An Integrated Scheme for Address Assignment and Service Location in Pervasive Environments

M J Kim[1], M Kumar[1] and B A Shirazi[2]

[1] The University of Texas of Arlington
Box 19015, Arlington, TX 76019, USA
{mkim, kumar}@cse.uta.edu
[2] School of EECS, Washington State University
Pullman, WA 99164, USA
shirazi@eecs.wsu.edu

Abstract. We propose an efficient scheme called CoReS (Configuration and Registration Scheme) that integrates address assignment and service location for ad hoc networks prevalent in pervasive computing environments. CoReS exploits node heterogeneity such that more capable and stable nodes serve others. CoReS allocates addresses to individual nodes locally, but employs global allocation states to handle network merge situations. In addition, CoReS exploits the positive features of distributed directory services to perform service location in a centralized manner resulting in minimal communication overheads. We analyze the characteristics of CoReS architecture, evaluate its performance and compare with other schemes. Through the evaluation and comparison, we demonstrate that the integrated CoReS system exhibits high efficiency and cross-layer optimization.

1 Introduction

Pervasive computing is an emerging and very challenging area of research, envisaging numerous computing devices within a small area [1]. Devices possessing specific computing and sensing capabilities may want to communicate with each other to maximize users' quality of life. Such devices may need to form multi-hop mobile networks without prior infrastructures and make use of services available on other nodes. Efficient address assignment and service location are necessary for autonomic network formation and continuous collaboration among devices.

Routing protocols assume that every node in the network has a unique address and is aware of the address of the destination node it is communicating with. Perhaps, the greatest underlying question in networking for pervasive computing does not lie just in providing seamless mobility, but in supporting zero-configuration networking capabilities [2]. Some interesting proposals have come up in the area of automatic address configuration [3][4][5][6][7][8][9]. In addition, service discovery in pervasive computing environments especially for ad hoc networks has started to receive attention [10][11]. Conventionally routing protocols and service discovery protocols are placed in different layers, but integrated or tightly coupled versions of these two pro-

tocols have the ability to enhance efficiency especially in mobile ad hoc networks where each node needs to perform routing and service discovery [12].

In this paper, we propose CoReS (Configuration and Registration Scheme), an integrated protocol for address assignment and service location in pervasive computing environments. CoReS is a distributed, yet hierarchical approach using a small subset of the nodes called CR-nodes that are responsible for assigning addresses to other nodes and registering their services. Relatively stable and capable nodes serve as CR-nodes, thus CoReS exploits node heterogeneity in terms of mobility and capability. A node that has registered its services with a CR-node is referred as a *child node* of the CR-node. The CR-node that has the registration entry of a child node is referred as the *default CR-node* for that child node. CoReS is not limited to any particular MAC protocol or physical network media. Without loss of generality, in this paper, we assume multi-hop wireless ad hoc networks that are common in pervasive environments.

CoReS is adaptable to both IP and flat address formats. For small networks such as Personal Area Networks (PANs) or sensor networks, which do not need any Internet protocols or applications, flat address format is preferable because smaller address size can be used. CoReS assigns unique addresses serially using the *mod* operation, starting from a random reference address. With CoReS, new address allocation is performed locally and network merge situation is managed with global address allocation state. In addition, service location of CoReS operates in a centralized manner, yet flexibly through cooperation of distributed directory services.

The rest of the paper is organized as follows: in Section 2, we discuss related work on address auto-configuration and service discovery. Basic mechanism and protocols of CoReS are presented in Section 3. Section 4 presents protocols handling challenging situation such as network merge. In Section 5, we analyze characteristics of CoReS architecture and evaluate the protocols by comparisons with other schemes. We conclude and discuss future work as possible extensions to CoReS in Section 6.

2 Related work

Address auto-configuration has been a topic of research in various types of networks. Generally, there are two types of auto-configuration protocols - stateless and stateful. Protocols utilizing a stateless approach [3][6] do not maintain allocation states, thus new joining nodes self-configure themselves with randomly selected or h/w based addresses. To assure the uniqueness of the chosen address, Duplicate Address Detection (DAD) is needed causing high overhead/latency and scalability problems due to flooding. In network merge situation, stateless approaches require each node responsible for duplicate addresses since the allocation state is not maintained anywhere.

In stateful approaches, address conflicts are avoided as allocation is managed. Dynamic Host Configuration Protocol (DHCP) is one of the most popular protocols. DHCP maintains allocation states at a central server, thus not applicable in pervasive environments where accessibility to the server is not guaranteed always. More recently, there are interesting proposals employing a stateful approach in Mobile Ad hoc Networks (MANET). One initial effort is MANETConf [4] where every node maintains global allocation states. A new address allocation requires consensus of all

nodes, incurring high communication overhead/delay as well as high memory usage at each node. In network merge situations, detection of address conflicts is easy since global allocation information is maintained at each node. However, nodes with duplicate addresses need to tackle the conflicts themselves.

Another popular stateful approach exploits a buddy system [5][8] where disjointed allocation information is maintained locally at every node. Whenever a new node joins, a preconfigured node just splits its address pool and assigns the divided pool to the new node, ensuring unique address allocation. Zhou et al. proposed a unique address allocation scheme, which uses a special function to assign an address to each node for large scale MANETs [7]. These schemes [5][7][8] enjoy low communication overheads for address allocation procedure without network wide flooding. However, they have limitations in such dynamic situations as network merging and/or abrupt node departure since there is no global view of allocation status.

Service discovery architectures/protocols have been developed and investigated with an emphasis on auto-configurable networks. Sun's Jini [13], Service Location Protocol (SLP) of IETF and Microsoft's Universal Plug and Play (UPnP) are prominent among them. Jini operates with centralized directory servers and UPnP performs in a decentralized manner. SLP works in the two operation modes with and without centralized directory services. All the above protocols are not well suited for pervasive environments due to limitations in supporting node mobility or scalability.

There have been several proposals for service discovery in ad hoc networks recently [10][11][12]. Schemes in [10] and [11] achieve flexible service matching by employing a service hierarchical tree to describe and discover services. However, all nodes are required to maintain the fixed hierarchical tree and employ complex matching mechanisms. In [12], authors proposed the service discovery architecture that is supported by the network layer. Besides the formation of a virtual backbone, the periodic service registration and management of multicast group incur high overhead.

3 CoReS Architecture/Protocols

Each node maintains its degree of mobility (d_m) and resource value (R). d_m is a relative value to indicate dynamism of the node and a function of average speed of the node. For static nodes, $d_m = 0$. R is a quantitative value to indicate the amount of resources the node has. There are three types of resources that make up R - computation, communication and energy. Using d_m and R, each node sets utility value ($\delta = f(d_m, R)$) that is proportional to R and inversely proportional to d_m. The nodes with high δ are good candidates for CR-nodes.

A CR-node predetermines its workload by setting address range (size of address pool) for its child nodes. The address pool size is set to be proportional to utility value (δ) of the CR-node. A CR-node assigns a unique address to each joining node serially ensuring no address duplication unless there are more nodes than total global addresses. After getting an address, a new node registers its service functionalities and its δ with its default CR-node. A CR-node maintains a child table and a CR table. A child table maintains local address allocation states with service information and the utility values of child nodes for directory service. A CR table maintains global address

states with information of all CR-nodes: i) addresses of CR-nodes; and ii) address ranges that CR-nodes manage.

A CR-node creates another CR-node among its child nodes when it has exhausted a predetermined number of addresses, a *threshold* of the given address pool. A child with the highest δ is selected for a new CR-node. A new CR-node gets the CR table from its default CR-node and sets its address range starting from the next available address.

3.1 Address auto-configuration and service registration

When a node N_i wants to join a network, it sends one-hop broadcast join message to its neighbors. If no reply is received before the given *timeout*, N_i tries again with longer *timeout* until the number of retrials exceeds its predefined value ω, or it receives a reply from a neighbor. ω is set to be inversely proportional to δ. After ω number of retrials without receiving any reply, N_i assumes that there is no CR-node. Thus N_i becomes the first CR-node and sets own address pool range starting from a randomly chosen address within the global address space. After that, the first CR-node assigns itself the first address of the own address range and self-registers its service in the child table. In addition, N_i announces its creation to the whole network to let other (possibly) existing nodes register with N_i. When several nodes try to start the network simultaneously, the node with the lowest ω value becomes the first CR-node because it has relatively higher resources and lower mobility.

Let us consider a general situation where some CR-nodes exist. N_i's neighboring node, say N_j upon receiving the join message, sends a probe message to its default CR-node. The default CR-node in turn replies to N_j with remaining address count (A_c). N_j forwards the reply that includes A_c to N_i. Suppose N_j is a CR-node, it directly replies to N_i with its A_c. After the given period of time, N_i selects the neighbor node N_k as a *broker*, where N_k is the neighbor responding with the highest A_c. N_k's default CR-node assigns an address for N_i through the broker N_k as shown in Fig. 1.

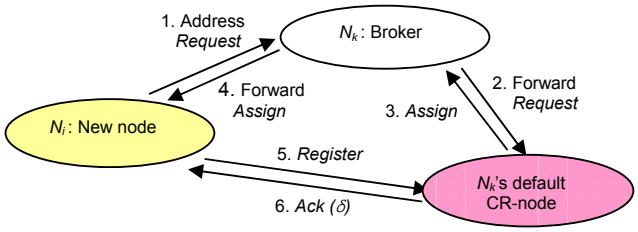

Fig. 1. Node Join Operation: After getting an address, N_i registers its service with the default CR-node that in turn sends an acknowledgement with its δ.

3.3 Service location

When a node needs to avail of a service, it sends a service request message to its default CR-node with the service description. After receiving the service request mes-

sage, the default CR-node looks up its child table to find out matched services. Additionally, it multicasts the service request message to other CR-nodes. Any CR-node that has the information about the requested service in the child table forwards the request message to the child node that provides the requested service.

We call a node requesting a service, *a client* and a node providing the service, *a server*. When a server receives a forwarded service request from its default CR-node, it needs to decide whether to accept the service request according to its current conditions such as workload and available resources. The server may either reject the service request by simply ignoring it, or accept it by replying to the original client along with the detailed service specifications including current resource status and cost. The client selects a server that provides the best preferable service by comparing several replies with details of service specifications.

Late service de-registration may be performed at the CR-node as follows. When a server receives a forwarded service request message, the server needs to send acknowledgement to its default CR-node. If the CR-node does not get acknowledgement within the given number of trials, it suspects that the server (the child) has left and marks the server in the child table. For the marking, one bit of flag field is used in child tables. Similarly, CR tables also have the marking field. When a CR-node has used up all addresses in its pool, it tries to contact the marked child nodes again. If the CR-node still cannot hear from the marked child nodes, it de-registers the child nodes' services and caches their addresses for next address allocation. However, before de-registration of a child, whenever the CR-node realizes the child node is present in the network, the CR-node resets the marking field. We call this *lazy service de-registration* because de-registration does not take place when the server leaves, but is delayed until there is no available address at the default CR-node. The lazy service de-registration makes the protocol more flexible by handling the situation where nodes temporarily drift from the network and come back later.

4 Robustness of CoReS

4.1 Loss of CR-nodes

CR-nodes are not expected to leave often because they are relatively stable and have more resources including energy. When a CR-node A needs to leave, it is responsible to create a substitute CR-node. If A has a capable child node (that has utility value higher than or equal to that of A), A makes the child a substitute. Otherwise A asks a peer CR-node to make a substitute. However, nodes moving out of network or nodes that fail due to some unexpected reason do not have any prior knowledge that they are separating from the network. As a result, a CR-node may leave abruptly without handing over to a substitute, resulting in loss of the CR-node. This case is handled starting from the marking process similar to that in lazy service de-registration. Whenever a node sends a message to a CR-node, the former is supposed to get acknowledgement from the latter. A node suspects loss of a CR-node when the former fails to get acknowledgement from the latter within a given number of retrials with increasing *time-out*. A node suspecting loss of a CR-node, say A, floods this information to the net-

work. (If several nodes suspect the loss simultaneously, the situation is handled with *content flooding* explained in Subsection 4.3.) When other CR-nodes acquire the information about the loss, they set the marking field of A in their CR tables. When a child node of A gets this information, the following algorithm is executed at the child (δ_A is a utility value of A):

```
If (need to increase service utilization) {            1
  If (δ ≥ δA)                                          2
    become a substitute CR-node                        3
  else {                                               4
    relinquish the current address                     5
    get a new address from another CR-node             6
    reregister services with the new default CR-node   7
  }                                                    8
}                                                      9
else {                                                10
  wait until a default CR-node is detected again      11
  if (the detected CR-node is a substitute one)       12
    reregister service with the new default CR-node   13
}                                                     14
```

During the missing period of the default CR-node, a child node sends service requests to another CR-node through neighbors.

After marking, if a CR-node B wants to create a substitute CR-node, first, B needs to wait for a predefined time and check again for presence of the marked CR-node A. If B still cannot hear from A, the former confirms the latter has left without informing, and creates a substitute among its child nodes by assigning the same address range of A. When a child of A becomes a substitute CR-node (line *3* in the above algorithm), the child gets the CR table information from another CR-node through neighbors. The substitution process of the child also includes the waiting and checking procedures. The substitute CR-node announces the substitution by flooding. After receiving the announcement, the remaining child nodes of A reregister with the substitute CR-node (line *13* in the above algorithm) and the other CR-nodes update their CR tables.

Before the substitution, if the marked CR-node appears again, other CR-nodes reset the marking field in their CR tables and the appearing CR-node synchronizes its CR table with the current one. Notice that substitution of a lost CR-node is not required urgently. By the waiting and checking procedures before substitution, temporary CR-node drift can be handled.

4.2 Creation of a CR-node

Creation of a new or substitute CR-node requires consensus among existing CR-nodes in the system. When a CR-node A wants to create another CR-node, first it needs to check a possible ongoing creation event. If A has a pending creation event saved in its cache, it waits for completion of the event before requesting for permission. Otherwise, A directly requests other CR-nodes for permission to create. Another CR-node, say B, upon receipt of the request, sends a negative reply if it has a previous pending creation event. Otherwise, B designates the creation event of A as pending and sends a positive reply.

If *A* receives positive replies from all other CR-nodes other than marked ones, it creates a CR-node. If *A* receives any negative reply, it needs to wait until completion of the ongoing creation event and try again by asking for permission. When *A* does not get a reply from some CR-nodes, *A* starts the marking process by repeating the permission request (explained in Subsection 4.1). After completing the marking process, *A* resumes creation of the CR node. After a new (or substitute) CR-node is created, it informs to all other CR-nodes of its creation to let them update their CR tables and remove the pending creation event.

4.3 Networks merge and partition

When several networks merge, the addresses of all nodes in smaller networks will change by adding offset to larger networks. Let us consider a scenario when three networks *X*, *Y* and *Z* merge. Table 1 shows their address ranges and required changes.

Table 1. Merging of three networks

Network	Original address range (address pool size)	Needed change	Address range after merge
X	67 ~ 159 (93)	+ 0	67 ~ 159
Y	110 ~ 166 (57)	+ 50 (159 – 110 +1)	160 ~ 216
Z	25 ~ 43 (19)	+ 192 (216 -25 +1)	217 ~ 235

Suppose $size(X) \geq size(Y) \geq size(Z)$, where $size(N)$ is the size of address pool assigned in network *N*. *X* does not need to change any address in its network. All addresses in *Y* need to be changed by adding 50 (159 - 110 + 1) to the original ones. Similarly, network *Z* starts its address from 217 by adding 192 (216 – 25 + 1).

CoReS needs network ID to distinguish nodes belonging to different networks [4]. When a CR-node *A* conceives network merge situation, it floods a merge message including its CR table and network ID to the combined network. When another CR-node *B* receives the merge message from *A* for the first time, it compares *A*'s network ID with its own. If they are same, *B* just retransmits the merge message. Otherwise, *B* saves *A*'s CR table along with network ID and retransmits the merge message. In addition, *B* floods another merge message including own CR table as well as network ID. After a fixed time period, each CR-node in the system collects all CR tables corresponding to different networks. When there are overlapped address pools among the CR tables, CR-nodes calculate the needed address changes for each network and merge the CR tables. In addition, CR-nodes let the other nodes in the system know the needed address change by flooding. All flooding in this subsection operates somewhat different from the general flooding protocol such that the messages with the same payload are treated as duplicates regardless of the sources of the messages. In other words, regardless of which and how many nodes initiate the flooding, all nodes send the message once to their neighbors by ignoring any duplicate message of the same content. We refer to this flooding protocol as *content flooding*.

When no address ranges of the merging networks are overlapped, address reconfiguration is delayed until a situation (like another merge) causing overlapped address pools occurs for the case of temporary network merger. When one part of the network separates from the rest, reconfiguration is needed in both partitions. The par-

separates from the rest, reconfiguration is needed in both partitions. The partition that has less than half CR-nodes, changes its network ID. Regardless of getting the new network ID, each network partition needs to handle two side effects as follows: i) missing of some child nodes - use the lazy service de-registration in Subsection 3.2; and ii) loss of CR-nodes - apply the protocol for loss of CR-nodes in Subsection 4.1.

5 Evaluation

5.1 Analysis and comparison of auto-configuration schemes

Auto-configuration protocols using a stateful approach maintain address allocation information in several ways: i) centralized global allocation states; ii) distributed global allocation states; and iii) distributed local allocation states. Central global allocation states are unsuitable for pervasive environments due to uncertain availability of the central server. Distributed global allocation states in [4][9] need to be synchronized among all nodes, causing high overhead/latency. In contrast, distributed local allocation states allow concurrent assignments with low cost as in [5][7][8], but network merge situation cannot be handled efficiently due to lack of global allocation view.

CoReS exploits a stateful approach requiring special nodes (CR-nodes) to maintain distributed allocation states. CR-nodes maintain global allocation states in CR tables as well as local allocation states in child tables. Therefore, address allocation is performed locally and network merge situation is handled with global view of allocation states. In this subsection, we investigate and compare CoReS with other auto-configuration schemes in terms of handling of network merger, address (as resource) management and scalability.

First, CoReS handles network merge situation efficiently due to maintenance of global allocation states. The number of total broadcasts (content flooding) to handle network merger is the number of merging networks + 1 (dissemination of a CR table per network and dissemination of the needed address change). The key advantage is that each node in smaller networks can resume ongoing communications swiftly since the node can apply the same conversion rule to all addresses it maintains including its destination nodes' addresses and cached addresses (routes) in routing tables.

Second, in terms of address resource management, CoReS uses address resource optimally by serial assignment while some schemes [6][7] need significantly larger global address space compared to the number of nodes. Most stateful schemes [4][5][8][9] use address resource efficiently, but they have some difficulty in handling abrupt node departure or temporary node drifts while CoReS handles reclamation of addresses and temporary node drifts in a natural way with the marking process.

Third, with regard to scalability, CoReS is scalable compared to those with distributed global allocation states due to the difference of synchronization requirements: protocols using global distributed allocation states need to synchronize the information among all nodes while CoReS requires only CR-nodes to synchronize their address pools (CR tables) regardless of individual address allocation. Stateful schemes with local allocation states seem efficient for basic address assignment, but are not scalable facing challenging situation such as network merge or abrupt node departure.

5.2 Performance comparison of service discovery protocols

We compare CoReS with well known MANET broadcast based service discovery protocols that employ pull and push models. Pull scheme operates such that whenever a client needs a service, it floods a service request message and any server providing the matched service replies to the client. Push scheme performs such that whenever a client needs a service, it looks up its own cache for information of the needed service. If the client cannot find a match in its cache, it needs to wait for the service advertisement before utilizing the service. A server floods its service advertisement periodically. When a client receives a service advertisement, the client caches the advertisement if it is interested in using the service in future.

We compare the service discovery protocols in terms of operating cost (c_o) measured by the number of exchanged messages for a client to find out all matched services (matched servers) in the system. Let us say n is the number of total nodes and n_s is the number of matched servers in the system. To make things simple, we assume that there is no message loss, thus the system can discover n_s matched services.

In CoReS, a client sends a service request message to its default CR-node that in turn forwards the request to the other CR-nodes in the system. The number of request messages sent (and forwarded) to all CR-nodes is equal to the number of CR-nodes (n_{cr}). After receiving the request messages, CR-nodes that have information of the requested service forward the message to n_s servers (n_s messages). Finally, n_s servers reply to the client directly (another n_s messages). Thus, with CoReS, $c_o = (n_{cr} + 2 \times n_s)$.

When a node floods a message in the network, every node in the network forwards the message once ignoring duplicates. Hence, we assume that n messages are needed for one flooding for simple comparison. With pull scheme, a client floods a service request and the n_s servers reply to the client, which leads to $c_o = (n + n_s)$.

With push model, c_o is dependent on the probability that a client has cached information of the requested service. The probability depends on cache policy and cache size that is another complex issue. Here, we simply refer to the probability that a client has no cached information for the matched service as P_{miss}. Generally, as network size (in terms of n) increases, P_{miss} increases due to limitation of cache size. With P_{miss}, c_o is given by $(n_s \times P_{miss} \times n)$ since $n_s \times P_{miss}$ advertisements are required.

Usually $n_s << n$ and $n_{cr} << n$, thus service discovery of CoReS incurs a lot less operating costs than pull scheme. In turn, operating cost of pull scheme is less than that of push scheme when $n_s \times P_{miss}$ is high, e.g. greater than 1.

6 Conclusion and future work

We propose CoReS, an efficient and integrated protocol to assign addresses for nodes and to support service location in local pervasive environments. CoReS exploits unevenness of nodes to allow more capable and stable nodes to serve others. CoReS employs the hierarchical and distributed architecture such that address assignment is performed locally and network merge situation is handled with global view. Service discovery in CoReS operates efficiently compared to distributed approaches based on broadcast, especially in terms of operation costs.

There are several issues we plan for future work. First, we will utilize CR-nodes for other network functionalities such as routing and security to justify the cost of building and maintaining the CoReS architecture. Second, trust management will be deployed to establish reliable relationship among entities for overall secure interactions. Third, we try to provide locality between child nodes and their default CR-nodes to improve system performance and scalability. Finally, we will incorporate context awareness to CoReS operations for pervasive environments.

Acknowledgements

This material is based on work supported by the National Science Foundation under Grant No. 0129682.

References

1. Saha D and Mukherjee A, "Pervasive Computing: a Paradigm for the 21st Century", IEEE Computer, March 2003, pp 25 – 31
2. P. Mahonen, J. Riihijarvi, M. Petrova, and Z. Shelby, "Hop-by-Hop Toward Future Mobile Broadband IP", IEEE Communications Magazine, Volume 42, Issue 3, March 2004.
3. Zero configuration networking group. http://www.ietf.org/html/charters/zeroconf-charter.html. Cited 29, January 2004
4. S. Nesarig and R. Prakash, "MANETConf: Configuration of Hosts in a Mobile Ad Hoc Network," Proc. IEEE INFOCOM 2002, pp 1059-1068
5. A. P. Tayal and L. M. Patnaik, "An Address Assignment for the Automatic Configuration of Mobile Ad Hoc Networks", Personal and Ubiquitous Computing Volume 8 , Issue 1, 2004
6. C. Perkins, J. Malinen, R. Wakikawa, E. M. Belding-Royer and Y. Sun, "IP Address Autoconfiguration for Ad Hoc Networks", IETF, MANET Working group, November 2001
7. H. Zhou, L.M. Ni and M. W. Mutka, "Prophet Address Allocation for Large Scale MANETs", Proc. IEEE INFOCOM 2003
8. A. Cavalli and J. Orset. "Secure hosts auto-configuration in mobile ad hoc networks", To appear, Ad Hoc Networks Journal by Elsevier Science, 2004
9. M. Mohsin and R. Prakash, "IP Address Assignment in a Mobile Ad Hoc Network," IEEE MILCOM, volume 2, October 2002, pp. 856 – 861
10. D. Chakraborty, et al, "Towards Distributed Service Discovery in Pervasive Computing Environments", IEEE Transactions on Mobile Computing, July. 2004.
11. S. Helal, N. Desai, V. Verma and C. Lee, "Konark--A Service Discovery and Delivery Protocol for Ad-hoc Networks," Proc. the Third IEEE Conference on WCNC, March 2003.
12. Ulas C. Kozat and Leandros Tassiulas, "Network Layer Support for Service Discovery in Mobile Ad Hoc Networks", Proc. IEEE INFOCOM 2003
13. Sun Microsystems, Jini Architecture Specification, June 2003