Assigning Game Server Roles in Mobile Ad-hoc Networks

Oliver Wellnitz Lars Wolf

IBR
Technische Universität Braunschweig
Mühlenpfordtstrasse 23, 38106 Braunschweig, Germany

{wellnitz|wolf}@ibr.cs.tu-bs.de

ABSTRACT

Over the last couple of years, multi-player games have become more and more popular. Additionally, new mobile devices now have sufficient resources to play these multi-player games in mobile and wireless networks. As the classic centralised game server design is unsuited for mobile ad-hoc networks, a group of nodes can take the role of a distributed game server. This group of nodes can provide the necessary redundancy which is needed in the dynamic environment of a mobile ad-hoc network.

In this paper we present a modified dominating set algorithm for server selection which uses local information to determine well-suited nodes from the group of players. Our algorithm uses three phases (discovery, determination and marking) to calculate an initial server set and adjusts to network changes during the game. We implemented our server selection algorithm in NS-2 and evaluated its behaviour in two different realistic scenarios for mobile games (schoolyard and train) as well as in an artificial stress scenario.

1. INTRODUCTION

Multi-player games have become very popular in the last couple of years. However to allow for players to compete against each other, these networked games usually require the users to be on the same LAN or have persistent connection to the Internet for the duration of the game. Additionally, mobile computers have become increasingly popular and are now on par with desktop computers to play today's games. Furthermore, the introduction of hand-held game consoles with wireless capabilities like Sony's Playstation Portable (PSP) or Nintendo's Dual-Screen (DS) underline the development towards playing multi-player games anytime and anywhere.

Mobile ad-hoc networks can support this paradigm by offering wireless communication between players without any need for networking infrastructure. These networks can be created in a spontaneous manner whenever two devices move within communication range of each other. Additionally, mobile ad-hoc networks can support communication to remote devices by using intermediate nodes as relays to forward data to their destination. Thus, multi-hop ad-hoc networks can grow beyond the range of a single wireless transmitter by using a fair cooperation between mobile devices, In [6] we proposed a game architecture for mobile ad-hoc networks.

In general, multi-player games for mobile devices often do not take the different networking environment into account. Wireless networks are more error-prone than wired networks because of radio interferences, distortion, diffraction, reflection and device mobility. Thus, existing approaches like a central game server for a multi-player game are unsuitable for networks with such a dynamic environment. On the other hand a fully distributed approach like peer-to-peer networks does not make efficient use of the available bandwidth, makes cheating in a game easier and does not take differences in terms of resources of the mobile devices into account. By using a distributed server architecture, the game can create the necessary redundancy while keeping network requirements at a minimum. Each game server is responsible for game clients in his vicinity to which it can communicate efficiently. In [6] we introduced the idea of zone servers, which are responsible for a certain area (zone) of the network. Zone servers are not independent devices. The game server software runs on a player's mobile device in addition to the game client. Zone servers communicate in a peer-to-peer fashion with each other. Although, due to the knowledge of the game rules, they are able to make local decisions on their own and delay and aggregate or omit information to other servers. We propose that these game servers should be determined from the group of players and that this selection should be made based on the capabilities and performance of the mobile device as well as its position in the ad-hoc network. Other possible factors may include mobility information, battery level and energy consumption or the player's trustworthiness to act as a game sever.

In this paper we propose an algorithm which determines a number of suitable servers from the group of player nodes. To keep network overhead low, our algorithm uses only local information and information gathered from its direct neighbours. The algorithm is not specific for a certain game genre or multi-player games in general but can also be applied for other kinds of mobile ad-hoc applications which use a distributed server architecture. Furthermore, as a side effect, the results from our algorithm also produce a list of nearby servers for game clients which makes the game server selection process that follows our game server determination

NOSSDAV '06 Newport, Rhode Island, USA

algorithm easier. After a number of game servers have been determined by the algorithm, the game client is responsible of selecting a suitable game server. The game server selection itself is out of scope of this paper.

The remainder of this paper is structured as follows: Section 2 gives some background information and discusses design issues. Section 3 gives an overview of related work. Section 4 explains the three phases of the game server determination algorithm in detail. Section 5 discusses additional server determination during the game. Section 6 shows an evaluation of our algorithm and section 7 discusses some improvements. In section 8, we briefly address the problem of cheating for our algorithm. Finally, section 9 concludes this paper.

2. DESIGN ISSUES

Figure 1 shows an example of a mobile ad-hoc network in which vertices represent mobile nodes and edges denominate the possibility of a direct wireless communication between two nodes. For this paper, we assume that every link between two nodes is bidirectional.

We distinguish two different types of nodes participating in the game: Players actively take part in the game (player nodes) while other nodes cooperatively forward traffic in the network but otherwise do not have knowledge about the game or our server determination algorithm (supporting nodes). Supporting nodes may use the network for other applications and may also rely on player nodes to forward their traffic to its destination. Examples of supporting nodes could be two business people sharing information or a person who needs other nodes to forward Internet traffic to the nearest access point which is not directly reachable from his position.

All nodes using the ad-hoc network have a self-interest that this network works properly and that their traffic is forwarded. Therefore, they also fulfill their responsibility to forward traffic from other nodes. However, this self-interest is not satisfied in the case of being selected as a game server. We think that only players of the game have an interest in keeping the game running while supporting nodes will only forward game traffic because of their own needs from the network. They also may not have the necessary game server software installed on their device or allow that kind of software to drain their sparse device resources. Therefore, we believe that only player nodes should be considered when determining servers for the game. Hence, we define server nodes to be a subset of the group of player nodes.

Because radio communication is a broadcast medium, in principle every data transmission can be received by any node in the sender's neighbourhood. We can use this network characteristic by using broad- or multicast to efficiently communicate information from the server to its players. Hence, we aim at determining servers which are a dominating set of the player nodes in which all servers can communicate directly with their players. During the game, as nodes move around and the network topology changes, we also allow distances of two hops between server and player. If a player moves further away from its server and discovers no new nearby server to which he can initiate a handoff, he initiates a new server determination process in his neighbourhood.

However, a single player could exist isolated in the ad-hoc network, with only supporting nodes nearby and all other players and servers located more than two hops away. This

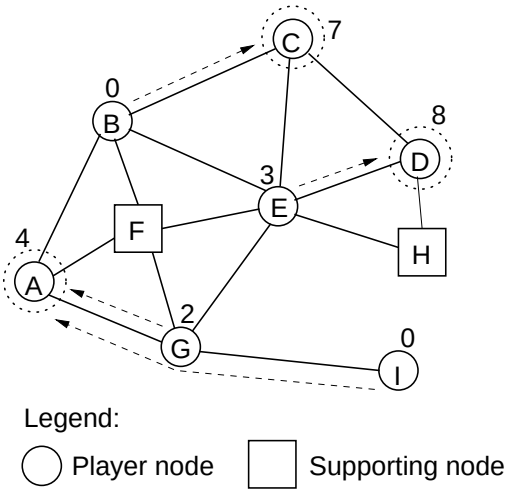

Figure 1: Example of an ad-hoc network

remote player should connect to its nearest server rather than become a game server itself.

As depicted in Figure 1, every player node has a weight associated with itself which represents the node's ability to host a server for the game. This weight is computed locally and may include information about available resources like CPU power, free memory, battery time left or predicted mobility. A node with a weight of zero is not considered as a server for the game.

The degree of a node is defined as the number of edges between this particular node and other nodes in the network. This degree has to be determined in cooperation with a node's neighbours. For example, in Figure 1 node E is participating in the game and has determined it has a weight of 3. It can also communicate directly with six other nodes, so it's degree is six.

3. RELATED WORK

Determining a group of game servers where each server is in the vicinity of a group of players resembles a dominating set from graph theory. A dominating set describes a subset of nodes of a graph and is defined as follows: For all nodes, either the node itself or a direct neighbour is a member of the dominating set. A dominating set that minimises the number of dominating nodes is called minimum dominating set. Determining a minimum dominating set (MDS) is an np-hard problem.

Kuhn and Wattenhofer[5] introduced a distributed algorithm to find a fast and non-trivial approximative solution to the MDS problem in a constant number of rounds. They achieve this goal by using information about the degree of all neighbours up to the distance of two and a suitable weight function.

CEDAR[7] is a distributed routing algorithm which supports quality of service. It relies on an approximated MDS to calculate a core of nodes which are used to forward traffic.

In [8], Wu proposes a routing scheme which uses a dominating set algorithm to determine a backbone in mobile ad-hoc networks. Traffic is routed only by nodes in this backbone while other nodes simply transmit their traffic to the backbone nodes.

All these approaches give a good insight into the distributed calculation of dominating sets. However, they cannot be directly applied onto the problem of game server determination in mobile ad-hoc networks. As mobile ad-hoc networks are heterogeneous consisting of devices with different capabilities, a dominating set algorithm should take the available resources of a device into account. Furthermore, as discussed in the previous section only player nodes should be considered as game servers.

4. THE ALGORITHM

Our algorithm comprises three phases. During the discovery phase nodes exchange information about their resources and capabilities as well as the total number of their neighbours with other nodes in their vicinity. In the determination phase, every node determines itself or a neighbour as a server for the game based on information gathered during the discovery phase. In the final phase all nodes are tagged and the tags are communicated back to the network. We will now take a closer look at each phase.

4.1 The Discovery Phase

In the discovery phase, every node determines its weight and degree. The weight is computed with local information by the game application. Yet, the degree of a node must be determined from other information. Information about neighbouring nodes may already be present from previous communication or from information from the routing protocol. As discussed in [1], low-layer beacons can be used to create a neighbour list without any additional overhead.

Every node periodically broadcasts its ID, weight and degree in status updates packets of 20 bytes size to its neighbours during the whole game session. This information is collected and stored by all nodes in a status table together with a tag which is initially empty. Table 1 shows an example of such a status table.

Node ID	Weight	Degree	Tag
B	0	4	–
C	7	3	–
D	8	2	–
E	3	4	–
G	2	3	–

Table 1: Status table of node E

In Figure 1, the status update of node E is received by all other nodes in the network but nodes A and I which are farther than one hop away. While this broadcast message is ignored by nodes F and H that are not participating in the game, all other neighbours update their status table and insert node E. Because node E does not necessarily know all its neighbours, the initial status update packet may contain a degree of zero. However, as other nodes send their status updates as well, node E will learn about its number of neighbours. As nodes F and H do not participate in the game server determination algorithm, the degree of node E in further status update messages might actually be lower than its number of neighbours.

The discovery phase lasts a predetermined period of time, usually a couple of seconds, after which each nodes moves into the next phase. Additionally, a node can delay the transition to the next phase if it has a neighbour in its status table with a degree of zero. This is usually a node starting late in the server determination process.

4.2 The Determination Phase

During this phase game servers are determined. Every player node checks its status table to see which node has the highest weight among all neighbouring nodes. If this weight is greater than zero, this node becomes game server. Otherwise, no game server is found. If the node itself has the highest weight, it immediately knows about its new role. For all other nodes, a ticket is sent to inform the node that it has been determined as a game server. Nodes with a degree of one are never selected as servers as they have no central role in the network. Any game traffic to and from this node would have to traverse this single link.

In our example in Figure 1, node E selects node D as server, because it has the highest weight of all its neighbours. Node D could also already have determined its server role by itself, as neighbouring node H is not participating in the game and neighbouring node C has a lower weight. In this network nodes A, C, and D are determined as game servers indicated by the dotted circle. The dashed arrows indicate the game servers that could be selected by the players. Besides node I all player nodes have a direct link to a game server.

If two or more nodes in the status table have the same weight, the node with the highest degree becomes server. If weight and degree are the same, the node with the lowest ID is selected.

4.3 The Marking Phase

The marking phase is the final phase of the server determination algorithm. During this phase, every node which has been determined as a game server tags itself as a server. Every node who is a direct neighbour to a server tags itself as a neighbour. Nodes which are not a server or a direct neighbour to a server get an empty tag. Every node broadcasts its tag along with its weight and degree in the previously mentioned regular status updates packets. The tags for all node are also stored in the status table as can be seen in Table 2.

Node ID	Weight	Degree	Tag
D	8	2	Server
C	7	3	Server
E	3	4	Neighbour
G	2	3	Neighbour
B	0	4	Neighbour

Table 2: Final status table of node E

Player nodes which have become game server or have sent a ticket to a neighbouring node can pass the information about the nearest server directly to the game application. Player nodes with no neighbouring server such as node I in Figure 1 can use the information about server neighbours by taking a look at their status table. For example, node I can learn from its status table that a server can be reached through node G and that the server is located at a distance of two hops because node G is tagged as a server neighbour in the status table of node I. Other player nodes with no

servers or server neighbours nearby have to rely on service location mechanisms such as SLP[2] or Konark[3] to select a suitable game server. This is also true for player nodes which have only a supporting node between themselves and a game server because supporting nodes do not take part in the server determination process. These player nodes must also use a suitable service discovery protocol.

5. MOBILITY MANAGEMENT

Ad-hoc networks are mobile and dynamic environments. Unlike the Internet, assigning server roles and selecting a suitable server for a client at startup time is not sufficient for games in an environment that may change constantly. The movement of a player, server or supporting node may effect network performance in terms of latency, jitter or, to a lesser importance for games, throughput.

To deal with such a changing environment and to allow for a good network connection to its server, a game client should be able to initiate a handoff to another, better-suited server at any time. However, as a server handoff requires additional communication, the client should employ a threshold mechanism to avoid constantly switching back and forth. When a server node itself or its group of clients moves further apart, its network performance may become unsuitable for the game. The game application on a server node may also decide to drop its server role after it had no clients for some time. Finally, if a new player joins the game or a powerful player node moves to a new position in the network, it may be useful to use these nodes as additional servers during the game.

To deal with mobility and a changing environment, we use a fourth phase, the game phase, where each node monitors changes during the game. Whenever a node discovers that its distance to the next server has increased to three or more hops, it starts a new server determination process in its area of the network. A node can easily detect that no server is available within two hops when there is no neighbour node in its status table which has the server or server neighbour tag. This node then restarts the algorithm with the determination phase. A state diagram for the complete algorithm is shown in Figure 2. The synchronisation methods between game servers and the setup of new servers were part of a separate research work and are considered out of scope for this paper.

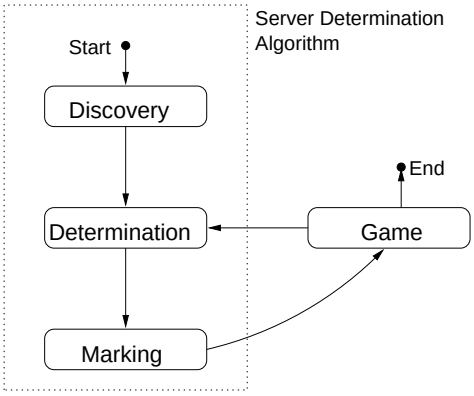

Figure 2: Algorithm State Diagram

Our algorithm to assign game server roles works in a distributed way that we cannot ensure global requirements such as a minimum number of servers. For example, for redundancy reasons, a minimum of two servers is advisable for games in mobile ad-hoc networks. However, there may be situations of high mobility where you want to keep more than two servers all the times. On the other hand, a single game server can also be useful if only one player node has sufficient resources to host this game server as the alternative is not to allow the game to be playable at all. In situations with close proximity of all players and low mobility, a single server may also provide a good and stable service. Such global rules have to be checked and enforced by the game application.

6. EVALUATION

We evaluated our algorithm with NS-2 in three different scenarios for mobile gaming. We used a school yard scenario during a 15 minute break, a train scenario with limited transmission range and an artificial stress test scenario with speeds of up to 10 m/s. In each scenario we used two settings with a total number of 15 and 35 nodes with 10 player nodes and 5 supporting nodes as well as 25 player nodes and 10 supporting nodes respectively. All nodes moved according to the random waypoint model. As determined in [4], we simulated game traffic with constant bitrate traffic pattern with 64 byte packets at 20 pps. Additional background traffic consists of 5 concurrent connections for simulations with 15 nodes and 15 concurrent connections for simulations with 35 nodes (64 byte packets, 10pps). All simulations used the two ray ground radio propagation model and the AODV routing protocol. We also generated random weights for the player nodes ranging from 0 to 99. For each scenario we used ten different weight settings and repeated each simulation with ten different movement patterns. The start time of the algorithm for each node is selected randomly during the first five seconds of each simulation

The school yard scenario used a rectangular area of 400 m × 400 m for the smaller number of nodes and 800 m × 800 m for the larger node number. Because people walk around on the school yard, we employed a maximum speed of 2 m/s. The wireless communication range was set to 250 m. For the train scenario, we used areas of 240 m × 5 m and 450 m × 5 m. All nodes also moved at low speed between zero and 2 m/s. The radio range was restricted to 40 metres which covers less than two coaches. The most significant difference for the train scenario is that the nodes in an ad-hoc network resemble pearls on a string. Node movement was further restricted in the train scenario, so that 50 percent of player nodes did not move at all.

Finally, the stress test was used to analyse our algorithm in a more dynamic environment. In this scenario, all nodes participate in the game and move randomly at speeds of up to 10 m/s.

In the first version of our algorithm, every node sends updated status information to its neighbours every 0.5 seconds, thus creating an overhead of 40 bytes per second and node. As soon as the status table has information about weight and degree of every host, it continues with the determination phase. Tables 3 and 4 show the results of the evaluation of our initial implementation.

A good measure for the quality of the algorithm is the total number of servers. This can be determined by adding the

Scenario	Nodes	# of Servers		Players
		Avg	StdDev	/Server
School	15	3.47	0.90	2.88
Train	15	3.42	0.75	2.92
Stress	10	3.48	0.93	2.88
School	35	8.75	1.31	2.86
Train	35	8.20	1.69	3.05
Stress	25	7.45	1.22	3.36

Table 3: Nodes with a server role at the beginning

Scenario	Nodes	# of Servers		Players
		Avg	StdDev	/Server
School	15	3.49	0.90	2.78
Train	15	4.45	0.81	2.24
Stress	10	6.73	0.91	1.48
School	35	11.22	1.28	2.22
Train	35	11.81	1.30	2.11
Stress	25	12.00	1.99	2.08

Table 4: Total number of game servers

Scenario	Nodes	# of Servers		Players
		Avg	StdDev	/Server
School	15	2.40	0.60	4.17
Train	15	2.83	0.65	3.53
School	35	6.57	0.84	3.81
Train	35	6.45	1.48	3.88

Table 5: Nodes with a server role at the beginning

Scenario	Nodes	# of Servers		Players
		Avg	StdDev	/Server
School	15	2.67	0.59	3.77
Train	15	4.17	0.74	2.40
School	35	10.61	1.13	2.36
Train	35	10.39	1.28	2.19

Table 6: Total number of game servers

number of servers selected at the initial run of the algorithm and the number of servers added during game due to mobility management. The right number of servers for a game depends on the characteristics of the scenario, node movement and the communication requirements of the game. However, to allow for data aggregation, game servers should have at least two or more clients. Additionally, the previously mentioned redundancy requirement of two or more servers should also be fulfilled. Therefore, we aim at a total number of game servers between two and $N/2$ with N being the number of player nodes in the game.

Table 3 only shows the number of servers after the initial run of the algorithm and Table 4 shows the total number of servers at the end of the game. As you can see, the ratio of players to servers at the end of the game is between 1.48 and 2.78. The best result is achieved in the school yard while the result for the stress test is quite bad. The results in the train were predictable because of the special topology and the small wireless range in this scenario.

During the game our algorithm determines new servers only and does not switch off old servers that have no more clients. One reason for that behaviour is that new clients for this server could appear any time as nodes continue to move. Also, the synchronisation time needed to setup a new server should be taken into account when making the decision to degrade a server to a player. This decision should therefore be taken by the game application and is not considered during this evaluation.

As previously mentioned, node movement in the school yard and train scenarios is rather low and should result in no or only a few additional servers. However, as you can see when comparing the larger scenarios in Tables 3 and 4, up to three or more additional servers are determined during the game. The main reason for this is packet loss. The algorithm sends status updates every 0.5 seconds. If two consecutive packets are lost, the node's neighbours assume that this node has become unreachable. Simulations without any game or background traffic show only one additional server in every three simulations. This first version of the algorithm determines close to $N/2$ servers.

7. IMPROVEMENTS

In order to improve the algorithm, we made three changes. The first version of the algorithm sends two status updates per second. A different sending rate for status updates can influence the results of the algorithm. Increasing the rate to five updates per second showed worse results while decreasing it to one status updates every two seconds produced a slightly better result.

Furthermore, player nodes do not start at the same time. When a game is already running, every new node can initiate a new server determination in its area of the network. A more efficient way of dealing with late arriving nodes is to let these nodes directly start in the game phase without going through the server determination algorithm. So a server is immediately selected if one is available in the neighbourhood

The third improvement is delaying the transition from discovery to determination phase. A node must send at least three status packets before moving on to the next phase. This delay improves the quality of the information about the node's neighbourhood. The more complete the view of the neighbourhood, the better results can be achieved by the server determination process. The downside of this change is the increase of the algorithm's run-time until the game phase is reached. For the improved version of the algorithm, the startup delay ranges between 3.3 and 3.5 seconds for all school yard and train scenarios.

While the first two changes have only a negligible impact on the quality of the algorithm, the improvements of the third modification are significant. Tables 5 and 6 show the results of the improved version of our algorithm with all three changes for the school yard and train scenarios.

Further improvements could be achieved by using in-game traffic to detect server failures. At a sending rate of 20 pps between servers and players, we can more accurately decide

whether some packets were lost or if a node has become unreachable.

8. CHEATING

Although the problem of cheating is not the focus of this paper, it requires some serious thoughts. With the distributed algorithm proposed in this paper, any capable mobile device has the possibility of being selected as a server. A player can use this procedure to his advantage and increase the weight of his own device in order to become a server for the game. As a server, he is able to directly impair the performance of players which are connected to his machine by delaying, dropping or altering their packets. Depending on the game logic, it may also be possible to effect other players by taking part in distributed decisions at the server level.

There are different solutions to this problem. When a player detects that a server is being unfair, he can use existing in-game mechanisms to exclude this player from the game or inform other players that this server is untrustworthy. Such voting mechanisms are often present in today's multi-player games. In addition, a player can also initiate a handoff procedure if another server is nearby and thus remove the attacker's ability to directly effect his game. Finally, a reputation mechanism could be established so that only trustworthy players can be selected as servers.

Unlike in the Internet, players in mobile ad-hoc networks are close together so that they can directly interact with each other which makes a social solution to this problem easier.

9. CONCLUSIONS & FUTURE WORK

In this paper we presented a distributed algorithm to determine suitable servers from the group of players in a mobile ad-hoc network. Our algorithm uses local information from its neighbours to determine a nodes role as player or server in the game. It ensures that the distance between player and server stays below two hops where possible. A mobility management mechanism dynamically determines new servers as the topology of the network changes.

We have shown through simulations that our algorithm provides a workable solution to the problem. However, in our simulations packet losses in the network resulted in an unnecessary large number of game servers. The quality of the results could be further improved by finding a good tradeoff between phase transition times, the status update sending rate and the total run-time of the algorithm. Priority queuing or other QoS mechanisms could also be employed to deal with this problem.

Further improvements include matching the status table with the neighbour list from lower layers, such as beacons in IEEE 802.11 networks or information from the routing protocols. An analysis of the importance of the node's available resources and its position in the network for different scenarios could also be used to improve the algorithm.

10. ACKNOWLEDGEMENT

The authors would like to thank Edgar Liptay for his work on the server determination algorithm which eventually lead to this paper.

11. REFERENCES

[1] D. Budke, K. Farkas, O. Wellnitz, B. Plattner, and L. Wolf. Real-Time Multiplayer Game Support Using QoS Mechanisms in Mobile Ad Hoc Networks. In *Proceedings of the 3rd Annual Conference on Wireless On demand Network Systems and Services (WONS 2006)*, Les Ménuires, France, Jan. 2006.

[2] E. Gutman, C. Perkins, J. Veizades, and M. Day. RFC 2608: SLPv2: A service location protocol, June 1999.

[3] S. Helal, N. Desai, V. Verma, and C. Lee. Konark – A Service Discovery and Delivery Protocol for Ad-hoc Networks. In *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC)*, New Orleans, Mar. 2003.

[4] Johannes Faerber. Network Game Traffic Modelling. In *Proceedings of the 1st Workshop on Network and System Support for Games*, pages 53–57, Apr. 2002.

[5] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings of the 22nd ACM International Symposium on the Principles of Distributed Computing*, Boston, Massachusetts, USA, July 2003.

[6] S. M. Riera, O. Wellnitz, and L. Wolf. A zone-based gaming architecture for ad-hoc networks. In *Proceedings of the Workshop on Network and System Support for Games (NetGames2003)*, Redwood City, USA, May 2003.

[7] P. Sinha, R. Sivakumar, and V. Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. In *Proc. IEEE INFOCOM 1999*, pages 202–209, Mar. 1999.

[8] J. Wu. Dominating-set-based routing in ad hoc wireless networks. In *Handbook of wireless networks and mobile computing*, pages 425–450, New York, NY, USA, 2002. John Wiley & Sons, Inc.