A Novel Approach for Execution of Distributed Tasks on Mobile Ad Hoc Networks*

P. Basu, W. Ke, and T.D.C. Little
Department of Electrical and Computer Engineering, Boston University
8 Saint Mary's St., Boston, MA 02215

MCL Technical Report No. 08-15-2001

Abstract– A novel distributed approach for executing distributed tasks on mobile ad hoc networks (MANETs) is presented. A distributed application is represented as a complex task comprised of simpler sub-tasks that need to be performed on different categories of computing devices with specialized roles. The dependencies induced by logical patterns of data flow between these specialized devices which are responsible for performing the aforementioned sub-tasks, yield a *task graph* representation for a given application. In this paper, we present a simple and efficient distributed algorithm for dynamic discovery and selection of suitable devices in a MANET from among a number of them providing the same functionality. We refer to this process as *instantiation* which is carried out with respect to the proposed task graph representation of the application. We also present a distributed algorithm for *detecting* disruptions in application execution that can occur due to device mobility. The algorithm then recovers quickly from the situation by re-instantiating affected parts of the task graph, if possible. Finally, we propose metrics for evaluating the performance of these algorithms and report simulation results for a variety of application scenarios.

Proc. IEEE WCNC 2002, Orlando, FL, March 2002. This work was supported by the NSF under grant No. ANI-0073843. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

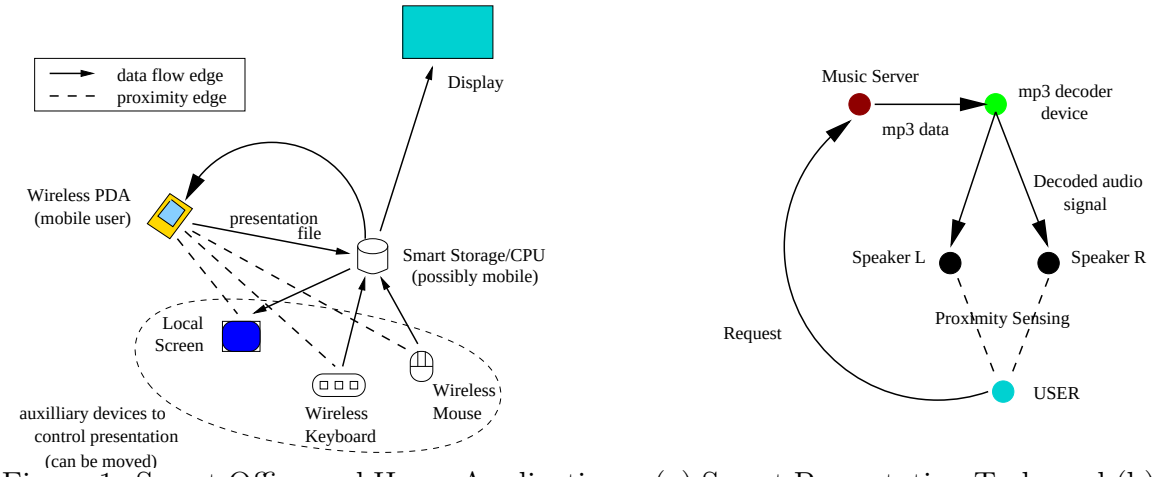


Figure 1: Smart Office and Home Applications: (a) Smart Presentation Task, and (b) Stereo Music Service

1 Introduction

A mobile ad hoc network (MANET) is a rapidly deployable, autonomous system of mobile devices which are connected by wireless links to form an arbitrary graph at any instant of time. With increase in popularity of portable devices and wireless connectivity standards such as IEEE 802.11b and Bluetooth, MANETs are likely to gain popularity in the near future, especially in settings where building a networking infrastructure is expensive, cumbersome, and/or impossible. When a large number of computing devices become equipped with wireless connectivity, they can offer services to other devices for performing several tasks. In such a situation, since the service providing devices may themselves be mobile, a user cannot rely on one particular device for a certain service since its reachability/availability is not guaranteed. Instead, a user must be prepared to access the required service from any of the several devices in the MANET providing similar services, if possible.

Recently, lower layer issues such as channel access and routing have received a lot of attention from the MANET research community [5, 12]. However, higher layer application design issues have received little attention so far. We bridge this gap with a novel scheme for modeling and executing distributed applications on MANETs that rely on services offered by devices scattered across the network.

We introduce the abstraction of a *Task-Graph* (TG) for representing higher-level tasks that a user may want to perform, in terms of smaller sub-tasks. It is a graph composed of *nodes* and *edges*, where the nodes represent *categories* of devices/services needed for processing data related to the task while the edges represent necessary associations between different *nodes* for performing the task. Fig. 1 shows two examples of task graphs in the pervasive computing domain. Fig. 1(a) shows the TG of a smart presentation application in which a user, carrying only a PDA with wireless connectivity, automatically discovers and instantiates the most suitable devices in a room needed to make a full presentation. Fig. 1(b) shows the TG of a music service application, in which the application tracks the physical location of the user and instantiates the appropriate speaker devices that will play the music the user has selected. The proposed TG modeling approach can find applications in sensor networks, disaster relief, and distributed mobile computing/gaming too.

Before task execution, specific devices need to be selected (in other words, *instantiated*)

at runtime, and then made to communicate with one another according to the structure of the TG. When a participating device becomes unavailable due to network failures, a new substitute device with similar capabilities has to be appointed to continue the task. In this paper, we present efficient distributed algorithms for achieving the above goals.

We measure the success of our instantiation algorithm by some metrics such as average time for instantiation, quality of the instantiation (in this paper, the technical term used for this is *dilation*), frequency of disruption of tasks due to mobility or route failures, and average effective throughput that is achieved after application data transmission begins.

The TG abstraction of a distributed task is potentially advantageous in many ways. It is inherently distributed, as most pervasive applications and futuristic services are likely to be. It also offers hierarchical composability, as collections of devices can be logically grouped together to constitute a single node in a TG.

The concept of a task graph was originally used in parallel computing and scheduling literature for representing tasks that can be split *temporally* into sub-tasks and then allocated to different homogeneous processors connected by a fixed high-performance interconnect for reducing the total completion time [6, 13]. Our notion of a task graph is different from this classical one. We are not necessarily concerned with tasks that are distributable among multiple homogeneous processors for speed-up. Rather, most tasks that we are concerned with in this work involve several specialized heterogeneous devices that communicate with each other, and there is no notion of minimizing the total completion time. To the best of our knowledge, this is the first attempt toward modeling distributed applications on a MANET using task graphs.

The rest of the paper is organized as follows: Section 2 introduces the TG modeling framework. Section 3 presents a distributed algorithm for TG instantiation in a network. Section 4 presents simulation results of the proposed algorithm. Section 5 briefly describes the related work in the literature. Section 6 concludes the paper.

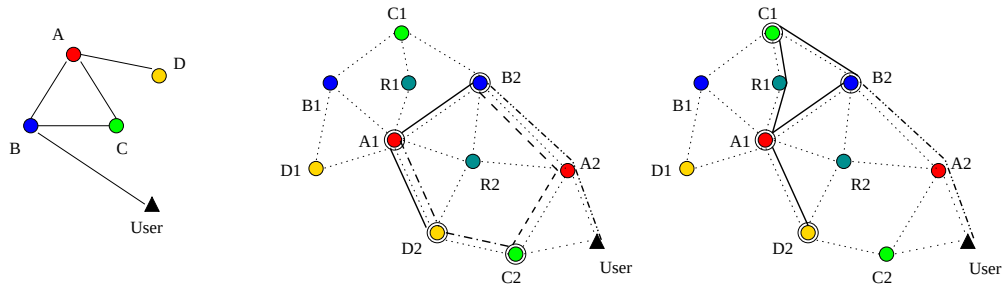
2 A Task Graph Based Modeling Framework

In this section, we first lay the foundations of a task graph based modeling framework and then introduce the task embedding problem.

2.1 Terms and Definitions

A *device* in our context is a physical entity that is equipped with an *embedded* processor for performing computations, sensors and actuators for interacting with its physical environment, a wireless communication port for communicating with other devices, and/or a user interface. If a device primarily performs one specific function, it is called a *specialized device*, otherwise, it is called a *multipurpose device*. A device can have both static and dynamic attributes reflecting its capabilities. In this paper, we only consider specialized devices with their principal attribute, i.e., their main function. Multi-attribute extensions are possible to our work on the lines of [1].

A *service* is a functionality provided by a device or a collection of cooperating devices. There may be multiple devices in the MANET providing the same service.



(a) Task Graph (b) Embedding #1 (c) Embedding #2

Figure 2: Example of Task Graph Embedding

A *node* is an abstract representation of a device or a collection of devices characterized by a minimal set of attributes that can offer a particular service. A node is *simple* when it represents a single physical device. It is *complex* when it represents multiple simple nodes. We consider simple nodes only in this paper. We refer to the principal attribute of a node or a device as its *type* or *category*.

An *edge* is a necessary association between two *nodes* with certain attributes that must be satisfied for a task to execute. Examples of edge attributes include causal ordering, weight, required data rate between nodes, allowable bit error rate, and physical proximity.

A *task* can be described as work executed by a node with a certain expected outcome. The work done by a constituent of a complex node is considered a *sub-task* of the bigger task. An *atomic* task is an indivisible unit of work which is executed by a simple node[1].

A *task graph* is a graph $TG = (V_T, E_T)$ where V_T is the set of *nodes* that need to participate in the task, and E_T is the set of *edges* denoting the data flow between participating nodes.

2.2 Embedding a Task Graph onto a MANET

Embedding a task graph $TG = (V_T, E_T)$ onto a MANET $G = (V_G, E_G)$ involves finding a pair of mappings (φ, ψ) such that $\varphi : V_T \rightarrow V_G$ and $\psi : E_T \rightarrow P_G$, where the *category* of $v \in V_T$ is the same as that of $\varphi(v)$ and P_G is the set of all source-destination paths in G. Fig. 2(a) depicts a hypothetical task graph. Figures 2(b-c) show a sample network topology with two possible embeddings.

The entire process of device discovery, selection of a device from multiple instances of devices in the same category, and the assignment of a physical *device* to a logical *node* in the task graph is referred to as *instantiation*. The collective process is called *embedding*.

2.3 Metrics for Performance Evaluation

In general, the embedding process maps edges in TG to *paths* in G. *Average Dilation* of an embedding is the average length of such paths taken over all edges in TG. Mathematically, if $\|a, b\|_G$ denotes the length of a shortest path between nodes a and b in G, dilation is given

[1]Atomicity is related to the core capability of a device, and is partially constrained by subjective design choices.

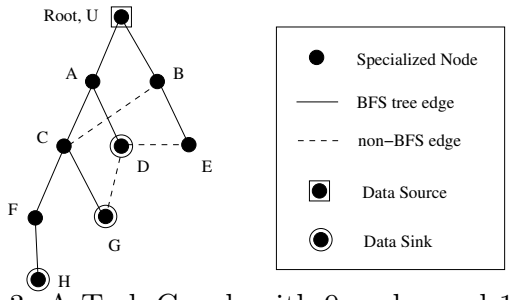


Figure 3: A Task Graph with 9 nodes and 11 edges

by:

$$D_{avg} = \frac{1}{|E_T|} \sum_{(x,y)\in E_T} \|\varphi(x), \varphi(y)\|_G$$

Dilation is an important metric since it impacts the throughput between instantiated devices. An embedding with high dilation signifies long paths between directly communicating devices, which is undesirable since TCP throughput drops significantly with increase in hop distance [11]. In contrast, an embedding with low dilation results in better task throughput.

Speed of instantiation is another useful metric which measures the time taken to instantiate a node in TG on G. When an embedding is disrupted owing to network failures, it also measures the time taken to find a new replacement device.

A useful metric for measuring the resilience of the protocols to failures is *Average Effective Throughput*, ($AvgEffT$), which is the average number of application data units (ADUs) actually received at instantiated data sinks divided by the number of them that were supposed to be received at the intended targets in an ideal situation[2].

3 A Distributed Embedding Algorithm

In this section, we present a distributed approach for solving the task graph embedding problem in a MANET with an objective of minimizing D_{avg}. In this work, we assume that each heterogeneous device can provide a single type of service, and that all nodes in the network are *simple*. We assume the presence of a MANET routing protocol (DSR, in our case) and a reliable transport protocol (TCP).

All devices in the network execute copies of the same algorithm except the user node, U which executes a different algorithm since it acts as a *state synchronizer* (or coordinator[3]) in the initial phases of the embedding process.

The embedding process begins at U with a distributed search which proceeds through the MANET G hand-in-hand with a *breadth-first search* (BFS) through TG. Fig. 3 depicts a task graph with its BFS and non-BFS edges. We call the spanning tree on TG induced by BFS and rooted at U, a BFS tree ($BFST_{TG}$) of TG. We propose a *greedy* solution to keep the dilation of the embedding low: the algorithm begins from U by progressively mapping

[2]If a relaying node in the path from source to sink gets uninstantiated, effective throughput will be affected since certain data flows will not reach the sinks.

[3]In our opinion, the user devices are best suited for acting as coordinators since they usually originate the application data flows, and even under mobility, always remain near the user.

the nodes of $BFST_{TG}$ to nearest devices and the edges to shortest paths in G. Instantiation of any pair of nodes $x, y \in V_T$ cannot affect each other if x is not a parent of y in $BFST_{TG}$, or vice-versa. Hence, the search can proceed in a distributed manner along the branches of $BFST_{TG}$.

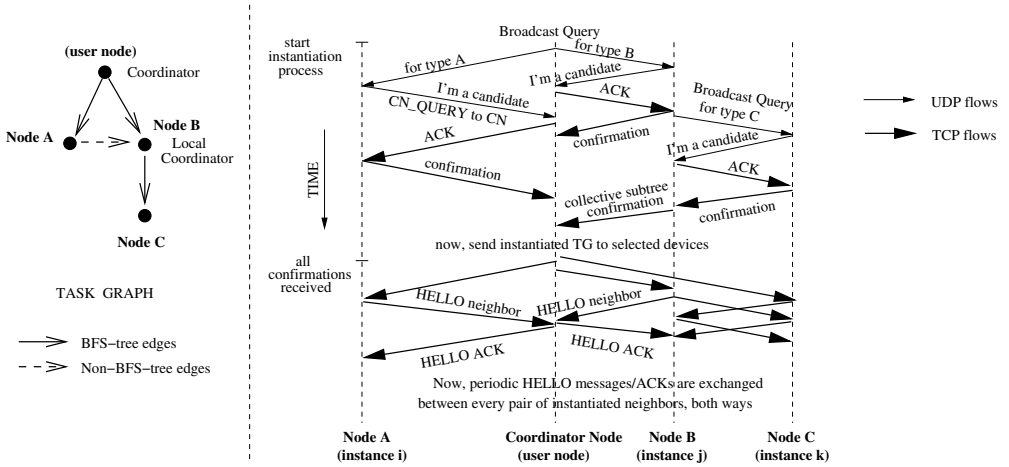


Figure 4: Dynamics of the Embedding Scheme

The salient steps of the algorithm have been illustrated in Fig. 4 for a smaller TG by means of a time-based message diagram, and are summarized below. Details of the protocol (finite state machine descriptions, in particular) have been omitted due to paucity of space and can be found in [3].

1. U broadcasts search queries for each neighbor category in TG[4] (A and B).

2. Available instances of each queried node reply to U. Candidate devices that reply first (A_i, B_j) become the chosen *instances* at U.

3. U sends an ACK to these selected devices which send back confirmations.

4. If there are any uninstantiated nodes rooted at any instance in TG (such as C below B_j), then it broadcasts search query packets for all those node categories and the instantiation proceeds further[5].

5. When confirmations from all nodes reach U, the data transmission can begin[6].

The task graph itself is sent as control data during the instantiation process. After the selection of a device, control packets and application data are transmitted using TCP since packet losses due to route errors are very common in MANETs.

[4]The broadcast is controlled by sending the query packet to all one-hop neighbors which examine its contents and decide whether to rebroadcast it. A time-to-live (TTL) field in the packet also prevents it from causing a storm.

[5]B_j here acts as the local coordinator responsible for instantiation of nodes rooted below it.

[6]In an ideal situation, all data originating at the source should reach the instances of the sink nodes in TG (A_i and C_k in the example in Fig. 4) after having been massaged and relayed by the intermediate devices (B_j).

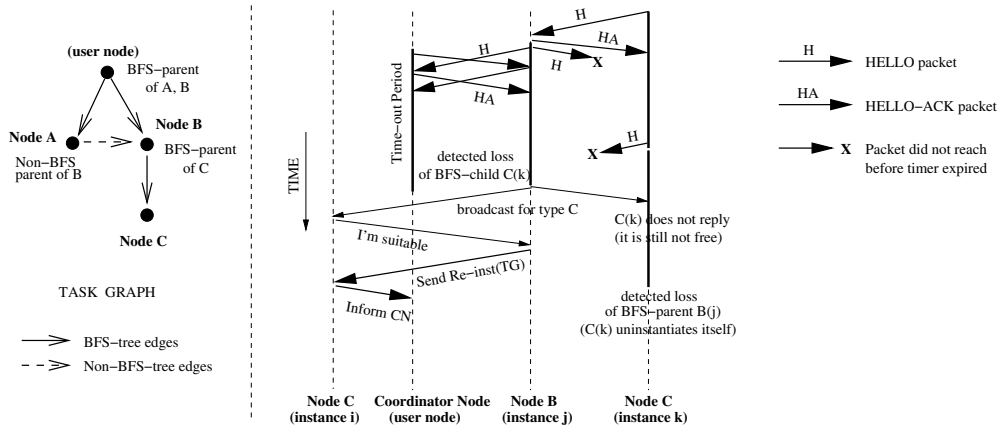


Figure 5: Handling Task Disruptions by Re-instantiation of Devices

3.1 Impact of Mobility of Devices

If the devices in the network are highly mobile during the lifetime of the applications running on the network, the network topology and previously established connections may change, and this may disrupt the applications. Therefore, it is not sufficient to permanently appoint specific devices for executing the application – continual monitoring must be performed to detect disruptions, and replacement devices must be selected for resuming the application.

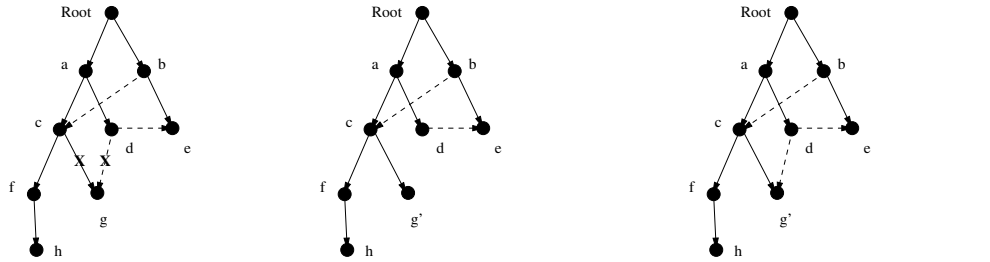


Figure 6: Bookkeeping after Re-instantiation

Detection of Disconnections Mobility of devices may cause network partitions or disconnections, and two instantiated devices may no longer be able to communicate if all paths between them are broken. We propose a lightweight, soft-state exchange protocol for detecting disconnections in an instantiated TG. The protocol requires each instantiated device to send *periodic* HELLO messages (with period T) to its logical neighbor instances in TG, which reply with a HELLO-ACK.

Each instantiated device keeps track of its BFS parent and BFS children. If a BFS parent device stops hearing from one of its BFS children[7], it uninstantiates its child and starts searching for a replacement of the same category. The child meanwhile would stop hearing HELLO-ACKs from the parent (assuming bidirectional links), and will uninstantiate itself. This has been illustrated in Fig. 5.

[7]The parent concludes this if it does not receive a HELLO-ACK from that child before the expiry of its HELLO timer

Mobility of devices may also result in lengthening or shortening of routes between TG-node instances. Ideally, if there is no disconnection, the application should proceed without disruption, taking help from the underlying routing protocol. But such ideal conditions may not hold in reality where route failures at an intermediate node in DSR trigger route discovery which along with TCP re-transmissions after timeouts may cause several seconds of delay [11]. Hence, this can result in HELLO-ACKs not coming back in T seconds resulting in the conclusion that a disconnection has happened, even when the node instances are reachable from one another.

Recently, researchers have proposed solutions to the above problem based on explicit notification of route errors to TCP [8]. In this study, we do not attempt to alter TCP or DSR (including their default timer settings), and simply build our protocol on top of these protocols. Hence, if a device does not receive a HELLO-ACK from its neighbor in T seconds, we deem the neighbor to be disconnected. A reasonable value of T is one which is not low enough to cause significant control overhead[8], and not high enough such that disconnections are not detected fast enough. For our simulations, we chose $T = 7\,seconds$ ($> 6s$, the default TCP retransmission timer).

Process of Re-instantiation and Bookkeeping A BFS parent device acts like a *local coordinator* upon detecting a disconnection with its child at the expiry of the HELLO timer. It then locally broadcasts a search query for the same device category. After discovering a new replacement device, it instantiates the child in the TG and performs some bookkeeping steps to propagate the new state information to its logical neighbors as illustrated by the example in Fig. 6. With a little thought, one can see that information about the addresses of parents, children, children's parents and children's children is enough to handle single node disruptions in an instantiated task graph. For more details about the protocol, we direct the reader to [3].

4 Simulation Results

We simulated our algorithms using the popular network simulator *ns-2*[15]. We simulated 50 (moderate) and 100 (dense) specialized mobile devices moving in a 1000m $\times$ 1000m area according to the *random waypoint* mobility model [7]. The transmission range for each node was 250m and each device belonged to one of 12 different *device categories*[9].

We show results of simulation for the task-graph in Figure 3. The total simulation time was 600s – the instantiation process began at 50s, and at 60s, the *user/root* node started sending data to the data sinks. The data flow consisted of a CBR source, with a burst of S bytes of data every T seconds. We report results for $(S, T) = (12500, 5)$. Devices which are not part of the instantiated TG do not forward packets, and such packets are not buffered[10].

We observe from Fig. 7 that the average dilation for the MANET with 100 devices does not vary greatly with speed of mobility. This means that the average number of physical

[8] although exchanging HELLO messages with higher frequency could result in the DSR caches having fresher routes

[9] uniformly distributed

[10] In other words, if a device which was part of a TG becomes disconnected while there is a packet in transit, the packet is lost.

hops between two instantiated nodes in TG is low and remains approximately constant under mobility. Actually, even if a HELLO message does not reach a device within expiration time, owing to the uniform distribution of device categories in space, the re-instantiation process will find another device with similar attributes within a range that keeps the dilation value constant. A similar pattern is observed for the network with 50 devices, although D_{avg} is slightly higher in that case owing to a lower device density.

Fig. 8 shows the average times taken to discover and instantiate a device as a function of speed. We find that this does not vary much with speed for both cases (in the $0.1s - 0.15s$ range) for the reasons given in the previous paragraph. The values are lower for the 100 device MANET owing to more abundance of replacement device instances in the area.

We plot $AvgEffT$ as a function of speed in Fig. 9. We can see clearly that, in general, effective throughput drops with increase in speed. In Fig. 10, we plot the average number of re-instantiations underwent during the entire simulation time. The rate of change in network topology increases with speed causing more network partitions (mainly in the 50 device MANET) or route errors (in both MANETs). These in turn prevent HELLO packets from arriving in time, and this triggers more re-instantiations. Since packets caught in transit during the re-instantiation process are dropped (we do not consider application layer buffering in this work), $AvgEffT$ is directly affected by re-instantiations.

For the 100 device MANET, there are more disruptions for medium/high speeds than the 50 device MANET. This phenomenon is counter-intuitive because, for a greater device density, one would expect a lesser possibility of network partitioning, suggesting a more stable network with a less disruption-prone condition. However, *lesser/no* network partitioning does not mean *lesser* route changes; since DSR adopts caching, source routing and ring-zero search[11], it is sensitive to frequent route changes. This is because at medium-high speeds, routes change frequently causing the *cached* routes to become *stale* repeatedly. Stale routes trigger route errors which in turn trigger a route recovery mechanism at the source. Since the average node degree is higher in a 100 device MANET ($\frac{100\times\pi\times250^2}{1000^2} - 1 \approx 18$), there will be *more extensive* route caching at intermediate nodes with more alternate routes. Because of the higher density, a route error could cause many alternate routes to become stale simultaneously, i.e. there is a high correlation between neighboring route caches. This fact is not known by the algorithm, which keeps attempting to follow these alternate routes unsuccessfully. This causes greater packet delays and hence more disconnections and re-instantiations. This results in lower average effective throughput ($AvgEffT$) for that range of speeds (see Fig. 9).

5 Related Work

In recent years, *service discovery* in networks has been a popular topic of research in the industry as exemplified by IETF's SLP [10], Sun's Jini [14], and its variation for mobile networks called MOCA [4]. Our approach is different from these as it operates at a layer above service discovery and it can co-exist with any of these schemes.

[11] A device requesting routes initially queries its 1-hop neighbors, and these neighbors can answer from routes in their caches.

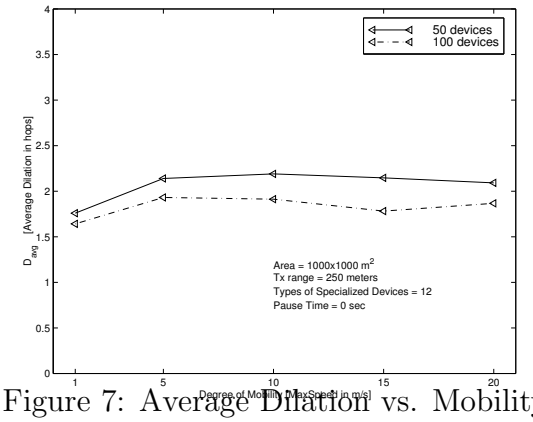


Figure 7: Average Dilation vs. Mobility

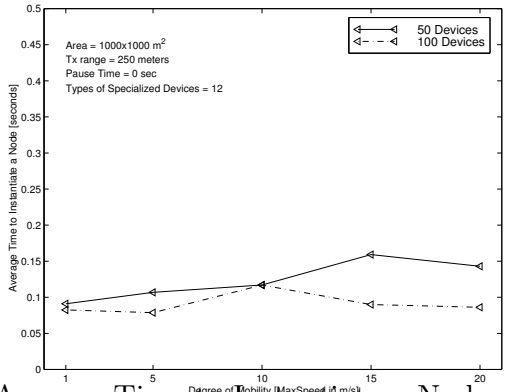


Figure 8: Average Time to Instantiate a Node vs. Mobility

Some researchers have proposed to capture the intent of users for discovering appropriate devices suitable to them [9, 1]. Neither of these approaches attempt to systematically model and make use of the relationships between the components of a distributed task, which is our principal focus.

IBM's PIMA has a vision somewhat similar to ours. In their vision paper [2], they argue very briefly for the design of applications in terms of sub-tasks instead of specific devices. However, they have not mentioned any approach for realizing this vision so far. Our task-graph concept on the other hand is a systematic and concrete approach which can help realize this vision.

6 Conclusions

In this paper we presented a distributed approach for application execution on a network of specialized, mobile devices. We developed a *task graph* abstraction for applications by taking into account the dependencies induced by logical data flow patterns existing between the components of an application. We then presented a distributed algorithm which yields an efficient embedding of a given task graph onto a MANET. In other words, the algorithm discovers devices in a MANET that are *most suitable* for executing a task, by an average hop-count measure called dilation. We also presented a scalable, local detection and repair mechanism for recovering from task disruptions caused by the mobility of devices. From our simulation studies, we observed that our algorithm was able to instantiate and re-instantiate TG nodes quickly with low dilation, and yielded a reasonably high effective throughput even

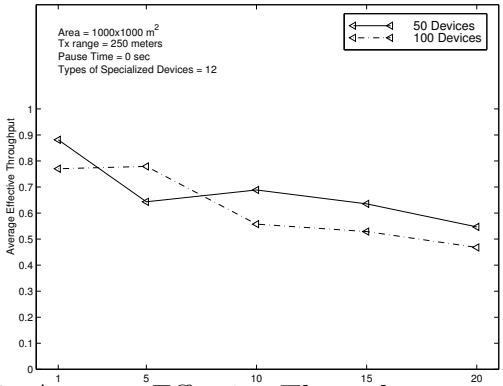


Figure 9: Average Effective Throughput vs. Mobility

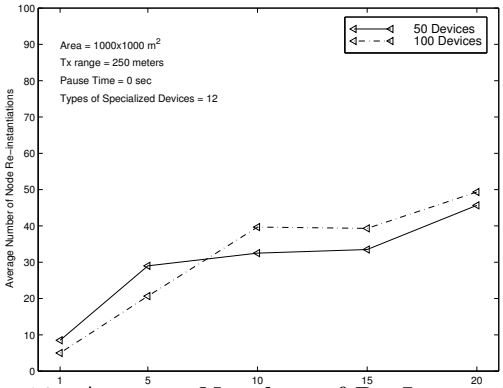


Figure 10: Average Number of Re-Instantiations

for fairly mobile situations. Although we do not focus on reliable task execution in this paper, we note that it can be achieved with buffering and re-transmissions in the application. This will be a topic for future research.

References

[1] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley, "The design and implementation of an intentional naming system," *Proc. ACM SOSP '99*, Kiawah Island SC, December 1999.

[2] G. Banavar, J. Beck, E. Gluzberg, J. Munson, J. Sussman, and D. Zukowski, "Challenges: An Application Model for Pervasive Computing," *Proc. ACM MobiCom '00*, Boston MA, August 2000.

[3] P. Basu, W. Ke, and T.D.C. Little, "A Novel Task Based Approach for Supporting Distributed Applications on Mobile Ad Hoc Networks," *BU MCL Technical Report, TR-07222001*, July 2001. *URL – http://hulk.bu.edu/projects/adhoc/TR-07222001.ps*

[4] J. Beck, A. Gefflaut, and N. Islam, "MOCA: A Service Framework for Mobile Computing Devices," *Proc. ACM MobiDE '99*, Seattle WA, August 1999.

[5] V. Bhargavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A media access protocol for Wireless LANs," *Proc. ACM SIGCOMM '94*, London, UK, August 1994.

[6] S. H. Bokhari, "On the Mapping Problem," *IEEE Trans. on Computers*, Vol. 30, No. 3, 1981, pp. 207-214.

[7] J. Broch, D. A. Maltz, D. B. Johnson, Y. C. Hu and J. Jetcheva, "A Performance Comparison of Multi-Hop Ad Hoc Network Routing Protocols," *Proc. ACM MobiCom '98*, Dallas TX, October 1998.

[8] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback-Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks," *IEEE Personal Communications Magazine*, February 2001.

[9] M. Esler, J. Hightower, T. Anderson, and G. Borriello, "Next Century Challenges: Data-Centric Networking for Invisible Computing The Portolano Project at the University of Washington," *Proc. ACM MobiCom '99*, Seattle WA, August 1999.

[10] E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services," *IEEE Internet Computing*, July 1999.

[11] G. Holland and N. Vaidya, "Analysis of TCP Performance over Mobile Ad Hoc Networks," *Proc. ACM MobiCom '99*, Seattle WA, August 1999.

[12] D. B. Johnson and D. A. Maltz, "Dynamic Source Routing in Ad Hoc Wireless Networks," in *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, pp. 153-181, Kluwer Academic Publishers, 1996.

[13] R. Monien and H. Sudborough, "Embedding one Interconnection Network in Another," *Computing Suppl. 7*, 1990, pp. 257-282.

[14] Sun Microsystems, "Jini Technology Core Platform Specification," *URL – http://www.sun.com/jini/specs*

[15] VINT Network Simulator: ns-2. *URL – http://www.isi.edu/nsnam/ns*