Wireless **Ad**-hoc **L**attice computers (WAdL) for Analogical Simulations of Physical Phenomena

Gaurav Mathur
BITS-Pilani, India
(gauravjsr@hotmail.com)

Vishakha Gupta
BITS-Pilani, India
(vishakhasgupta@hotmail.com)

Mentor
Mohan Sharma
Intel-India
(mohan.sharma@intel.com)

Mentor
Rahul Banerjee
BITS-Pilani, India
(rahul@bits-pilani.ac.in)

April 15, 2004

Abstract

We propose an architecture to harness the comparatively low computational power of geographically concentrated mobile devices (such as in a wireless ad hoc network, especially a sensor network) to build a wireless ad hoc lattice computer (WAdL). The existence and usefulness of such an architecture is justified by the phenomenal increase in the number of mobile devices and the rapid increase in their computational capabilities.

The primary contribution of the WAdL design is the ability to maintain, despite the mobility of the participating devices, a virtual lattice where the devices represent lattice points.

WAdL is a cellular automaton-like architecture designed to analogically simulate the unfolding of a physical phenomenon (e.g., fluid flow, system of moving, interacting objects, etc.) in the bounded region of euclidean space represented by the underlying virtual lattice of WAdL.

We present the design of the WAdL architecture, and demonstrate its use with an example application (lift and drag on an airplane wing in flight) implemented on a simulated WAdL environment. We also discuss current issues and future directions of work on the WAdL architecture.

Acknowledgements

There are a lot of people who have helped us during the course of this project, and we would like to take this opportunity to thank them.

We would like to sincerely thank Dr. Anil M. Shende (Associate Professor of Computer Science, Roanoke College, VA, USA), without whose help this project would have been impossible to conceive and implement. It is an under-statement to say that his help, guidance and suggestions have been invaluable to us through the various stages of the project.

Our mentor from Intel, Dr. Mohan Sharma, has been a core part of our project team since he joined us in December, 2003. His questions and suggestions often sent us back to the drawing-board to correct flaws in our ideas and the design. We would like to sincerely thank him for taking time out to help and guide us during the course of this project.

We would also like to thank our BITS mentor, Prof. Rahul Banerjee, for his constant support, and for being our morale booster and critic at the appropriate times.

I (Gaurav) would also like to thank my family members, friends and Jyo for their incessant support during the course of the project. I (Vishakha) would also like to acknowledge the support and encouragement I received from my family, friends and Mehul.

Contents

1 Introduction

Scientific computing largely deals with the prediction of attribute values of objects participating in physical phenomena. For most phenomena, we have analytical models describing the state of the object and its associated attributes. Thus, knowing the analytical model, we can compute the state of the object at any arbitrary time t.

For some physical phenomena, there are no known analytical solutions, and the only apparent method of prediction is the *analogical* simulation of the unfolding of the phenomenon. Thus, to find out the state and the attribute values of the phenomenon at any given time t, it is neccessary to simulate the entire phenomenon from start through time t. (We describe analogical simulations in Section 1.1.)

These analogical simulations can be carried out in a cellular automaton-like architecture — a *lattice computer* — representing the region of euclidean space in which the phenomenon unfolds [26]. Lattice computers are massively parallel machines where the processing elements are arranged in the form of a regular grid, and where the computational demand on each individual processing element is quite low [27, 16, 35]. Each processing element represents a point/region of euclidean space. In analogical simulations on a lattice computer, the motion of an object across euclidean space is carried out as a sequence of steps, uniform in time, where in each step the representation of the object may move from one processing element to a *neighbour*, as defined by the underlying grid of the lattice computer [41].

The proliferation of portable, wireless computing devices (e.g., cell phones, PDAs) promises the availability of a large number of computing devices in a relatively small geographic region. When such devices are equipped with sensors, the resulting wireless sensor networks are typically used for data acquisition; each sensor collects data from its surroundings that can be used for analysis and/or initiating some action. Much research is in progress to develop tiny self-contained computational devices (like the Intel Mote[6]) that could form the building blocks of wireless sensor networks. This promises an increase in the number and density of mobile devices, in the future.

Such an ensemble of wireless devices (computing devices and/or sensors), despite their limited computational capacity, and limited range of communication, provides a rich infrastructure for creating a wireless ad-hoc lattice computer (WAdL). We propose the WAdL architecture as a wireless ad-hoc distributed computing environment for harnessing the collective computing capabilities of the devices for the common cause of scientific computing.

The rest of this report is organised as follows: We discuss analogical simulations in more detail in Section 1.1. Section 2 has a summary of related work in the area of wireless ad-hoc networks. Section 3 describes the architectural frame-work of WAdL, and Section 5 describes the implementation-specific decisions we took to create the *WAdL Simulator*. As a proof of concept, we present results of computing the lift and drag on a simple airplane wing in flight in our simulated WAdL frame-work; Section 4 describes this application. We discuss current issues and future work in Section 6, and lastly, present our conclusion in Section 7.

1.1 Analogical Simulations

A physical phenomenon is a development in a region of euclidean space over a period of time. At each instant in time (in a given time period), the set of objects participating in the phenomenon, together with their attribute values (such as speed, spin, etc.) at that time, completely describes a snapshot in the unfolding of the phenomenon. Most problems in scientific computing are about phenomena whose unfolding involves the motion of participating objects in euclidean space. Solutions to these phenomena usually involve determining (predicting) the attribute values of objects over time. Some phenomena can be solved analytically using closed form functions of time. On the other hand, there are phenomena where the only apparent method for predicting the attribute values of participating objects at any instant in time, is to *simulate* the unfolding of the phenomenon up through that

instant of time [21, 26].

When carried out on a digital computer such simulations, *necessarily*, develop in a discretized representation of a region of euclidean space, and over discrete time units. Moreover, such simulations must use, at any given instant of simulation time, only information available *locally*, at a discrete point in the represented euclidean space, to compute the attribute values of participating objects at the next instant of simulation time. Cellular automaton based machines [35] and lattice computers [16] provide the necessary framework for a discretized representation of euclidean space in which to carry out such simulations. We describe cellular automata in Section 1.2, and also discuss some examples of problem domains where cellular automata are used for problem-modelling.

Several physical phenomena, including spherical wavefront propagation [15] and fluid flow [23, 45], have been successfully simulated on such a framework where the simulation algorithms do *not* use the traditional analytical models for the phenomena.

1.2 Cellular Automata

A *cellular automaton* is a discrete dynamic system. Here space, time, and the states of the system are discrete. The discrete space is represented by a regular spatial lattice. Each point in this spatial lattice, (called a cell) can have any one of a finite number of states. The states of the cells in the lattice are updated according to a local rule. That is, the state of a cell at a given time depends only on its own state one time step previously, and the states of its nearby neighbors at the previous time step. All cells on the lattice are updated synchronously. Thus the state of the entire lattice advances in discrete time steps.

Cellular automata (or CA) have their origin in systems described by John von Neumann and Stanislaw Marcin Ulam in the 1940s. Cellular automata are - by definition - dynamic systems which are discrete in space and time, operate on a uniform, regular lattice - and are characterised by 'local' interactions.

A very important feature of CA is that they provide simple models of complex systems. They exemplify the fact that a collective behavior can emerge out of the sum of many, simply interacting, components. Even if the basic and local interactions are perfectly known, it is possible that the global behavior obeys new laws that are not obviosly extrapolated from the individual properties, as if the whole is more than the sum of all the parts. These properties make CAs a very interesting approach to model physical systems and in particular to simulate complex and non-equilibrium phenomena.

The following sub-sections briefly describe how cellular automata have been used in modelling problems in various domains.

1.2.1 Computational Fluid Dynamics (CFD)

The equations governing fluid flow are : a) the continuity (conservation of mass), b) the Navier-Stokes (conservation of momentum), and c) the energy equations. These equations form a system of coupled non-linear partial differential equations (PDEs). Because of the non-linear terms in these PDEs, analytical methods can yield very few solutions to problems in the fluid flow domain.

In general, closed form analytical solutions are possible only if these PDEs can be made linear, either because non-linear terms naturally drop out (eg., fully developed flows in ducts, etc.) or because nonlinear terms are small and can be neglected (eg., small amplitude sloshing of liquid, etc.). If the non-linearities in the governing PDEs cannot be neglected, which is the situation for most engineering flows, then alternate methods are needed to obtain solutions.

Two approaches exist to solve these problems - a) traditional numerical techniques of CFD, and b) the lattice Boltzmann method. The latter is based on the CA model, and has been widely used to simulate various fluid flows. It is believed to be a strong candidate to replace the traditional numerical CFD techniques.

CFD is used in a lot of domains and industries, some of which are : a) the aircraft industry (eg., wing design, flow of air over the wing, wing's response to turbulence levels), b) weapons design (eg., missiles, torpedos), c) industrial applications (eg., fluid flow in pipes, etc.)

1.2.2 The Game of Life

Conway's Game Of Life (see Figure 1) is the most well known cellular automaton. It has been extensively explored, and a large number of extraordinary patterns have been found. The Game of Life is more of a simulation where you can alter the parameters but you cannot actually alter the outcome directly; that is done by the conditions of the simulation.

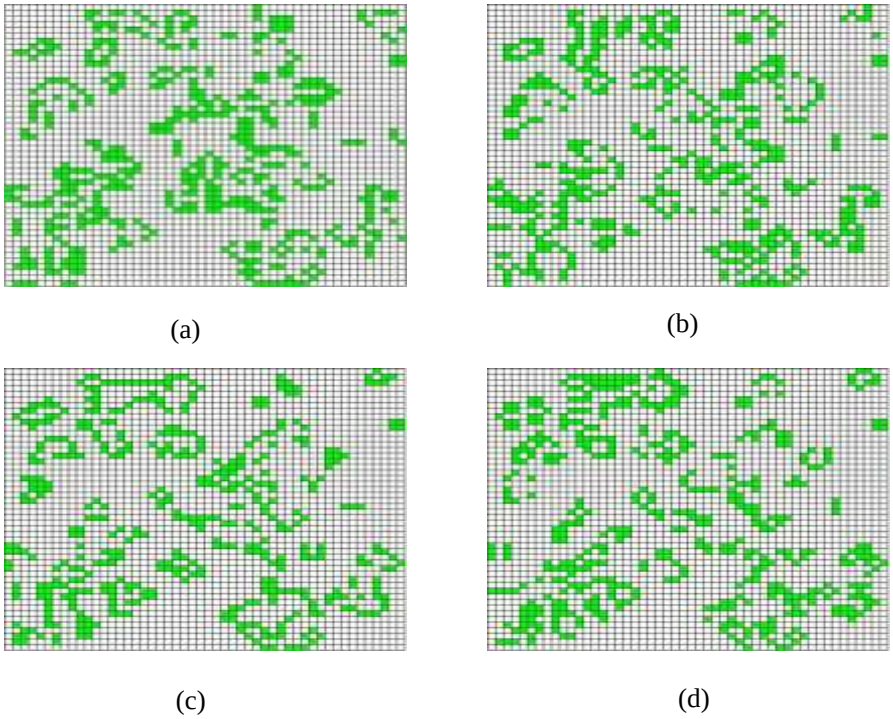

Figure 1: Four consecutive generations (a, b, c and d) in a Game of Life implementation (see [2]). (The green cells are alive, while white cells are dead).

The game is played on a 2-dimensional grid. Each cell can be either 'on' or 'off'. Each cell has eight neighbors, adjacent across the sides and corners of the square. The Game of Life rules can be simply expressed (in terms of the way it affects a cell's behavior from one generation to the next) as follows:

- If a cell is off and has 3 living neighbors (out of 8), it will become alive in the next generation.
- If a cell is on and has 2 or 3 living neighbors, it survives; otherwise, it dies in the next generation.

These specific rules were selected in 1970 by the mathematician J.H. Conway to guarantee that the cellular automaton is on the boundary between unbounded growth and decay into dullness. It was proven that its chaotic behavior is unpredictable and it could be used to build a universal Turing-machine. The contrast between the simplicity of this rule and the complexity of the behavior it produces is a constant source of wonder.

1.2.3 Fractal Drainage Systems

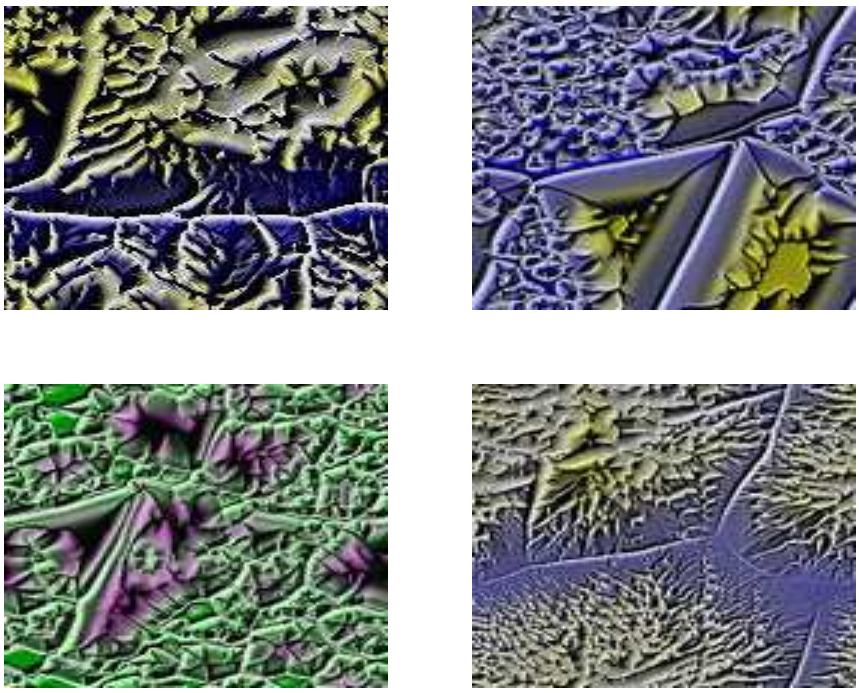

Figure 2: Four example terrains created by CA-based fractal drainage simulations (see [5]).

Cellular Automata can be used to model fractal drainage systems (see Figure 2) - for example, the one caused by erosion due to rain on a landscape. Falling rain turns into little rivulets, which move down local gradients, forming streams and rivers - and corroding the surface across which they flow. The resulting patterns are known as *fractal drainage patterns*.

Cellular automata and particle systems are the most obvious approaches to model fractal drainage systems. One possible approach uses a two-layer cellular automata for this model - one layer represents the density of the liquid at each point, and the other represents the height of the landscape above the ground. The liquid flows across the surface (described by the array of height values) under the influence of gravity.

A partitioning system can be used to ensure that the volume of fluid is conserved. In each time step, the water in each cell is divided into eight approximately-equal portions - allocating one to each neighbour. Fluid flow between that cell and its neighbour is proportional to their height difference, and to the volume of fluid in the higher cell.

The landscape also has a cellular automaton governing its behaviour. The landscape rises up (to replace the material lost to erosion), undergoes simple diffusion processes, and is corroded by fluid flow across the surface. The corrosive effect is proportional to the volume of fluid moving over a given spot.

1.2.4 Biological Simulations of Cell Membranes

The entire cellular system of most of the living organisms has various types of membranes viz. Expanding membranes, Elastic membranes, Semi-permeable membranes, Directional semi-permeable membranes, Amphipathic membranes, etc. These enclose the cells and perform a variety of functions.

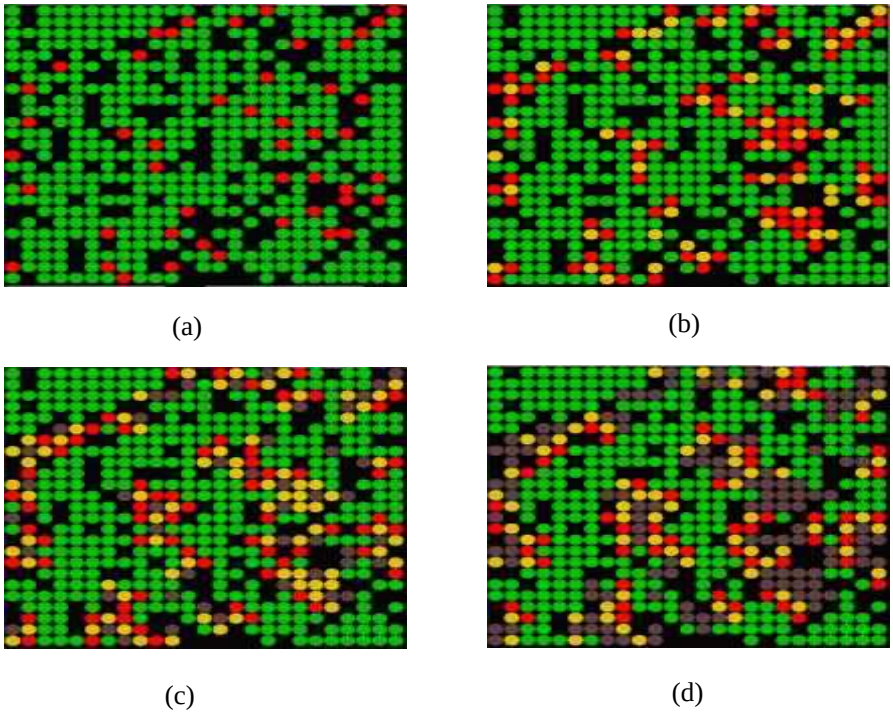

Figure 3: A CA-based simulation to model forest fires (see [4]), a modification of the Game of Life. This figure shows four consecutive generations (a, b, c and d) in a simulation run. (Green circles indicate living trees, red circles are burning trees, yellow circles are burned trees and brown circles are tree stumps)

It would be interesting to attempt to model these directly at a low level - using a simulated liquid and simulated molecules that orient themselves as a result of collisions with water particles. A cellular automaton is ideal for modelling these interactions.

Techniques for modelling membranes will eventually prove useful to those attempting to develop behavioral models of cell walls and even cell membranes in biological sytems.

2 Related Work

There has been intensive work done in the general areas of ad-hoc mobile networks and ad-hoc sensor networks. In such networks, there is a need for routing algorithms to react quickly and adapt effectively, to highly dynamic network conditions. Most routing algorithms follow either the reactive (e.g., indirect proxy-based routing [18]), proactive [13, 42] or on-demand (e.g., AODV [1]) routing strategies. Another solution is offered by geometric/location/position-based routing algorithms such as - Face Routing [33], Adaptive Face Routing [33], Bounded Face Routing [33], Geocasting [31] and Greedy Parameter Stateless Routing [30] - all of which assume location awareness in the mobile device [12, 19, 17], for routing. Other solutions include energy-aware routing [34, 37] that allows a node to efficiently utilize its limited power resources.

Effort has also gone into controlling the network topology so that existing higher-level network protocols, e.g., TCP [10] or UDP [11], can be implemented. This has led to the development of local algorithms for topology control [29], to replace or supplement global algorithms.

Other focus areas include developing efficient media access control protocols to optimize available

bandwidth - such as using orthogonal arrays for scheduling [43]. Work has also gone into optimizing the performance of existing protocols such as TCP, on mobile networks [28, 20]. Improving end-to-end delays is another important concern and has led to the development of techniques such as role-based hierarchical self-organization [32].

There are some innovative approaches that are being developed to better utilize the computational capabilities of even low power and low computational capability devices. Swarm computing [24, 25] is one such approach that involves developing methods for creating, understanding and validating properties of programs that execute on "swarms" of computing devices, analogous to cells in a biological system. Another approach is the integration of mobile devices into computational grids [38, 39, 22]. Both the approaches try to couple the distributed computing power of scattered devices for supporting compute-intensive scientific and commercial applications using traditional solution techniques, i.e., analytical solutions.

3 The WAdL Architecture

Our aim is to use the participating mobile devices (henceforth referred to as *nodes*) in a geographic area to simulate a bounded region of euclidean space, B. The nodes are logically organised so that each node is at a lattice point of a virtual lattice; this virtual lattice represents B.[1] The logical organisation of the nodes is such that all the neighbours, in the virtual lattice, of a node are within communication range of the node. Thus the size of the virtual lattice (or the resolution of the representation of B) is dependent on the density and geographical placement of the nodes, and *not* on the communication range of the nodes.

3.1 Assumptions

We assume that the participating nodes (e.g., tablet PCs, laptops, palmtops, notebooks, PDAs, cell phones, sensors, etc.) have the following capabilities:

1. Computing – each node has some computational capabilities, and offers a frame-work (API) that can utilize these.

2. Location Service – all participating nodes have some form of a location service, such as *Global Positioning System (GPS)* or *base-station based triangulation* that can pinpoint the geographical location of the node with some degree of accuracy.

3. Communication – all nodes have the software and hardware implementation of any single, agreed-upon, short-range wireless communication protocol (e.g., Bluetooth) to enable communication among them. A device is assumed to be able to directly communicate with other devices within antenna-range.

4. Storage – all devices have some storage capabilities. The exact requirement for storage depends primarily on the application utilizing WAdL.

3.2 The Architecture

A WAdL consists of a single immobile node or a base-station (denoted by I) designated as the manager, and a collection of mobile devices as the nodes in the lattice computer. I fixes a lattice, L, with a fixed origin, in its region of influence, and then, each mobile device, p, is mapped to the

[1] A lattice, by definition, is an infinite object. Nonetheless, for expositional convenience, in this paper, by lattice we will mean a finite piece. Thus, it is reasonable for a lattice to represent a bounded region of euclidean space, to count the number of points in a lattice, etc. Also, in this paper, by lattice we mean a finite piece of the infinite lattice whose basis vectors are all equal in magnitude and orthogonal to each other, e.g., a square grid in 2 dimensions.

lattice point $L_p = (x, y)$ closest to it. Thus, all devices in the *Voronoi cell*[2] around a lattice point m are mapped to that lattice point. The dimension of the lattice L is dependent on the scenario, e.g., if I is the manager of a WAdL consisting of devices in a multi-storied building, then L would be 3-dimensional. For example, Figure 4 shows a piece of the region of influence of the manager I, and the lattice L. In this case, $L_p = L_q = (1, 1)$. This mapping is accomplished as follows:

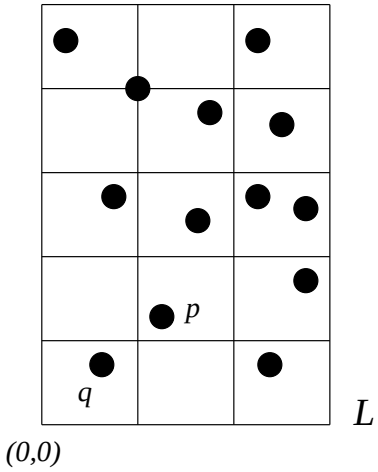

Figure 4: Lattice L and mapping devices to nodes

when a device, p, enters the region of influence of I, the device is sent the dimension of L, the physical coordinates of the origin of L, and the minimal length of L. p then computes L_p using this information and its own physical coordinates. As the device p moves, it re-computes L_p (see also Section 3.2.3 below).

In some cases, the dimension of the euclidean space in which the devices live may be different from the dimension of the lattice needed for the application. For example, in the application we describe in Section 4 below, we assume that the devices (sensors) are on the surface of an airplane wing, i.e., in 2-dimensional space, but our application involves the motion of the wing in 3-dimensional space. In such instances, we first map the devices on a 2-dimensional lattice as described above. Then, the 2-dimensional lattice is logically rearranged as a 3-dimensional lattice. In Figure 5, (a) shows some devices, (b) shows their arrangement as a lattice L with minimal distance 1, and (c) shows the logical lattice V with the effective minimal distance being 3 in the X-Y plane, and 2 in the third dimension.

In WAdL, a node communicates only with its immediate neighbors in L (or V if a logical lattice is being used). The neighbours of a node in the underlying lattice (L or V) are called *virtual* neighbours. The *physical* neighbours of a node (device) are the devices that are in direct communication range. Most virtual neighbors of a node are also the node's physical neighbors (at a distance of one hop), though some of them might be at a distance of more than one hop. This is largely dependent on the lattice mapping algorithm used.

This difference in the distance to virtual neighbors should be transparent to the application. Algorithms presented in [41] help ensure that in a lattice computer, messages propagating from a node at a lattice point m to a node at lattice point n take time proportional to the euclidean distance represented by the two lattice points. Alternatively messages to destinations more than one hop away, can be routed to their destination using one of the wireless ad hoc message routing algorithms (see Section 2).

[2]The *Voronoi cell around a lattice point* m, by definition, is the set of points t in euclidean space such that t is closer to m than to any other lattice point.

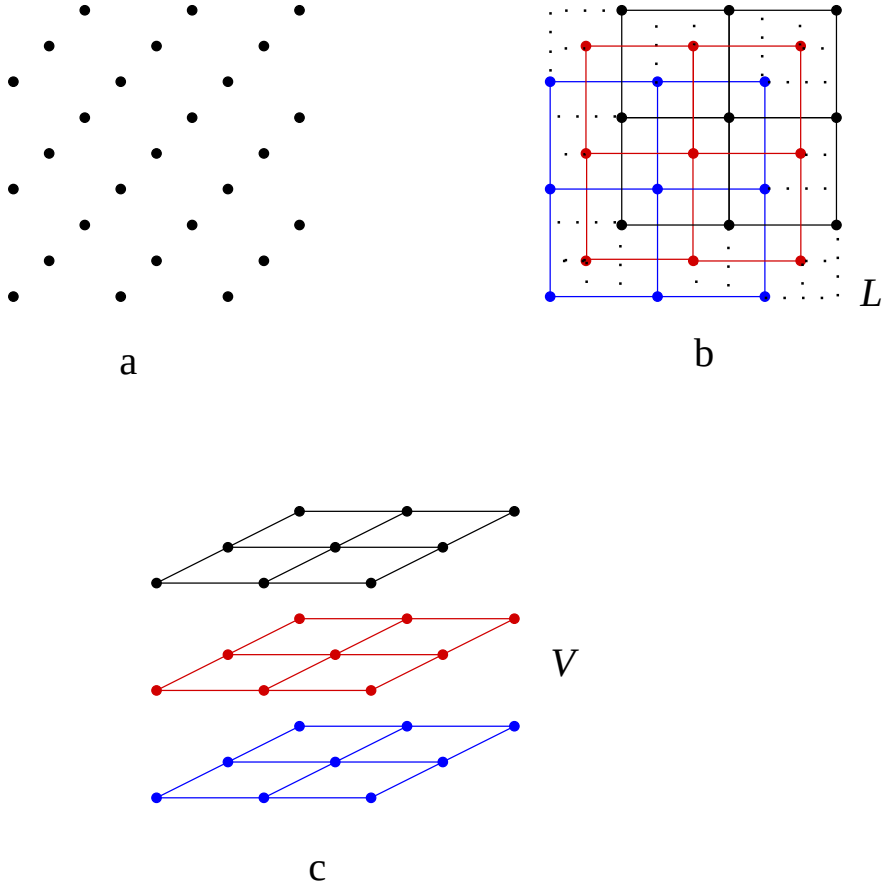

Figure 5: Mapping a 2-dimensional lattice to a 3-dimensional lattice

3.2.1 Lattice granularity

Since the application we present in Section 4 requires the use of a logical lattice V, and the discussion on WAdLs with logical lattice subsumes the discussion on WAdLs without a logical lattice, we will assume for the rest of the paper that we are dealing with a WAdL using a logical lattice.

We define the granularity of the underlying lattice (L and V) as follows:

1. *Fine granularity* – This kind of lattice is formed when the minimal distance in L is small, allowing the number of points in L to be higher. The increase in the number of points of L similarly affects the number of points of V, thus providing a greater euclidean space for the application. Some problems associated with a fine granularity lattice are -

   (a) The motion of the nodes causes them to "hop" from one point to another within L, and hence within V. This motion is rapid (due to the minimal distance in L being small) and causes higher maintenance overheads for WAdL.

   (b) The level of fault tolerance is lower, since each lattice point can have only a few associated nodes (see Section 3.2.2). For fine-granularity lattices, the nodes have to be in close physical proximity for them to be mapped to the same point in the lattice(s). There is a lesser probability of several nodes being in the Voronoi cell around a lattice point, if we assume a uniform distribution of nodes.

2. *Coarse granularity* – This kind of lattice is formed when the minimal distance in L is large. This causes the lattice size of L to be smaller in comparison to fine granularity lattices. Consequently, the size of V and the size of the euclidean space, is also smaller.

   (a) Unlike fine granularity lattices, the movement of nodes within L and V is slower, causing lesser management overheads for WAdL.

   (b) A higher level of fault tolerance is possible in such lattices. This is possible because the physical distance between two adjacent lattice points is greater. There is a higher probability of several nodes being in the Voronoi cell around a lattice point, assuming a uniform distribution of nodes over a given geographical area, and thus being mapped to the same lattice point (see Section 3.2.2).

3.2.2 Fault Tolerance

The first node that arrives at a lattice point m becomes the *primary node* for that point. When more nodes are mapped to m, these nodes join the *backup pool* maintained for m. One or more of the following fault tolerance strategies can be adopted for WAdL, depending upon the application requirements.

1. *The nodes in the backup pool normally remain passive.* Only when the primary node a) fails, or b) moves and hence is mapped to another lattice point, is one of the nodes taken from the backup pool, to replace the primary node. This node becomes the new primary node for that lattice point.

2. *The nodes in the backup pool are active.* The primary node passes on all received and sent parameters to the nodes in the backup pool. Every node in the backup pool performs computations in parallel with (but independent of) the primary node of the lattice point. When the primary node a) fails, or b) moves and hence is mapped to another lattice point, a node from the backup pool assumes the responsibilities of the primary node and resumes computation from where the previous primary node had stopped.

3. *A check-pointing and rollback scheme is adopted.* In this scheme, when a node fails, the application is stopped. When another node becomes available to take its place, the application state is rolled back to the last good checkpoint, from where the WAdL nodes resume computation.

4. *A node's neighbors are used for computation during node failure.* In this scheme, when a node mapped to a lattice point m fails and no nodes are available in the backup pool for m to take its place, one of the neighbors takes up the responsibilities of the node. The neighbor thus ends up performing tasks for two nodes. When a node becomes available at m, the neighbor transfers the state to this new node.

3.2.3 Mobility Scenarios

1. A node n_1 is moving from lattice point m to lattice point n. A node n_2 is available at m. In this case, n_1 transfers its application state to n_2, and then gets mapped to n.

2. A node n_1 exists at lattice point m and another node n_2 gets mapped to m as well. n_2 then joins the pool of backup devices for m. Any one of the fault tolerance strategies described in Section 3.2.2 can be adopted.

3. A node n_1 is moving from the region of lattice point m to the region of lattice point n, and no node is present in the backup pool at m. A *hole* is created at m and an appropriate fault tolerance strategy can be used (see Section 3.2.2).

4. A hole exists at lattice point m and a node arrives to fill the hole. Depending on the fault tolerance strategy used (see Section 3.2.2), the system either performs a rollback to a checkpoint, or a neighboring node transfers the failed node's state to the arriving node.

3.2.4 Executing the WAdL Application

The base-station I of a given WAdL controls the executing WAdL application (a sample application is described in Section 4). The following functions are performed by each I.

1. I is responsible for clock synchronization of all participating WAdL nodes.
2. Parameters (eg., dimensions of P and V, physical distance between neighboring lattice vertices, physical to virtual lattice mapping algorithm, etc.) used by nodes to map themselves to lattice vertices (in both P and V) during execution of a particular application, are also provided by I.
3. The initial parameters for the the application are also provided by I.
4. The application terminates : a) after a defined number of simulation time units, and/or b) when the application tries to move any body/phenomenon to any point outside of the simulated virtual euclidean space.
5. Upon application termination, the application results are transferred to I.

Here, it is worthwhile to note that I's functions are related only to application initialization and termination, and I does not play any role during execution of the application.

3.3 Integrating Multiple WAdLs

We provide an extension of our model to link multiple, geographically distant WAdLs to form a single virtual WAdL. This permits the simulation of a larger region of euclidean space by coalescing the individual euclidean spaces provided by the component WAdLs.

3.3.1 Assumption

1. We assume that each base-station (denoted by I) of participating WAdLs is linked to the other base-stations by a high-speed back-bone network (for example, I could represent the base-station of traditional cell-phone networks). The communication protocol used by the back-bone network is not of consequence to WAdL. For this discussion, we assume that the communication delays and latencies introduced by the back-bone network are negligible.
2. We assume the existence of a fault tolerance strategy that enables the application state to be rolled back to any arbitrary checkpoint out of the last n stored.

3.3.2 Integrating WAdLs

During the setup phase, base stations of various WAdLs are contacted to determine if they would be available for executing the application. The ones that are available (denoted by the set X) respond with the number of WAdL nodes that they could each contribute. Based on the responses received from X, the cumulative euclidean space is computed and then divided into contiguous chunks and distributed to each $I_i \, \epsilon \, X$.

X now performs clock synchronization not only with its associated WAdL nodes, but also with the virtual global clock shared by X. This is repeated periodically to ensure that the clock skew is within acceptable limits.

It should be noted that *only* the base-stations in $I_i \epsilon X$ are aware of the presence of other WAdLs. Each I_i supplies lattice configuration parameters to its associated WAdL nodes in accordance with the euclidean space allocated to I_i. Thus, each WAdL node is only aware of its immediate euclidean space boundaries.

After completion of the setup phase, the application execution is started in X.

The application may reach its termination condition in one or more WAdLs simultaneosly or in close proximity of each other (we denote the set of WAdLs by Y and the respective simulation times by S). When the application terminates, the nodes pass the simulation results to their respective base-stations. Note that collecting the simulation results takes a finite, non-negligible amount of time, in which the simulation in $(X - Y)$ could progress by a few simulation time units.

Each $I_j \epsilon Y$ individually processes the simulation results. Based on the obtained results, and the pre-computed WAdL integration information, I_j computes the next destination for the data in the virtual WAdL. The relevant data ia then passed to the I corresponding to the next computed destination, and the application state is rolled back globally to a time that corresponds to the minimum time in S. The application/simulation is now permitted to continue execution.

This permits the aggregation of the euclidean spaces of multiple WAdLs into a single virtual WAdL, which in turn, provides more virtual simulation space to the executing application.

4 A WAdL Application

We demonstrate the capabilities of WAdL using a simple application that computes the lift and drag on an airplane wing as it flies in virtual euclidean space.

The primary advantage of developing this particular application is the presence of *both* analytical, and analogical models of solving the problem. WAdL computes results using the analogical model, that can be verified using the analytical model.

4.1 The Scenario

We have modelled our application based on the scenario described in the presentation of [44] (at the Workshop on Mobile and Ad-hoc Networks, 2003). We assume that an aircraft is equipped with sensors (comprised of devices with wireless communication and computational capabilities – eg., the Intel mote [6]) embedded in the paint used on the aircraft. Thus, the sensors are scattered all over the aircraft, and in particular on the surface of the aircraft wing, forming a sensor network. The primary purpose of these sensors is to aid in maintenance operations on the ground; we propose to use the same sensor network as a WAdL to analyse the real-time status of the aircraft in flight.

While the aircraft is in flight, the application computes what the ideal lift and drag of the aircraft should be under the aircraft's current external environmental conditions. The obtained ideal values could then be compared against the aircraft's actual lift and drag values to indicate problem points in the aircraft's operation, and possibly take action pro-actively to prevent aircraft malfunction.

Section 4.2 provides a brief background on the theoretical model while Section 4.3 describes the application itself.

4.2 Calculating Lift and Drag of an Aerofoil

See Figure 6 for a visual representation of an aerofoil, and the associated lift and drag forces and their directions. An aerofoil is the 2-D cross-section of a wing (a 3-D structure). It is due to the shape of the aerofoil that lift is generated (due to the pressure difference generated between the upper and lower surfaces of the aerofoil) when the wing is moved within a fluid

Lift is generated in a direction perpendicular to the direction of fluid flow, and given by the following equation:

$$Lift(L) = C_L * 0.5 * density * (velocity)^2 * (wing\ area)$$

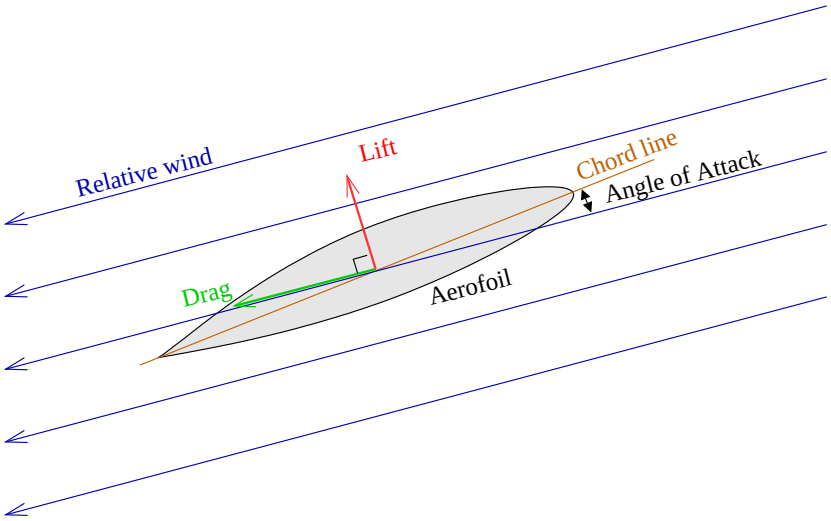

Figure 6: This figure depicts an aerofoil, and the directions of the generated lift and drag.

C_L is termed as the lift coefficient and is normally plotted against the angle-of-attack (or AOA)[3]. The C_L against AOA graph is specific to the wing design.

The lift generated by an aerofoil depends on a lot of factors, of which some are listed below.

- Decrease in air density decreases lift.
- The lift is directly proportional to the square of the airspeed.
- The lift is proportional to the wing area.
- Lift depends on the AOA.

Drag occurs in the same direction as the fluid flow. Drag is given by the following equation:

$$Drag(D) = C_D * 0.5 * density * (velocity)^2 * (wing\ area)$$

The drag coefficient is mainly plotted against the lift coefficient for a particular wing design.

4.3 The Application

This simulation aims at a simple computation of the ideal lift and drag of a wing.

We assume that the graphs - C_L versus AOA, and C_D versus C_L are pre-computed and known for the wing whose flight we wish to simulate. The virtual wing is represented by a set of lattice points in virtual space. Thus, each of these lattice points effectively represents a segment of the wing. The sum of the lift and drag value computed at each of these points provides the net lift and drag experienced by the wing.

The density of the air decreases with increase in altitude. To provide a realistic simulation, we provide the varying density as a parameter to the virtual lattice before start of simulation. We move the virtual wing through our generated euclidean space with constant velocity. As the wing moves, it generates lift, increasing the altitude of the wing. With increase in altitude the density of air reduces, and so should the observed lift on the virtual wing. The wing's virtual "flight" should be visible by observing the state of the WAdL nodes at every simulation time instant.

[3]The angle between the chord-line (an imaginary line between the leading edge and trailing edge) of the aerofoil and the oncoming wind is called the Angle-Of-Attack, or AOA for short (see Figure 6).

5 The Implementation

Our implementation of the WAdL architecture has three distinct parts, and our discussion is divided in a similar manner.

Section 5.3 describes the implementation of the *WAdL Application* (see Section 4) that forms the top-most layer in our architecture. Each participating WAdL node has a single instance of the application executing on it. The application instance interacts with the local *WAdL manager process* (Section 5.1) existing on the node. These manager processes form the middle tier of the architecture, and are responsible for performing all tasks related to the working and maintenance of WAdL. To enable communication, the WAdL manager process on a particular node interacts with another WAdL node's manager process over the bottom-most tier of the architecture, formed by the network.

As part of our implementation, we created a *WAdL Simulator* (Section 5.2) that has the capability of simulating the activities of multiple WAdL nodes, on a single computer. We also created a virtual communication network (Section 5.4) using Network Simulator 2 (ns-2). Finally, in Section 5.5 we discuss and analyze the results obtained from our simulations.

5.1 The WAdL Manager Process

There is a single instance of the WAdL manager process executing on every WAdL node.

The WAdL manager process is responsible for keeping the local clock synchronized, and for maintaining the clock skew (with respect to the global clock) within acceptable limits. It also maintains simulation time for the node.

The WAdL manager process at each node maintains the node's mapping to each of the two lattices – a) the physical lattice and b) the logical/virtual lattice (see Section 3.2).

The simulator uses the following algorithm to map a WAdL node to a vertex in the physical lattice L. The algorithm achieves this by using the location co-ods of every node to map the node to the nearest lattice vertex in L (illustrated in Figure 4).

```
Every node has co-ordinates (x_G, y_G) relative to the lattice origin L, and
co-ordinates (x_L, y_L) in the physical lattice L
dist = absolute distance between adjacent vertices in L

begin procedure map-location-to-physical
1.  x_L = (int) x_g / dist
2.  y_L = (int) y_g / dist
end procedure
```

As the virtual lattice is application-specific in nature, the manager process permits the *application* to define the algorithm for mapping the physical lattice to the virtual lattice.

The current application (see Section 4) requires a uniform 3-dimensional lattice of side s. We use a simple mapping algorithm that places every sth node from L in the same plane, thus creating a 3-dimensional virtual lattice V of "height" s, as depicted in Figure 5. The algorithm is described below.

```
Every node has co-ordinates (x_L, y_L) in the physical lattice L, and
co-ordinates (x_V, y_V, z_V) in the virtual lattice V
count = number of participating WAdL nodes
s = ∛count - side of the virtual 3-dimensional lattice

begin procedure map-physical-to-virtual
```

1. x_v = (int) $\frac{x_L}{s}$
2. y_v = (int) $\frac{y_L}{s}$
3. z_v = $x_L \ mod \ s$

end procedure

The two algorithms mentioned above provide a *local* method of mapping any given WAdL node to the virtual lattice V. Once a node becomes part of the virtual lattice, it can participate in the WAdL computation.

Periodic execution of these algorithms by each node ensures that the mapping of the node in L and V is accurate. The time period of execution is directly proportional to the rate of change of the node's location, and can be determined locally by every node. Thus, to ensure consistency, a node in rapid motion would need to execute these algorithms more frequently than a slow-moving node.

The manager process also handles the communication requirements of the application. It is the responsibility of the manager process to ensure that messages being sent from a node to one or more of its virtual neighbors, reach them all at the same simulation time instant, even though each virtual neighbor could be one or more hops away (see Section 3.2).

5.2 The WAdL Simulator

The WAdL simulator executes on a single machine, and is capable of simulating the behavior and activities of an arbitrary number of WAdL nodes. The number of simulated nodes is limited by the maximum number of virtual network nodes that can be created by ns-2.

To allow the simulation to be executed on a single machine, the simulator serially executes the WAdL nodes (one at a time), and uses the following algorithm to ensure that the state of each node is identical to the state that would arise if the nodes were executed in parallel on separate physical machines.

count = number of participating WAdL nodes

loop until the application finishes execution
1. i = 0
2. loop for every $node_i$ if (i < *count*)
3. receive message(s) from incoming message buffer on $node_i$
4. process the message(s)
5. if message(s) are to be sent, add them to outgoing message buffer on $node_i$
6. i = i + 1
7. end loop
8. j = 0
9. loop for every $node_j$ if (j < *count*)
10. send messages in the outgoing message buffer of $node_j$
11. j = j + 1
12. end loop
13. increment simulation time

end loop

The simulator interfaces with the virtual network (Section 5.4) that provides the communication links between WAdL nodes.

The purpose of this simulator was to demonstrate the design and use of WAdL, and so the simulator currently works on a simplified model of WAdL with ideal node behavior. Distributed clock synchronization (which is currently not required as the simulator executes on a single machine) and fault tolerance strategies have not been implemented yet, though the simulator is capable of

being extended to support much more complex WAdL models.

5.3 The WAdL Application

In this section, we discuss only the implementation details of our sample application, since the goal of this WAdL application and its associated background have already been described in Section 4.

A message is received by the application process at a node, p, from a neighboring node, when a section of the aircraft wing 'arrives' at p's virtual lattice co-ordinates. The information provided in the arriving message is then used by the application process to calculate the speed, lift and drag generated by the wing section. It also computes the direction of motion of the virtual wing and the virtual co-ordinates, C, of the wing's next destination. Accordingly, the application sends a message carrying the current values of the wing parameters, to the destination WAdL node with virtual coordinates C. This message arrives at the destination node two simulation time units later, irrespective of the distance (in hops) of the destination node from the source node. The processing of this message then, proceeds in a similar manner.

5.4 Simulating the Network

We use the Network Simulator 2 (ns-2) tool [7, 8] to create a virtual network for the WAdL simulator. The network simulator creates the required number of WAdL nodes, according to the specified network topology.

The current ns-2 implementation also supports limited node mobility, and we are currently working on extending this to support more complex mobility models. The movement of nodes is handled by ns-2, with ns-2 periodically communicating the new physical co-ordinates of the node to the WAdL Simulator (see Section 5.2). The simulator then takes appropriate action to ensure that this change of location remains transparent to the application.

The packet sent by the application is received first at the ns-2 layer where the packet is examined to determine the nodes in ns-2 that correspond to the packet's source and destination. Based on this information, the packet is then 'sent' over the virtual ns-2 network. Once the packet is received by the destination ns-2 node, it forwards the packet to its corresponding WAdL manager process, which then decodes and processes the data.

5.5 Results

We executed the WAdL simulator on the virtual network formed by ns-2 with the following parameters :

- We set the WAdL node count to 1000 nodes, so that we could construct a regular 3-dimensional virtual lattice with a side of 10 lattice points.
- We set one simulation time unit equal to 5 seconds of physical time. This was done to ensure that the packets travel one hop closer to their intended destinations, with every passing simulation time unit. This also helps in the elimination of non-deterministic delays in packet delivery caused due to communication jitter and network congestion.
- We provided the application with factual data (see [3]), for use in calculating the lift and drag on the wing – a) decreasing air density values for increasing altitude, b) coefficient of lift (C_L) and drag (C_D) values from graphs obtained from an actual wing.
- The ns-2 simulator was configured to provide TCP/IP based communication links between nodes, over IEEE 802.11 MAC protocol.

The obtained simulation results are *identical* to the results obtained from the analytical model of the phenomenon (the application and its underlying theory are discussed in Section 4 above). Figure 7 shows the virtual wing's flight through the virtual lattice, and Figure 8 shows the decrease in lift with the increasing altitude of the wing (in keeping with the constructed model, as described above).

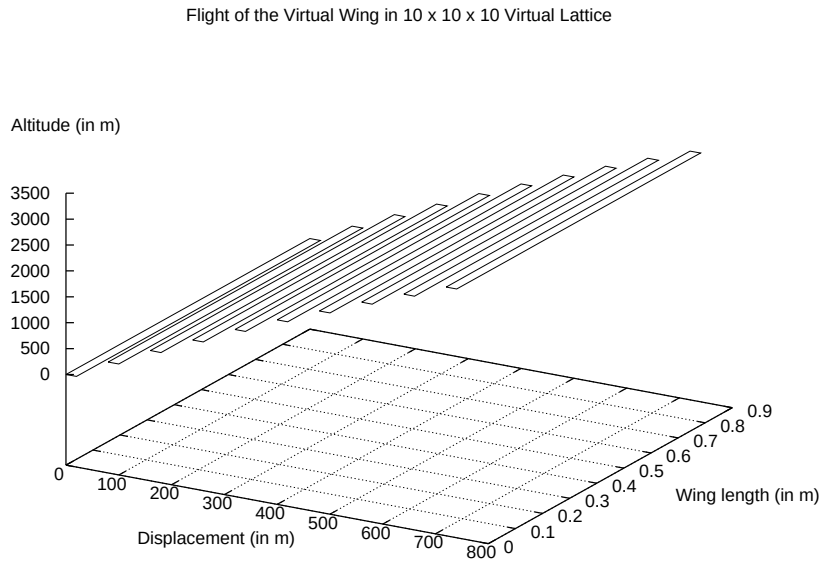

Figure 7: This figure has been created using data generated by the application (see Section 5.3), and depicts the flight of the virtual wing in the virtual, regular, 3-dimensional lattice of side 10 lattice points as generated by the WAdL simulator (see Section 5.2). The size of the wing has been exaggerated to improve figure clarity, while data pertinent to the wing's location in the lattice, has been obtained from the simulation.

We find that the results obtained from WAdL precisely match the results obtained from the analytical model of the wing's flight.

The traces generated by the application are plotted in Figure 9. As the graph depicts, the maximum bandwidth utilization for any active WAdL communication link is extremely low, 64 bytes (this is the size of each communication packet generated by the WAdL nodes). Thus, we conclude that WAdL-related communication causes negligible disruption to ongoing communication in the network.

6 Future Directions

In this paper we have presented an architecture for utilizing mobile computing devices as an ad-hoc lattice computer – a distributed computing environment for carrying out analogical simulations of physical phenomena. We conclude with some of the open problems that arise out of the work we have done so far on WAdL.

1. *Distributed Clock Synchronization* – For exact simulations in practice, WAdL requires that the clocks on all the devices be synchronized. Several schemes have been proposed in the literature

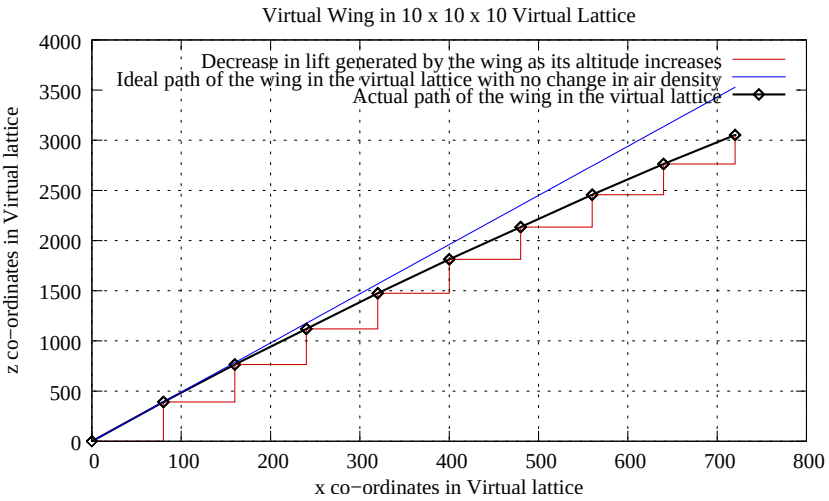

Figure 8: Decrease in lift generated by the virtual wing with increasing altitude. The data for this graph was gathered from the application traces (see Section 5.3).

for achieving this [40, 14, 36]. We are currently exploring the possibility of using the Network Time Protocol (NTP) [9] for this purpose.

2. *Fault Tolerance in WAdL* – The uncertainty caused by the constant motion of the mobile nodes, and their joining and leaving the lattice computer, makes it essential to have a fault tolerance mechanism in WAdL. The fault tolerant system should have the following properties – a) it should support distributed check-pointing and rollbacks, b) it should not be computation-intensive and c) use less bandwidth and infrequently.

   The fault tolerance system can use certain features of WAdL to simplify the implementation – a) it has distributed clock synchronization and b) the entire system uses discrete simulation time.

3. *Factors affecting fault tolerance* – If the density of mobile devices in WAdL falls below a certain threshold, or if the number of holes created in the lattice (see Section 3.2.3) goes above a certain threshold, it is infeasible to simulate the desired lattice, and so the application simulation has to be stopped. Further experimentation with the WAdL simulator will help us determine these thresholds.

4. *Accuracy of the Location Service* – Since devices are mapped to lattice points based on their geographic coordinates as determined by the location service, it is important for the location service to be accurate. Working out the tolerable drift in this accuracy, and correcting for the drift are open problems.

5. *Integrating multiple WAdLs* – Section 3.3 describes an extension of our model that enables the creation of a single virtual WAdL, composed of multiple, geographically remote WAdLs. Such an extension, in turn, raises all of the above issues in a different context, e.g., clock synchronization across WAdLs, handling the mobility of a device from one WAdL to another, etc.

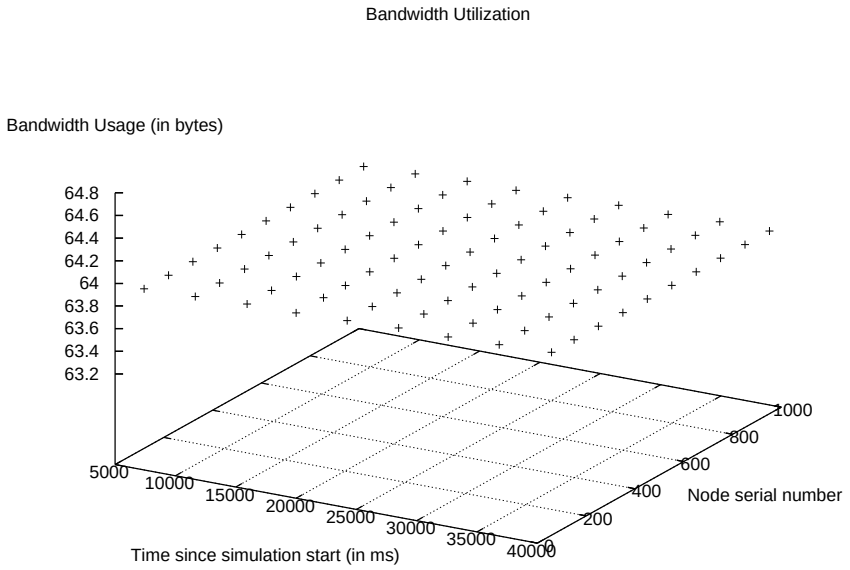

Figure 9: The graph shows the bandwidth utilization of the WAdL simulator nodes versus their serial number and time of sending (since start of simulation). The graph is based on data collected from the simulator (see Section 5.2) and ns-2 (see Section 5.4). Also see Figure 10.

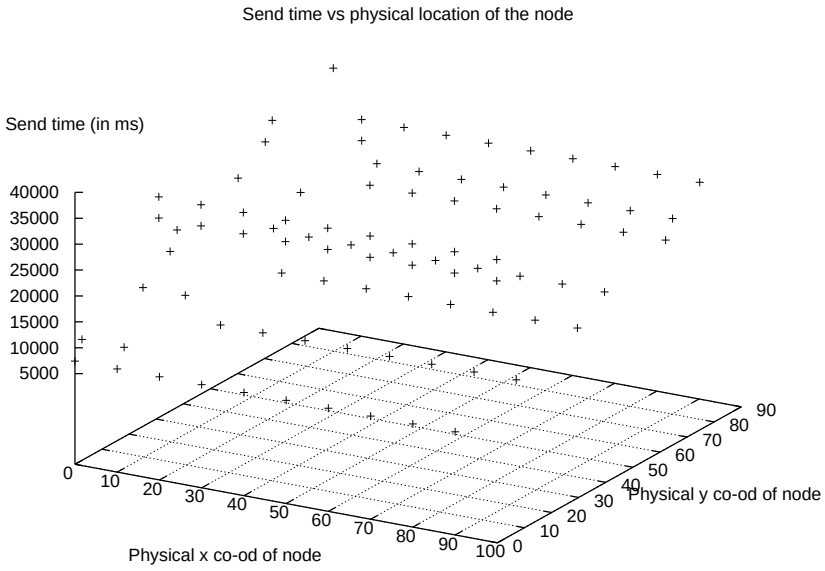

Figure 10: The graph shows the physical location of the node sending data, versus the time of sending the data. The graph is based on data collected from the simulator and ns-2.

7 Conclusion

We propose an architecture to utilize the geographical concentration of mobile devices in a wireless ad hoc network to construct a wireless ad-hoc lattice computer (WAdL), that could be used to perform scientific analogical simulations.

We present one of the scenarios where such a lattice computer could be used effectively. To show the feasibility of this architecture and its applications, we have implemented a *WAdL simulator* and a simple application that utilizes the architecture, to compute the lift and drag generated by an aircraft wing in flight.

Our results show that the network utilization of WAdL is extremely low. This, coupled with the fact that WAdL computation is also infrequent and not computation intensive in nature, enables the WAdL architecture to be deployed in existing mobile ad hoc networks (including mobile sensor networks), without adversely affecting their performance.

References

[1] Ad hoc On-Demand Distance Vector (AODV) Routing. RFC-3561, IETF.

[2] Conway's game of life implementation. http://www.claudeschneider.com/projects/.

[3] Factual data for lift and drag on an aerofoil. http://www.centennialofflight.gov.

[4] Forest fire simulation. http://www.claudeschneider.com/projects/.

[5] Fractal drainage simulations. http://fractaldrainage.com.

[6] Intel mote. http://www.intel.com/research/exploratory/motes.htm.

[7] Network simulator. http://www.isi.edu/nsnam/ns/.

[8] Network simulator (ns) manual. http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf.

[9] Network Time Protocol (version 3). RFC-1305, IETF.

[10] Transmission Control Protocol (TCP). RFC-793, IETF.

[11] User Datagram Protocol (UDP). RFC-768, IETF.

[12] Stavros Antifakos and Bernt Schiele. Beyond position awareness. *Journal of Personal and Ubiquitous Computing*, 6(5), 2002.

[13] Christopher L. Barrett, Stephan J. Eidenbenz, Lukas Kroc, Madhav Marathe, and James P. Smith. Parametric probabilistic sensor network routing. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 122–131, 2003.

[14] Philipp Blum, Lennart Meier, and Lothar Thiele. Improved interval-based clock synchronization in sensor networks. In *Proceedings of Information Processing in Sensor Networks*, 2004.

[15] John Case, Dayanand Rajan, and Anil M. Shende. Spherical wavefront generation in lattice computers. In *Proceedings of the 6th International Conference on Computing and Information*, May 1994.

[16] John Case, Dayanand S. Rajan, and Anil M. Shende. Lattice computers for approximating euclidean space. *Journal of the ACM*, 48(1):110–144, 2001.

[17] Liang Cheng and Ivan Marsic. Piecewise network awareness service for wireless/mobile pervasive computing. *Mobile Networks and Applications*, 7(4):269–278, 2002.

[18] Wook Choi and Sajal K. Das. Design and performance analysis of a proxy-based indirect routing scheme in ad hoc wireless networks. *Mobile Networks and Applications*, 8(5):499–515, 2003.

[19] Nikhil Deshpande and Gaetano Borriello. Location-aware computing: Creating innovative and profitable applications and services. Technical report, Intel Research and Development, August 2002.

[20] Hala Elaarag. Improving TCP performance over mobile networks. *ACM Comput. Surv.*, 34(3):357–374, 2002.

[21] Richard P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6/7), 1982.

[22] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Intl J. Supercomputer Applications*, 15(3), 2001.

[23] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier Stokes equation. *Physical Review Letters*, 56(14):1505–1508, April 1986.

[24] Selvin George, David Evans, and Lance Davidson. A biologically inspired programming model for self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 102–104, 2002.

[25] Selvin George, David Evans, and Steven Marchette. A biological programming model for self-healing. In *First ACM Workshop on Survivable and Self-Regenerative Systems*, October 2003.

[26] D. Greenspan. Deterministic computer physics. *International Journal of Theoretical Physics*, 21(6/7):505–523, 1982.

[27] W. D. Hillis. The connection machine: A computer architecture based on cellular automata. *Physica D*, 10:213–228, 1984.

[28] Gavin Holland and Nitin Vaidya. Analysis of TCP performance over mobile ad hoc networks. *Wireless Networks*, 8(2/3):275–288, 2002.

[29] Lujun Jia, Rajmohan Rajaraman, and Christian Scheideler. On local algorithms for topology control and routing in ad hoc networks. In *Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 220–229, 2003.

[30] Brad Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254, 2000.

[31] Young-Bae Ko and Nitin H. Vaidya. Geocasting in mobile ad hoc networks: Location-based multicast algorithms. In *2nd Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 101–110, 1999.

[32] Manish Kochhal, Loren Schwiebert, and Sandeep Gupta. Role-based hierarchical self organization for wireless ad hoc sensor networks. In *Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*, pages 98–107, 2003.

[33] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Asymptotically optimal geometric mobile ad-hoc routing. In *Proceedings of the 6th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 24–33, 2002.

[34] Qun Li, Javed Aslam, and Daniela Rus. Online power-aware routing in wireless ad-hoc networks. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 97–107, 2001.

[35] N. Margolus. CAM-8: a computer architecture based on cellular automata. In A. Lawniczak and R. Kapral, editors, *Pattern Formation and Lattice-Gas Automata*. 1993.

[36] Lennart Meier, Philipp Blum, and Lothar Thiele. Interval synchronization of drift-constraint clocks in ad-hoc sensor networks. In *Proceedings of the International Symposium on Mobile Ad Hoc Networking and Computing*, 2004.

[37] Anastassios Michail and Anthony Ephremides. Energy-efficient routing for connection-oriented traffic in wireless ad-hoc networks. *Mobile Networks and Applications.*, 8(5):517–533, 2003.

[38] Mauro Migliardi, Muthucumaru Maheswaran, Balasubramaniam Maniymaran, Paul Card, and Farag Azzedin. Mobile interfaces to computational, data, and service grid systems. *SIGMOBILE Mobile Computing and Communications Review*, 6(4):71–73, 2002.

[39] Thomas Phan, Lloyd Huang, and Chris Dulan. Challenge: integrating mobile wireless devices into the computational grid. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 271–278, 2002.

[40] Kay Römer. Time synchronization in ad hoc networks. In *Proceedings of the 2nd ACM international symposium on Mobile ad hoc networking & computing*, pages 173–182, 2001.

[41] Anil M. Shende. *Digital Analog Simulation of Uniform Motion in Representations of Physical N-Space by Lattice-Work MIMD Computer Architectures*. PhD thesis, SUNY, Buffalo, 1991.

[42] L. William Su, Sung-Ju Lee, and Mario Gerla. Mobility prediction and routing in ad hoc wireless networks. *International Journal of Network Management*, 11(1), 2001.

[43] Violet R. Syrotiuk, Charles J. Colbourn, and Alan C.H. Ling. Topology-transparent scheduling for manets using orthogonal arrays. In *Proceedings of the 2003 joint workshop on Foundations of mobile computing*, pages 43–49, 2003.

[44] A. Wadaa, S. Olariu, L. Wilson, K. Jones, and Q. Xu. On training a sensor network. In *Proceedings of the International Parallel and Distributed Processing Symposium*, page 220, 2003.

[45] Jerey Yepez. Lattice-gas dynamics, volume i viscous fluids. Technical Report 1200, Phillips Laboratories, November 1995. Environmental Research Papers.