# A Task Graph Based Application Framework for Mobile Ad Hoc Networks*

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

MCL Technical Report No. 08-12-2001

**Abstract**– We propose a task graph based framework for modeling and execution of distributed applications in mobile ad hoc networks. Our framework represents a distributed application by a graph composed of nodes and edges in which the nodes *logically* represent application sub-tasks that need to be completed and the edges represent associations, with certain attributes, between nodes. During application run-time, suitable devices that can complete the sub-tasks and can satisfy the attributes of the associations between them are selected on-the-fly to execute the application. New devices are selected to continue application execution if old devices become unavailable due to mobility. Thus, we de-couple the application from a specific set of devices and allow its execution if there is at least one suitable device in the network for carrying out each of the required sub-tasks. In this paper, we propose an application execution protocol to realize this vision and show simulation results which indicate that our approach is practical for environments with low user/device mobility.

*Proc. IEEE ICC 2002*, New York, NY, April-May 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.

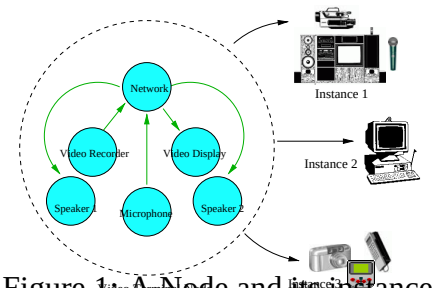

Figure 1. A Node and its instances

# 1 Introduction

New exciting possibilities are being opened by the continual decrease in size, increase in the quantity of processor-embedded devices that surround us, and the emergence of network enabling technologies such as Bluetooth [1] and IEEE 802.11 [2]. In a world where any display device, be it a PDA, a monitor, or even a projector can act as my "TV screen", and likewise, any earphone or speaker can act as my "radio" or "MP3 player," new challenges need be overcome to tap into the full potential of such environments.

Traditional operating systems and network protocols have been generally concerned with specific memory or network addresses. While the abstraction of "address" is essential, its underlying emphasis is on reaching a *specific* host that can complete a required task. However, with the proliferation and multiplicity of devices that can perform similar tasks, we need a shift in focus towards emphasizing on how to characterize an application logically and how to allow an application component to "dynamically bind" itself to a device that can complete the corresponding task. No longer should we be prevented from running an application when one specific device becomes unavailable. Also, with specialized heterogeneous devices offering different services in the environment of the user, an application may need to simultaneously request multiple services and therefore must be able to determine the correct communication relationships required between these devices to ensure proper execution. In addition, due to the shrinking size of the devices and added wireless networking capabilities, device mobility, which implies fluctuating service availability, is another issue that must be addressed in order to develop, run and support different applications in a *Mobile Ad-hoc NETwork* (MANET) environment.

To tackle the challenges described, we introduce a *Task Graph* (TG) based framework for application development and execution in MANETs. Our TG approach is inherently distributed, as we believe many of the applications for MANETs must be. It also de-couples the task to be performed on any specific host, allowing an application to take advantage of the multiple devices that are present in the environment. In addition, it supports hierarchical composability, allowing sets of devices to be logically grouped together and be treated as a single unit essentially enabling the network to offer new services based on existing ones. We offer support for applications built in our TG framework by adding an intermediate layer above the routing layer, the *Task* layer. Thus, a "tele-conferencing virtual terminal" application, with its requirements of audio input/output, video input/output and networking capabilities, need no longer be restrained to only one type of hardware configuration (typically a networked multimedia computer). Instead, it can also be executed by a combination of electronic devices in a home entertainment center or even by connecting a PDA, a digital camera and a cell phone as shown in Fig. 1.

In this paper, we present an initial implementation of the "task layer" of our TG-based approach in the network simulator *ns* [3]. We introduce some useful metrics to evaluate the performance of the system and show that such approach is practical under low mobility environments. In the next section we present related work in the area. Section 3 introduces theoretical foundations

for the TG concept and describes the protocol implemented by the Task layer. Section 4 shows simulation data. We conclude in Section 5 with an analysis of the data obtained and a discussion of the conditions under which our approach performs more efficiently.

# 2 Related Work

Task graphs have been used in the parallel computing and scheduling literature for representing tasks that can be split *temporally* into sub-tasks and then allocated to different homogeneous processors for reducing total completion time [4]. Our formulation of task graphs extends the concept described above by including heterogeneous devices, and switches focus from minimizing the completion time of distributable sub-tasks to ensuring the presence of suitable (heterogeneous) devices that can cooperatively complete the execution of a distributed application.

The de-coupling of a service from the host providing the service is the idea behind Service Location Protocol (SLP) [6] and Jini [7], as well as MOCA [8], which offers a Jini-like service to mobile computing devices. Intentional Naming System (INS) [9] is a more sophisticated scheme which attempts to capture "user intent" to find an appropriate service. It also has a late binding feature that allows applications to bind themselves to a service and not to any specific host. While we advocate the same de-coupling principle, we apply such a principle in building a systematic framework in which complex distributed applications can be modeled, with the inter-relationships of its components expressed and integrated with mobility management issues.

Both the Portolano project [10] and IBM's Platform-Independent Model for Applications [11] represent visions that consider applications as services provided to or task executed on behalf of the end user, which should be independent of the specific user interface devices available. Our work shares the same vision, and is a concrete and systematic approach that attempts to realize it.

# 3 A Task Graph Application Framework

**Concepts and Definitions** A *device* is a physical entity that performs at least one function such as interaction with its physical environment, computation, or communication with other devices. There is a *link* between two devices with networking capabilities when at least one device can receive data sent from the other. A link can be unidirectional or bidirectional. A device's capabilities are summarized by a set of *attributes*. Attributes can be static, such as the resolution of a digital camera, or dynamic, such as the location of the camera. A *service* is a functionality, possibly offered by a device or a collection of cooperating devices, in which a desired output is obtained through processing a given input data. A *node* is an abstract representation of a device or a collection of devices with a minimal set of attributes that can offer a particular service. A node is simple when it represents a single device and it is complex when it represents multiple devices. An *edge* is an association between 2 nodes that satisfies certain attributes necessary for the completion of a task. A *task* is work executed by a node. If the node is complex, then the work done by each of the constituents of the complex node is considered a "sub-task" of the task. An *atomic* task (defined in terms of a device's capability and subjective design criteria) is an indivisible unit of work which is executed by a simple node.

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 directed edges (from source node to the destination node) associated

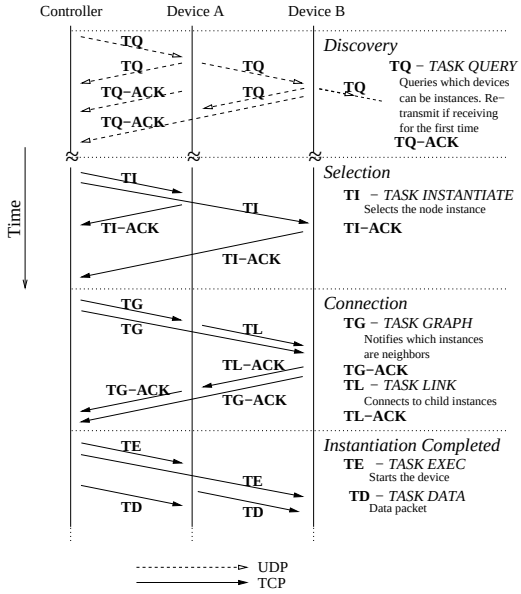


Figure 2: Application Instantiation Packet Exchange

with the task graph. A network of mobile devices can be represented as a graph $NG = (D_N, L_N)$, where $D_N$ is the set of devices that are present and $L_N$ is the set of links connecting any two devices. If we define a *path* of length $N$ as an ordered sequence of $N$ unique links, in which the destination of link $n$ is the source of link $n+1$, $n \in \{1, \cdots, N-1\}$, then an *embedding* or an *instantiation* of a task graph $TG$ onto a MANET $NG$ is the process of selecting a pair of functions $(\varphi, \psi)$, such that $\varphi : V_T \rightarrow D_N$ and $\psi : E_T \rightarrow P_{L_N}$, where the service and the set of attributes of a node $v \in V_T$ are contained in the set of services offered and the collection of attributes of a device $\varphi(v) \in D_N$, and where the attributes of an edge $e \in E_T$ are satisfied by a path $\psi(e) \in P_{L_N}$, $P_{L_N}$ being the set of paths formed by the links in $L_N$. A particular pair of functions $(\varphi, \psi)$ represent an *instance* of the task graph $TG$, and a device $\varphi(v)$ is an "instance" of the node $v$. If an instance of a node becomes unavailable, the process of selecting a new suitable device is called *node re-instantiation*. The source node of an edge is called a "parent" of the destination node, and the destination node is a "child" of the source node.

**Instantiation and Mobility Management in Task Graphs** We now introduce our protocol which can be implemented at a layer above the routing and transport layers to support the task graph abstraction. It assumes the presence of one node that will act as the *controller* (or *coordinator*) of the application, i.e., it is in charge of instantiating each node of the task graph and executing the node re-instantiation process, if necessary. Such a centralized approach simplifies the problem of instance synchronization, since all devices participating in the task know that the coordinator will always select a new instance if an existing instance becomes unreachable. In addition to that, this protocol can take advantage of any available powerful computing device in the environment to optimize the instantiation process, if applicable.

The *Application Instantiation* process can be decomposed into three major phases:

- *Discovery*: The controller broadcasts a TASK-QUERY packet with a list of nodes and attributes desired. A device receiving this packet for the first time rebroadcasts it, and one that can satisfy the attributes of at least one node in the list sends a TASK-QUERY-ACK

back to the controller indicating the node that it can be an instance of.

- *Selection*: The controller selects from among the possible instances one device that will be the node instance. The criterion we have used in this paper to select a device is to minimize the average path length over all edges in the task graph (see Sec. 4).

- *Connection*: Each selected node instance S of the task graph is notified of its parent and children devices. Then, S proceeds to contact its children and replies to any contact attempt made by its parents. The controller is notified when all of S's parents and children have been successfully contacted.

Fig. 2 shows the packet exchange for the application instantiation part. Data exchange (through TASK-DATA packets) can take place soon after the completion of this process. The transport protocol at the Discovery part does not have to be reliable, thus we characterized it as UDP in Fig. 2. However, during the Selection and Connection phases, the protocol should be reliable, so as to distinguish between temporary path unavailability (and consequent packet drop) and true unreachability (see Sec. 4 for details).

*Mobility Management* of devices involves primarily two problems: (1) how to "detect" when a device cannot be reached and (2) how to react to such an event.

We deal with the *detection* problem by a *periodic* packet exchange between the instances and the controller. The controller sends every $T$ seconds ($T$ is the task layer timeout period) a TASK-CONTROLLER-HELLO packet to each selected instance (and to each possible instance until the completion of the selection part). In the same way, each selected instance (and each possible instance) sends with period $T$: (1) a TASK-DEVICE-HELLO packet to the controller and (2) a TASK-NEIGHBOR-HELLO to each of its parent and child node instances, if these are known. Any device receiving such HELLO packets must send back an acknowledgment. If the sender of a HELLO packet does not receive any TASK-related packet (with the exception of TASK-QUERY) from the intended recipient within $T$ sec, then the recipient is deemed unreachable. If the unreachable host is the:

1. *Controller*: The sender device considers that the application cannot proceed and switches itself to an idle state (case *B* in Fig. 3).

2. *Node instance*: The controller may repeat the discovery, selection and connection steps as necessary but only with respect to the node with the missing instance, i.e., other instances are kept and no query is made regarding them. In Fig. 3, at *C*, the controller attempts to *re-instantiate* the missing node and starts from the discovery phase.

3. *Neighbor node instance*: The sender device informs the controller that it has a missing neighbor node instance (case *A* in Fig. 3). The controller performs a node re-instantiation if necessary and informs the reporting device of its neighbor node's new instance.

4. *Possible node instance*: The controller simply drops the unreachable device from the list of possible instances.

The main principle we tried to follow in designing the protocol was that an application should proceed as long as there are suitable devices on which it can run. In case such devices do

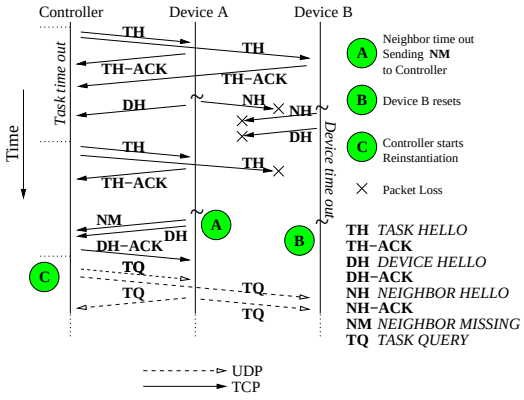


Figure 3: Mobility Management

not exist the application layer can be informed via a proper API mechanism to decide on a proper course of action. The "fate sharing" characteristic that a device acting as coordinator currently has with the application can be avoided by developing a state management and recovery mechanism to elect a new coordinator when the original becomes unavailable. This topic is beyond the scope of this paper and needs further research.

# 4 Simulation Results

We implemented our algorithm in the network simulator *ns*[3]. The routing protocol adopted was DSR[12]. One helpful characteristic of DSR is source routing, in other words, packets carry addresses of intermediate devices in the path between the source and the destination. This information gives the controller a partial knowledge of the network topology, which we explore in the selection phase of the instantiation. During selection, the controller must choose devices as node instances of the task graph, from among multiple devices that answer its query. We apply the "breadth-first search" algorithm, starting from the controller node and choosing the child instance that has the shortest path length (according to the network topology known by the controller at that instant) to its parent instance.

TCP was chosen as our reliable transport protocol. It minimizes the probability of mistaking a temporary route failure with a more permanent phenomenon of device unavailability due to mobility. However, delays due to TCP's inherent assumption of congestion in presence of packet loss, coupled with its subsequent backoff period before attempting retransmission, may result in HELLO packets arriving after their expected time and triggering re-instantiations when the old instance is still reachable. Researchers have proposed modifications to TCP to improve its performance over MANETs [13], but in this paper, we have built our protocol on top of existing ones. For simulations, we chose the task layer timeout $T = 7$s (the default TCP retransmission timer's value is 6s).

**Metrics and Results** From our simulations we observed the value of the average *Dilation*[1] of our instantiated task graphs. Average dilation is the average length of the paths mapped by an embedding algorithm over all the edges of the task graph. Lower dilation means less number of hops a packet must traverse, on average, when traveling between node instances. The average *Time*

[1]This metric was originally introduced in [5].

*to Instantiate* a task graph is the second metric we study – this is the time taken from the start of the discovery phase until the end of the selection phase. A related metric is the average *Time to Re-Instantiate*, which is the time taken to rediscover and re-select a new device when an existing instance becomes unreachable. The fourth metric we look at is the average *Effective Throughput*, which is the average number of application data units (ADUs) received at the data destinations over the number of ADUs that were supposed to be received by the intended targets if no packet drops occurred.

We simulated for 100 devices moving in a 1000m $\times$ 1000m area, with a transmission radius of 250m, and following the Random Waypoint mobility model [14]. The devices were "simple" and belonged to one of the 12 types we had for the simulation. Each device's type was randomly selected following a uniform distribution.

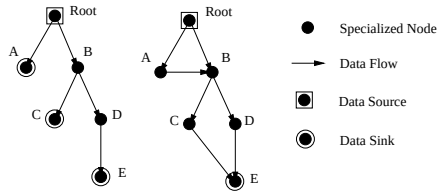


Figure 4: Task Graphs for Simulation

We show results for the task graphs in Fig. 4, named *TG1* and *Treetask*. The total simulation time was 600s, with the instantiation starting at 50s, and the controller beginning a CBR flow (12500 bytes every 5 seconds) at 60s. Devices which are not instances of the task graph do not forward these packets, thus if a downstream instance is missing at the moment a packet is in transit, the packet is simply dropped[2]. We simulated for the maximum device speed of 1, 5, 10, 15 and 20 m/s, and for a pause time of 0 and 60 s.

Fig. 5 shows that the average dilation does not change significantly with maximum speed, which is due to the fact that the transmission radius, coupled with the spatially uniform device distribution and the size of the simulation area, simply ensured that the devices needed for instantiating the task graphs would be within two hops.

The "average time to instantiate" metric (Fig. 6) shows an irregular pattern in our simulation results and we can see a peak when the maximum speed is 10 m/s. Such peaks happen most when

[2]Data packets that follow different paths to a data sink are assumed to have been processed by the nodes they meet in the way and contain different information. Thus multipath availability in this study does not contribute to increased reliability of data transmission.

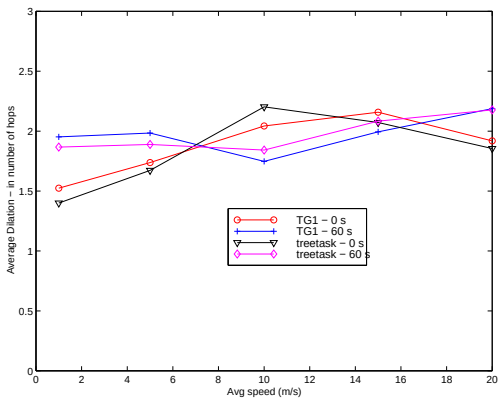


Figure 5: Avg. Dilation

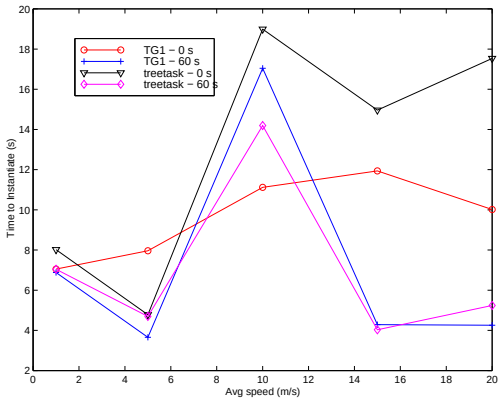


Figure 6: Avg. Time to Instantiate

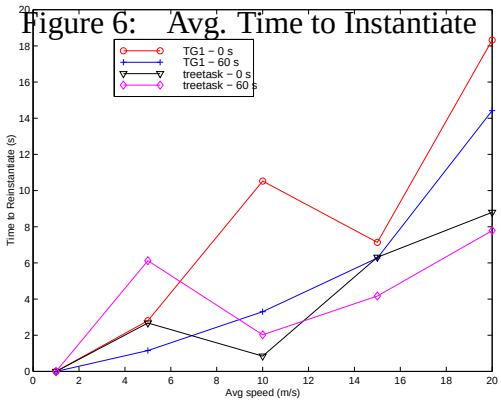


Figure 7: Avg. Time to Re-Instantiate

there is a packet loss or temporary route failure (congestion or partition), forcing the controller to go through multiple discovery phases. For the "average time to re-instantiate" metric shown in Fig. 7, we can see that it increases with the maximum speed. At higher speeds the route information cached by DSR becomes stale more quickly, and frequent route updates can result in higher delays in packet exchange or even packet drops, if the buffers of the routing agent become full. This eventually translates into the controller thinking that an instance may be unreachable, which will trigger the re-instantiation process, and for the re-instantiation process to complete, all instances of the task graph must successfully exchange packets with the controller within one timeout period. Our simulation results show that it probably takes multiple timeout It is interesting to notice that even though the re-instantiation time is long data are still reaching their intended destinations, as shown by the average effective throughput in Fig. 8. This shows that even though the whole task graph might not be fully instantiated (there are subtasks uninstantiated), still the parts which are instantiated can carry on effective communication. This is made clearer by the higher throughput of the Treetask graph, in which the data flows go through less number of instances (only parent → child flows).

# 5 Conclusion

We have proposed in this paper a task graph framework to model distributed applications and support their execution in MANETs. Applications are modeled as graphs composed of nodes, representing sub-tasks to be completed, and edges, representing associations with certain attributes between the sub-tasks. We introduce run-time support for applications modeled in this way through

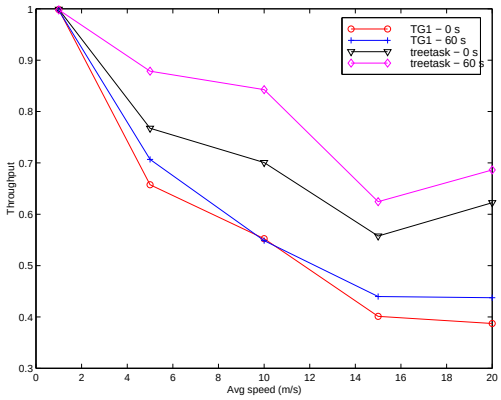


Figure 8: Avg. Effective Throughput

a task layer which supports dynamic binding of the tasks to specialized devices and performs their re-selection if any previously selected device moves away and becomes unreachable.

Our protocol assumes the presence of a controller for each application, responsible for performing discovery and selection of suitable devices (task graph instantiation) on which the application will be executed. Also, we require periodic task layer packet exchange among the selected devices to monitor availability. In the case of unavailability, the controller re-instantiates the nodes (selects new devices) whose instances are missing. Simulation results based on our protocol show that our approach is suitable for moderate to low mobility scenarios, e.g., where maximum speed does not exceed 10 m/s in a $1\ km^2$ area. Such environments can benefit from a higher throughput and lower application instantiation and re-instantiation delays. We noticed that task graphs with more independent data flows (flows intersect at less nodes) are more resilient to high mobility and perform better in terms of throughput values. Distributed instantiation protocols, although more complex in their operation, are likely to be more robust under higher degrees of device mobility. We have investigated such protocols elsewhere [15].

# References


[1] Bluetooth SIG, *http://www.bluetooth.com*

[2] B. P. Crow, I. Widjaja, J. G. Kim, P. T. Sakai, "IEEE 802.11 wireless local area networks," *IEEE Communications Magazine*, Vol. 35, No. 9, Sep 1997, pp. 116-126.

[3] VINT Network Simulator - ns (version 2). *http://www.isi.edu/nsnam/ns*

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

[5] R. Monien and H. Sudborough, "Embedding one interconnection network in another," *Computing Suppl. 7*, 1990, pp. 257-282.

[6] E. Guttman, "Service Location Protocol: automatic discovery of IP network services," *IEEE Internet Computing*, July 1999.

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

[8] J. Beck, A. Gefflaut, and N. Islam, "MOCA: a service framework for mobile computing devices," *Proc. ACM MobiDE*, Seattle WA, Aug 1999.

[9] 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, Dec 1999.

[10] 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, Aug 1999.

[11] 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, Aug 2000.

[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, chapter 5, pages 153-181, Kluwer Academic Pub., 1996.

[13] 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*, Feb 2001.

[14] 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.

[15] P. Basu, W. Ke, and T.D.C. Little, "A novel approach for execution of distributed tasks on mobile ad hoc networks," *Proc. IEEE WCNC '02*, Orlando, FL, March 2002, in press.