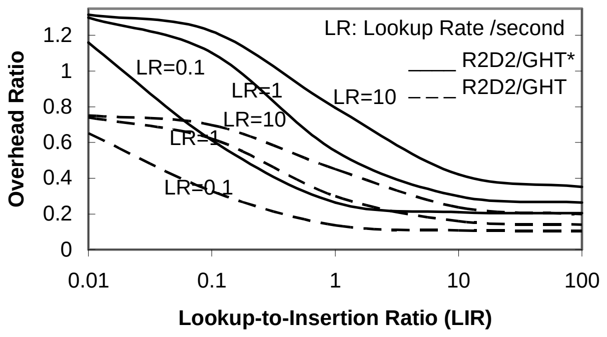


Figure 12: Overhead ratio for different LIR and different lookup rates (LR)

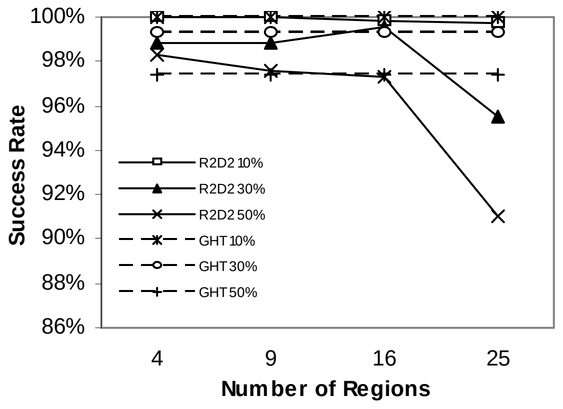


Figure 13: Lookup success rate for different node failure rates

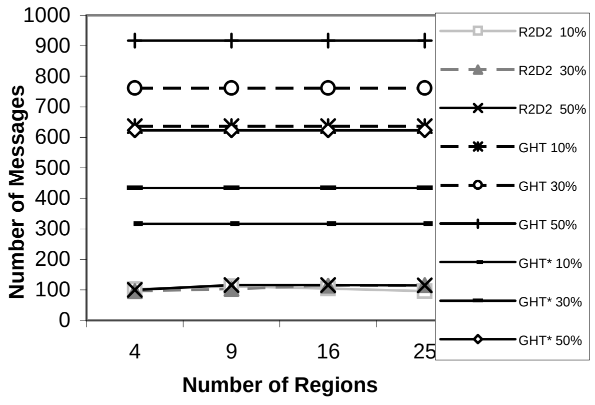


Figure 14: Periodic overhead for different node failure rates

5.1.3 Failures

In this section, we show the robustness of R2D2 with node failures. Robustness is achieved by the replication among servers and the periodic failure checking mechanism. We run simulations of 100 nodes where 10%, 30%, or 50% of the nodes fails at random times. In Figure 13, the success rate remains high with failures in both GHT and R2D2 (around 97% when 50% of the nodes fail). With 25 regions and 100 nodes in R2D2, the success rate falls down because on average we have only 4 nodes per region, and less than that with failures. By setting the region size appropriately we can avoid this problem. Figure 14 shows the periodic overhead with failures; R2D2 overhead increases slightly but remains around the number of nodes, while GHT overhead increases significantly due to the extra perimeter traversals for key refreshments after node failures.

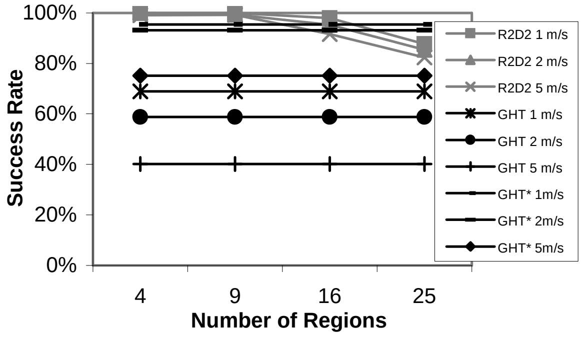


Figure 15: Lookup success rate for different node mobility rates

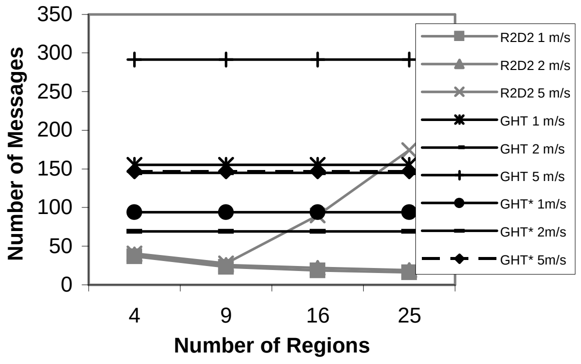


Figure 16: Insertion overhead for different node mobility rates

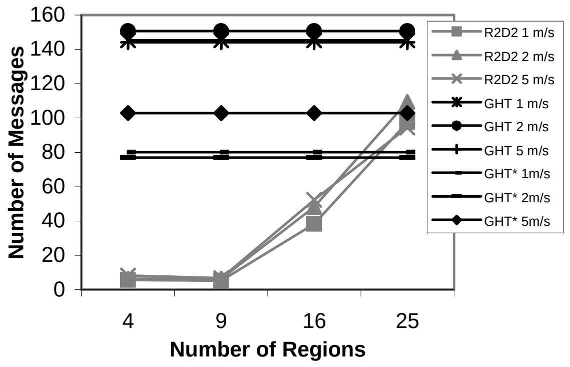


Figure 17: Lookup overhead for different node mobility rates

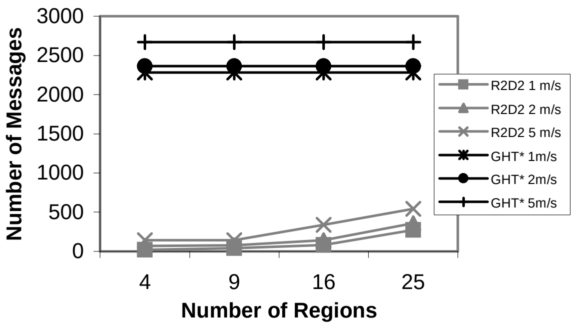


Figure 18: Mobility update (refresh) overhead in R2D2 and GHT

5.1.4 Mobility

R2D2 is very robust to node mobility. The main reason is that local movements as long as the servers remain in the region, do not require change of servers or any extra overhead. Mobility updates happen only when a server moves out of region. In this experiment, nodes are moving using the random waypoint model [3], with a maximum rate of 1m/s, 2m/s, or 5m/s. There is no pause time, all nodes are moving continuously. Figure 15 shows the high success rate of R2D2 under mobility. It drops only with higher number of regions, because of the low number of nodes per region, which is also the reason for the insertion and lookup overhead increase in Figure 16 and Figure 17, because of the extra geocasts and retransmissions looking for servers in a certain region. This problem does not exist with higher number of nodes and can be avoided by setting an appropriate region size. For example, with 100 nodes moving at 5m/s, 9 regions give a success rate above 99%. GHT success rate drops faster with mobility, since any small movements can cause changes in the key storage.

Lookup packets reach home nodes (nodes closest to destination point) that do not have the key due to changes in topology. In addition, perimeter traversal with mobility is susceptible to loops, which may cause the packet to exhaust its TTL. We used a 2 second replanarization timer with GHT to reduce this effect[4]. In Figure 18, we show the mobility update overhead during an interval equivalent to GHT refreshment interval, since both of them reflect the overhead due to mobility (in addition, GHT performs refreshes when new node joins or leaves are discovered). R2D2 has much lower overhead, since only the servers send updates when they move out of region. In Figure 19 and Figure 20, we fix the number of regions of R2D2 to 9 and change the pause time of nodes moving with a maximum random-waypoint velocity of 5m/s. We notice the high success rate and the low overhead compared to GHT and GHT*.

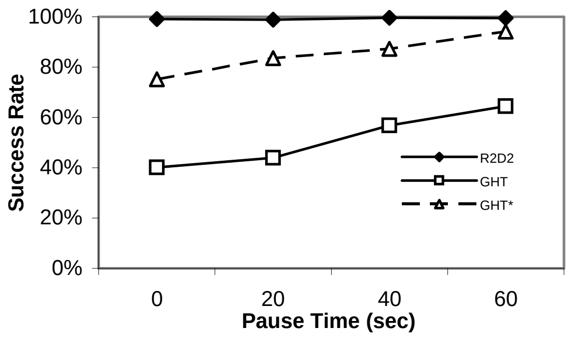


Figure 19: Lookup success rate for different node pause times

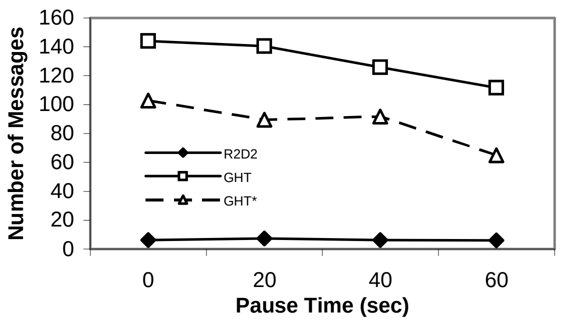


Figure 20: Lookup overhead for different node pause times

5.2 Location Inaccuracy

In this section, we study the effect of inaccurate location information on R2D2. One of the main objectives of R2D2 is to relax the accuracy required by nodes in estimating their location. In this section we consider only the routing behavior in an ideal wireless environment, in order to evaluate the effects of location inaccuracy without the interference from other layers such as MAC collisions or physical layer effects. We use a static and stable network of 1000 nodes having the same radio range and density as in the previous section. Results are computed as the average of 1000 runs, where in each simulation run, nodes are placed at random locations in the topology and 10 insertions and 100 lookups are generated by random nodes. The success rate is the percentage of successful lookups. The maximum localization error is presented as a fraction of the radio range. The estimated node location is picked uniformly from a random location around the node accurate position limited by the maximum localization error.

Figure 21 shows the success rate of R2D2 with different number of regions (16, 36, and 64 regions) compared to GHT at low inaccuracy range (2-10% of the radio range). R2D2 is more robust to location inaccuracy than GHT and the effect of inaccuracy is less on larger regions. Figure 22 has a higher inaccuracy range (20-100%),

[4] In [18] the TTL was also limited during refreshments to reduce the overhead if looping happens, but we have not included this change here, since it is not clear how the TTL can be set dynamically without extra overhead and what its effect is on other functions of the protocol.

which shows the effects of inaccuracy more clearly. For example, at an inaccuracy equal to 60% of the radio range, GHT success rate goes below 60%, while R2D2 is still above 95%.

By adding the fix we introduced in [22] to GHT[5] and the geographic routing in R2D2, the success rate improves significantly. At low inaccuracy range, both R2D2 and GHT have a success rate higher than 99.8%, with larger regions still having better rate, but the differences are small. Figure 23 shows the success rate at high inaccuracy range. With an inaccuracy up to 80% of the radio range, both R2D2 and GHT have above 95% success rate. At an inaccuracy up to the whole radio range, R2D2 is above 90% and GHT above 80%, which is a significant improvement.

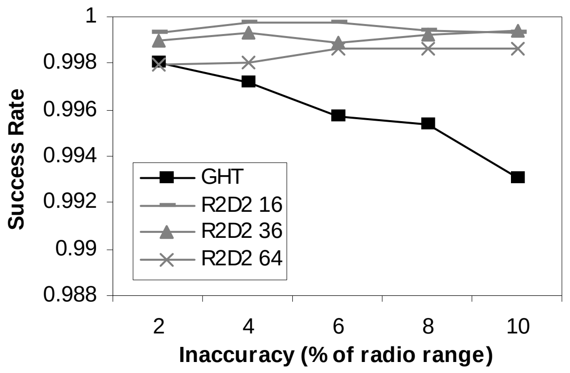
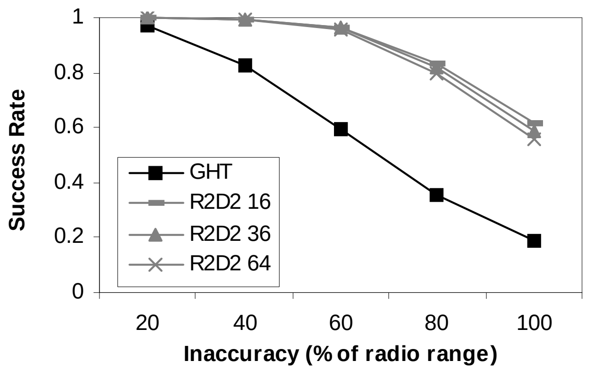


Figure 21: Success rate at low inaccuracy range (2-10% of radio range)



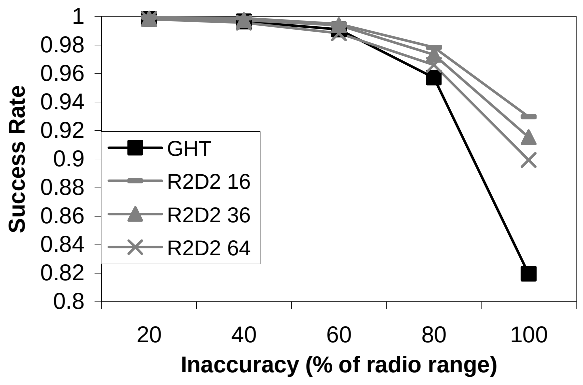


Figure 22: Success rate at high inaccuracy range (20-100% of radio range)

Figure 23: Success rate at high inaccuracy range (20-100% of radio range) with the fix added

5.3 High-Level Simulation & Analysis

To evaluate the scalability of R2D2 to higher number of nodes, we perform high-level simulations without the wireless MAC and physical details. We compute the total message overhead and the hotspot message overhead (maximum node overhead) in a static network. In a sensor network, the overhead reflects also the energy consumption of a sensor node and the lifetime of the whole network. Without errors or dynamics such as failures and mobility, the success rate need not be computed (always 100%). We compare R2D2 to GHT, flooding (local storage), and centralized storage (external storage). We count only the insertion and lookup overhead. The periodic refreshment overhead in GHT and the failure check overhead in R2D2 are not included, but we can easily compute an estimate for them based on the detailed simulations results. In GHT, we do not use hash points that lead to long perimeter traversal (more than square root the number of nodes), in order to avoid the high overhead of the external perimeter traversal.

We will first do some approximate analysis for the communication overhead of the mechanisms. We will consider general insertion and lookup operations, where *n* is the number of nodes, *I* is the number of insertions, *L* is the number of lookups, *R* is the number of regions in R2D2, and S is the average number of servers per region. We will use the asymptotic expression of $O(n)$ for flooding the whole network, $O(\sqrt{n})$ for point-to-point routing, and $O(n/R)$ for region geocasts. In flooding we assume that the nodes store (insert) their keys locally and other nodes flood (lookup) to get them. In centralized storage, we assume a centralized node, storing all the keys, so that all insertions and lookups are forwarded to it. The following table shows the asymptotic insertion and lookup message overhead:

	Total message overhead	Hotspot message overhead
Flooding	$O(n) \times L$	$O(L)$
Centralized	$I \times O(\sqrt{n}) + L \times O(\sqrt{n})$	$O(L+I)$
GHT	$I \times O(\sqrt{n}) + L \times O(\sqrt{n})$	$O(\frac{L}{I})$, {for L > I}
R2D2	$I \times O(\sqrt{n} + \frac{n}{R}) + L \times O(\sqrt{n})$	$O(\frac{I}{R} + \frac{L}{\min(I,R) \times S})$

The total message overhead in R2D2 includes the insertion overhead, where an insertion is composed of a point-to-point route to reach the region and a geocast inside the region. The lookup is anycast to a cached server, so it can be considered as a point-to-point route. In the hotspot message overhead of GHT, we assume lookups are uniformly distributed over keys so that for each key inserted, its home node will have *O(L/I)* lookup. In R2D2, we assume also that keys are uniformly distributed over regions and lookups are uniformly distributed over keys, so that the overhead of a server is the insertions of keys in its region and the lookups (anycasted) are distributed between the *S* servers in the region. The term *min(I,R)* takes care of the case when the number of insertions is less than the number of regions, so that lookups are distributed only over those regions that have insertions. In GHT the refresh overhead per interval is equal to '*the number of keys stored (I) * average perimeter length*'. In R2D2, failure check overhead per interval is at most '*n/R * minimum (I,R)*', since we do

[5] This fix solves the disconnection problem in planarization, which we found to be the major problem that results from location inaccuracy and causes failures during face routing. In this fix, a node does not remove an edge to another node in

not need to geocast in regions that have no keys. At a constant node density, increasing the number of nodes will increase the network size, and by increasing the number of regions in R2D2 with the network size and keeping the region size fixed, n/R remains constant. In other way, the average number of nodes in the region is constant, since the density and region size are constant. In this case, the asymptotic overhead at large number of nodes for R2D2 will be similar to centralized storage and GHT. We can also see that in the simulations, where we increase the number of nodes from 100 to 100000 with different LIRs. The number of insertions is 10. The density is similar to the detailed simulations and the region size in R2D2 is set to have an average of 100 nodes. The results are the average of 10 random simulations with 10 random topologies. In Figure 24, we see flooding has the highest message overhead since each lookup is flooded. Centralized, GHT, and R2D2 have close total overhead. Figure 25 shows the hotspot message overhead, which is computed as the maximum message overhead at a single node. Centralized and flooding have a high hotspot overhead compared to GHT and R2D2. R2D2 has lower hotspot overhead than GHT in these scenarios, because the lookups for a key in R2D2 are distributed over multiple servers in the region, while in GHT the same home node get all lookups for a certain key. As we notice also from the table, the hotspot overhead of R2D2 is low when *I* is small compared to *R*. As *I* increases above *R*, the hotspot overhead will increase. For GHT lower LIR is preferred.

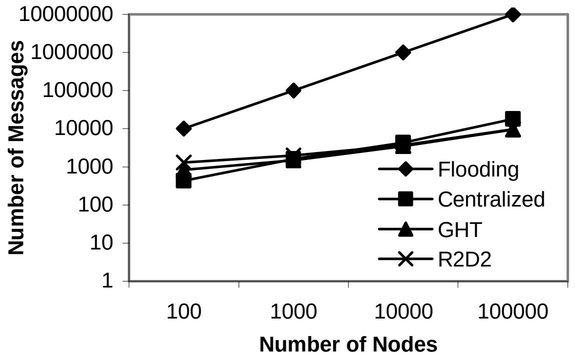


Figure 24: Total message overhead with increasing number of nodes LIR=10

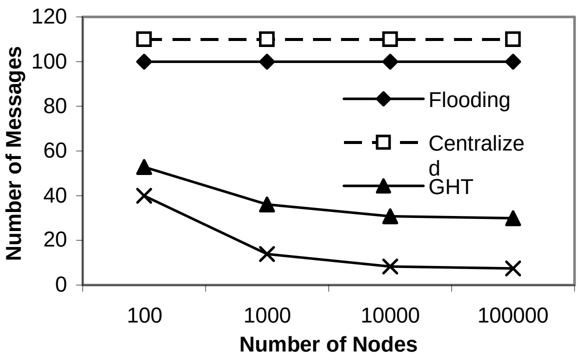


Figure 25: Hotspot message overhead with increasing number of nodes LIR=10

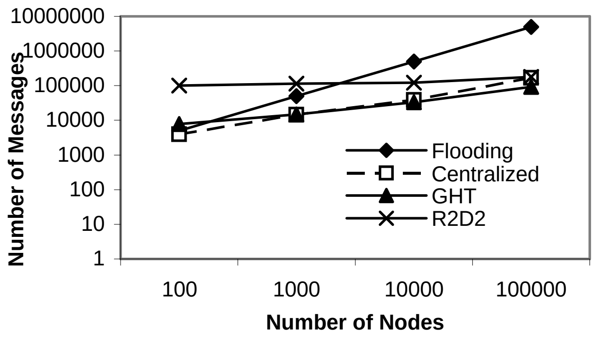


Figure 26: Total message overhead in the event model, LIR=0.05

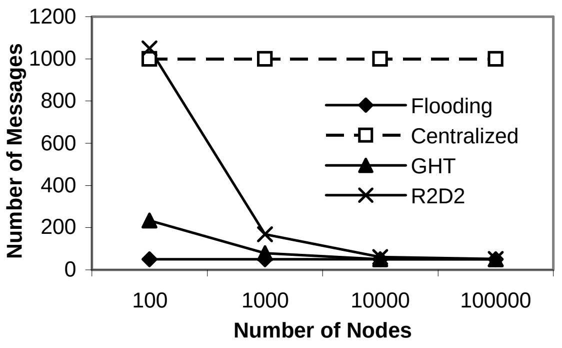


Figure 27: Hotspot message overhead in the event model, LIR=0.05

the planar graph, unless the other node is connected to the same witness. [22]

In Figure 26 and Figure 27, we perform the comparison in the *event model*. We assume 100 event types, 10 detected events per type, and 50 queries. This gives 0.05 LIR. Events are detected at random nodes, while all queries are from a single node representing the network gateway. We assume aggregation is performed at all of them, such that a lookup query returns results as a single reply. We do not consider the structured replication version of GHT. This technique could be used similarly in both GHT and R2D2. The figures show that R2D2 will have higher overhead at low number of nodes, because of the insertion geocast overhead with large number of insertions. As the number of nodes increase, R2D2 will be similar to Centralized and GHT, since the unicast hops will dominate the overhead in large networks, which confirms the asymptotic overhead shown by the table equation.

6 Additional Design Considerations

In this section, we consider additional design issues: the existence of gaps in the region, empty regions and region resizing.

6.1 Region Gaps

As shown in Section 4.3, when a key is inserted in a region, it is stored by all the elected servers in the region. We use geocasting in order to reach all the nodes in the region including the servers. In order to achieve consistency between insertions and lookups, we need a geocasting mechanism that can reach all nodes in the region, otherwise lookups may query servers that are not reached by the insertion geocast. A simple geocasting mechanism is to forward the packet to the region and then flood inside the region boundaries. This mechanism is sufficient at high density networks when all nodes inside the region are connected. But if a gap exists inside the region such that nodes in the region are not connected without going outside the region, this simple mechanism may fail as shown in Figure 28. To solve this problem we proposed a novel geocasting mechanism, Geographic-Forwarding-Perimeter-Geocast (GFPG) [21] that provides guaranteed delivery to all nodes in the region without global flooding or global network information even at low densities and with the existence of region gaps or obstacles. GFPG uses a combination of region flooding and face routing to reach all nodes in the region as shown in Figure 29. In [21], we evaluated this mechanism in detail.

Initially, nodes outside of the geocast region use geographic forwarding to forward the packet toward the region. As the packet enters the region, nodes flood it inside the region. All nodes in the region broadcast the packet to their neighbors and in addition, all nodes on the border of the region send perimeter mode packets to their neighbors that are outside of the region. A node is a region border node if it has neighbors outside of the region. By sending perimeter packets to neighbors outside the region (notice that perimeter mode packets are sent only to neighbors in the planar graph not to all physical neighbors), the faces intersecting the region border are traversed. The node outside the region, receiving the perimeter mode packet, forwards the packet using the right-hand rule to its neighbor in the planar graph and that neighbor forwards it to its neighbor and so on. The packet goes around the face until it enters the region again. The first node inside the region to receive the perimeter packet floods it inside the region or ignores it if that packet was already received and flooded before. Notice that all the region border nodes send the perimeter mode packets to their neighbors outside of the region, the first time they receive the packet, whether they receive it through flooding, face routing, or the initial geographic forwarding. This way if the region consists of separated clusters of nodes, a geocast packet will start at one cluster, perimeter routes will connect these clusters together through nodes outside the region, and each cluster will be flooded as the geocast packet enters it for the first time. This guarantees that all nodes in the region receive the packet, since perimeter packets going out of the region will have to enter the region again from the opposite side of the face and accordingly all faces intersecting the region will be covered. For the detailed explanation and evaluation of GFPG and a comparison to other geocasting mechanisms see [21].

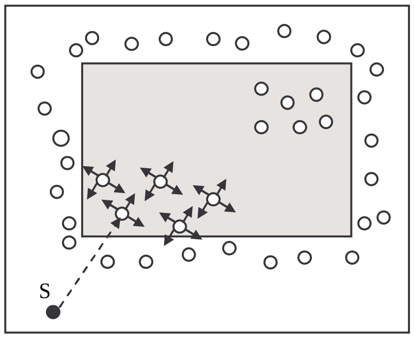


Figure 28: A gap (disconnection) in the geocast region. A packet flooded in the region cannot reach all nodes without going out of the region

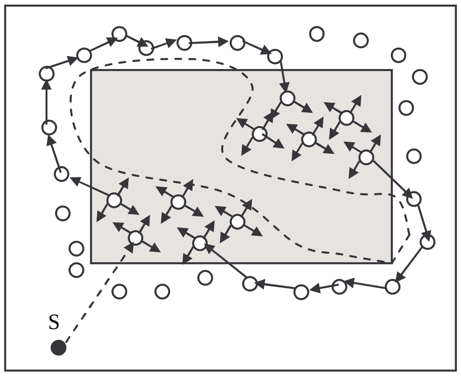


Figure 29: A mix of region flooding and face routing to reach all nodes in the region. Nodes around the gap are part of the same face. For clarity, here we are showing only the perimeter packet sent around the empty face, but notice that all region border nodes will send perimeter packets to their neighbors that are outside of the region

6.2 Empty Regions

One aspect of R2D2 we need to consider is how to deal with empty regions that have no nodes inside. With the relatively large size of regions, this is unlikely to happen when the nodes are uniformly distributed. But for a more general solution that can deal with irregular distributions and obstacles covering a region, we need to consider that. The solution we provide for this problem is to store keys belonging to the empty region in the region of the node closest to that empty region. In both insertions and lookups, a packet forwarded to the empty region will reach the node closest to the region. The node closest to the region will send the packet in perimeter mode, and since there are no nodes closer to the region, it will receive the packet back and it will know that it is the closest node. The node then will insert or lookup the key in its own region similar to regular keys. A server storing keys belonging to other region will have to check periodically that the region is still empty and that no node from other region has become closer to the empty region. If this happens, the key (or a pointer to it) has to move to the other region.

6.3 Dynamic Region Resizing

One option we examined is the dynamic resizing of the regions based on the network conditions. We decided not to perform dynamic resizing due to the complexity it will cause and the extra overhead, while the benefits are not clear. We believe the region size should be set as a multiple of the radio range and since the radio range is fixed, then the region size should be fixed. The tradeoff in region size is between the flexibility to mobility and inaccuracy larger regions provide and the low geocast overhead of smaller regions. Typically, a region size covering of a few radio hops is adequate to overcome the effects of inaccuracy and mobility, while our simulations show that reducing the region size too much (closer to the radio range) does not have significant effect on the geocast overhead, since most nodes become border nodes of the region and affect neighbor regions. Accordingly, increasing or reducing the region size beyond a few hops (e.g. 3-5 hops) of the radio range is not beneficial and it is preferable to keep it fixed as a factor of the radio range.

7 Conclusions

This paper presents the design and evaluation of R2D2, a scalable rendezvous-based architecture for wireless networks. R2D2 facilitates service location and bootstrapping, in addition to data-centric storage, configuration, and task assignment. We evaluated R2D2 using detailed simulations of a realistic wireless environment including the physical details and node dynamics, and we compared its performance and robustness to a rendezvous-point approach. We studied also the scaling properties of R2D2 using high level-simulation and analysis, and compared its scalability to rendezvous-point, flooding, and a centralized approach. The results show that R2D2 is scalable to large number of nodes and is highly efficient, especially in applications with high lookup-to-insertion ratios where it provides 80% overhead reduction over the rendezvous-point approach. It is also robust to node failures and mobility, and it relaxes the requirements for the geographic accuracy of node positions and network boundaries. R2D2 is more robust to location inaccuracy and it requires lower periodic overhead, since no periodic refreshments are needed for each individual key. In addition, R2D2 is more flexible in selecting which nodes to become servers and store the keys. It can choose nodes with certain capabilities from within the region, while with a rendezvous-point approach the home node of a key is determined solely by the geography. Selecting more stable nodes with higher power and memory can have a significant advantage in mesh networks, where nodes have different capabilities and resources.

References

[1] P. Bonnet, J. E. Gehrke, P. Seshadri. “Querying the Physical World”. *IEEE Personal Communications*, October 2000.

[2] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. “Routing with Guaranteed Delivery in Ad Hoc Wireless Networks”. *Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DialM* 1999*)*.

[3] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, J. Jetcheva. “A performance comparison of multi-hop wireless ad hoc network routing protocols”. *ACM MOBICOM* 1998.

[4] N. Bulusu, J. Heidemann, D. Estrin, T. Tran. “Self-configuring Localization Systems: Design and Experimental Evaluation”. *ACM Trans on Embedded Computing Systems (TECS)* 2003.

[5] D. Estrin, M. Handley, A. Helmy, P. Huang, D. Thaler. “A Dynamic Bootstrap Mechanism for Rendezvous-based Multicast Routing”. *IEEE INFOCOM 1999*.

[6] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, S. Shenker. “The Sensor Network as a Database”. *UCLA-TR* Sept 2002.

[7] A. Helmy. “Architectural Framework for Large-Scale Multicast in Mobile Ad Hoc Networks”. *IEEE ICC*, April 2002.

[8] J. Hightower and G. Borriello. “Location Systems for Ubiquitous Computing”. *IEEE Computer*, Aug. 2001.

[9] C. Intanagonwiwat, R. Govindan, D. Estrin. “Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks”. *ACM MOBICOM* 2000.

[10] B. Karp and H.T. Kung. “GPSR: greedy perimeter stateless routing for wireless networks”. *ACM MOBICOM* 2000.

[11] Y. Kim, J. Lee, A. Helmy. “Impact of Location Inconsistencies on Geographic Routing in Wireless Networks”. *ACM MSWIM* 2003.

[12] F. Kuhn, R. Wattenhofer, A. Zollinger. “Worst-Case Optimal and Average-Case Efficient Geometric Ad-Hoc Routing”. *Mobihoc* 2003.

[13] S. Kumar, C. Alaettinoglu, D. Estrin. “Scalable Object-tracking through Unattended Techniques (SCOUT)”. *IEEE ICNP* 2000.

[14] J. Li, J. Jannotti, D. Couto, D. Karger, R. Morris. “A Scalable Location Service for Geographic Ad Hoc Routing (GLS/Grid)”. *ACM Mobicom 2000*.

[15] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. “The Broadcast Storm Problem in a Mobile Ad Hoc Network”. *ACM MOBICOM* 1999.

[16] NS Network Simulator. http://www.isi.edu/nsnam/ns.

[17] N. B. Priyantha, A. Chakraborty, H. Balakrishnan. “The Cricket Location-Support System”. *ACM MOBICOM* 2000.

[18] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, S. Shenker. “GHT: A Geographic Hash Table for Data-Centric Storage”. *Proceedings of the ACM Workshop on Sensor Networks and Applications*, Sept 2002 and *ACM MONET Journal*, 2003.

[19] A. Savvides, C.-C. Han, M. B. Srivastava. “Dynamic Fine-Grain Localization in Ad-Hoc Networks of Sensors”. *ACM MOBICOM* 2001.

[20] Karim Seada, Ahmed Helmy. “*Rendezvous Regions*: A Scalable Architecture for Service Location and Data-Centric Storage in Large-Scale Wireless Networks”. *IEEE/ACM IPDPS 4th International Workshop on Algorithms for Wireless, Mobile, Ad Hoc and Sensor Networks (WMAN), Santa Fe, New Mexico*, April 2004.

[21] K. Seada, A. Helmy. “Efficient Geocasting with Perfect Delivery in Wireless Networks”. *IEEE Wireless Communications and Networking Conference (WCNC), Atlanta, Georgia*, March 2004.

[22] K. Seada, A. Helmy, R. Govindan. “On the Effect of Localization Errors on Geographic Face Routing in Sensor Networks”. *IEEE/ACM 3rd International Symposium on Information Processing in Sensor Networks (IPSN), Berkeley, CA*, April 2004.

[23] Karim Seada, Marco Zuniga, Ahmed Helmy, Bhaskar Krishnamachari. “Energy-Efficient Forwarding Strategies for Geographic Routing in Lossy Wireless Sensor Networks”. *ACM 2nd Conference on Embedded Networked Sensor Systems (SenSys), Baltimore, Maryland*, November 2004.

[24] D. Son, A. Helmy, and B. Krishnamachari. “The Effect of Mobility-induced Location Errors on Geographic Routing in Ad Hoc Networks: Analysis and Improvement using Mobility Prediction”. *IEEE WCNC* 2004.

[25] A. Ward, A. Jones, A. Hopper. “A New Location Technique for the Active Office”. *IEEE Personal Communications*, October 1997.