A Contiki-based Prototype for Creating Wireless Ad Hoc Grids

Elisa Rondini
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
E.Rondini@cs.ucl.ac.uk

Stephen Hailes
Dept. of Computer Science
University College London
Gower Street, London WC1E 6BT, UK
S.Hailes@cs.ucl.ac.uk

1 Introduction

There are several examples of situations in which information from a wireless sensor network (WSN) must be gathered, aggregated and processed to provide a meaningful summary to an external agency, all within the constraints of the processing power and bandwidth available within the network. Examples range from sensors used for traffic and wildlife habitat monitoring through to disaster and emergency scenarios. Our particular interest is in supporting emergency response for incidents occurring in indoor environments (e.g., metro systems, etc.).

At present, there are only two choices about where computation might occur within a sensor network: (i) on individual sensor nodes, with the advantage of achieving substantial data reduction, decreasing the cost of transmission and avoiding congestion; (ii) outside the sensor network, with the sensors simply supplying the data required for the calculation. This latter approach does simplify sensor nodes but necessitates a higher bandwidth network. If, over the deployment lifetime of a system, applications reach a level of sophistication at which they cannot be executed by a single node, then it would seem that the only option is to have processing performed centrally. However, the distributed systems community has addressed the problem of too little computing power on a single node by distributing computation within a computational grid. Such grids have, to date, been formed from high-end processors with fast network connections; these do not experience the types of congestion that are inevitable in the broadcast radio medium. Moreover, the approaches to computation distribution proposed for WSNs [2, 5] tend to focus exclusively on the CPU availability of nodes during the computational distribution process, also ignoring real communication issues.

The key contribution of this paper is a demonstration that this approach is simplistic and that there are significant benefits to be gained by considering local network conditions of candidate nodes in addition to load information. We present results from our implementation on a real Tmote Sky sensor testbed running the Contiki Operating System [1].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Contiki Workshop'07, March 26–27, 2007, Kista, Stockholm, Sweden.

2 General Model

We assume that the jobs to be performed by devices will be dynamically determined and sensor nodes may not be individually capable of executing the computational intensive applications sent to them over their lifetime. Neither is it possible simply to ship all data to a more capable sink because the network is inherently bandwidth limited. Thus, we assume that there may be a need to distribute computation, and that the network density is such that nodes can rely on the presence of other nodes for help. When a job must be executed that is either outside the computational capabilities of a single node or for which the real-time deadline cannot be met by a single node, subtasks into which a job is split are distributed to nearby nodes. The outputs of these tasks are combined to give a result that requires less bandwidth to transmit to the sink node than would the raw data. Clearly, there is a cost, both in terms of power and localised network contention when offloading tasks and in gathering the information about current network conditions, against a corresponding benefit in the execution parallelisation and data reduction that reduces contention in the wider network and, in particular, in the areas close to (remote) sink nodes. As a prerequisite of the realisation of the above vision, it is necessary to establish whether, and to what extent, bandwidth considerations affect the dynamic task allocation process.

3 The Bandwidth Problem

Assume the existence of a sensor network with a client node that needs to distribute tasks and two server nodes to which it could potentially distribute them. In this scenario, conventional load distribution algorithms would simply select the least loaded node as the target for distribution. However, this ignores network effects so, for example, the initial distribution of code and data, any subsequent requirement to communicate with the originating client, and the control overhead of maintaining accurate information about server state at the client side, all add to network utilisation. Thus, if we distribute a large task with substantial I/O requirements into an area of existing congestion, we will not only cause that task to run more slowly, but we would also expect others either within the area, or using the area for communication, to run more slowly too. On the other hand, if the task is small and CPU-bound, we would expect much less effect on both the task itself and on other nodes, regardless of placement.

4 Load Sharing Algorithms

Many load sharing and load balancing protocols have been presented in the literature. Our approach is based on the creation of ad hoc grid formations, in which a client communicates only with nearby servers, asking them for help in the computation of a specific job, trading off the costs of local distribution against faster execution and reducing the effect on the communication of others. For this reason, we selected, adapted and implemented two simple

load sharing algorithms to take account of network usage: the Auction algorithm and the Lookup List algorithm (adapted from [3]). To cope with the limited space, in the present abstract we only describe the Auction algorithm.

- **Auction Algorithm**
  The Auction is a very simple client/server reactive sender-initiated algorithm. For each task, the client broadcasts a request message containing details of its CPU and bandwidth requirements. On receiving this message, each server checks its CPU and bandwidth availability and sends a bid to the client. The client selects the best bid and broadcasts a message notifying the winner.

Both the algorithms have a phase in which the client has to choose the best server node to which offload the task execution. For bandwidth unaware systems, only CPU load is taken into account, whereas for bandwidth aware systems, a simple linear combination of CPU and bandwidth availability information determines utility. After the task offload, the server launches a process to compute the task, occasionally exchanging messages with the client if needed, and it sends the final result back when the computation terminates.

5 Experimental Setup and Results

In this section, we outline some early stage results obtained by testing our approach on a real sensor testbed consisting of 7 Tmote Sky motes running the Contiki Operating System [1], as shown in Figure 1. We tested the application with stationary motes and fixed transmission power levels but actual computation, actual profiling of the medium and actual network traffic. On the extreme left of Figure 1, we inserted a streaming sensor that continuously polls the two servers on the left, so saturating the medium in its reception area. The radio power level of this sensor was set to give a physical packet reception range of ∼50cm, rendering it inaudible to those servers on the right.

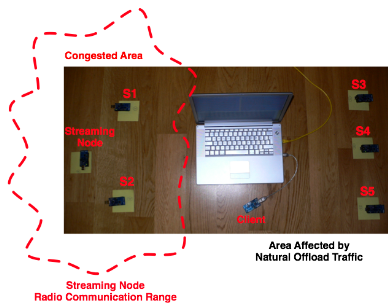

Figure 1. ***Overall view of the global simulation environment based on a real Tmote Sky sensor testbed.***

We used the Contiki OS to implement both the algorithms Auction and Lookup List. Client and servers can handle both UDP and TCP communications: the former is used to manage the negotiation phase characterising each algorithm, whilst the latter is used to cope with packets offload involved during task execution. The TCP protocol is more communication intensive than the UDP, but we need its reliability to be sure that every packet is received by the other end point. On the other hand, for a negotiation phase we might use a less reliable but more lightweight protocol such as UDP. Moreover, we assume that the servers always have to be ready and available to manage all kinds of request and the clients can handle several connections to cope with a variety of task offloads. The possibility of opening several TCP connections at the same time implies that different processes are allowed to execute tasks in a parallel way running on the same node. These processes have been implemented using the Contiki OS' own protothreads [4].

We measured the performance of the system for each load sharing algorithm in three subscenarios: (i) without contention (or rather with only traffic generated by offloading tasks), (ii) with contention as described above but considering only computational requirements, and (iii) with contention and using our algorithms to take account of bandwidth as well as CPU requirements. In Figure 2 we see that, for the Auction algorithm, performance in the presence of contention is poor unless one considers bandwidth during task allocation, in which case system performance approximates that of the uncongested case. This is as expected, though the dynamics are somewhat complex. Moreover, as the error bars show, the performance variations are small for the case with contention and bandwidth control because the algorithm manages to avoid the delays that would occur by sending tasks to congested areas. Similar results are obtained with the Lookup List algorithm. However, the overall performance of the Lookup List algorithm is better than that of the Auction algorithm.

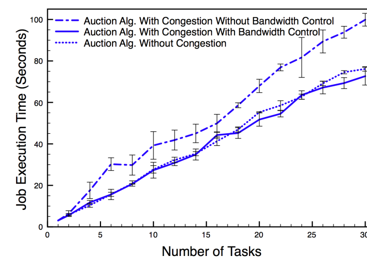

Figure 2. ***Performance of the Auction algorithm with a different number of tasks.***

6 Conclusion

In this work, we outlined results that represent an exploration of what we have termed ad hoc grids: the convergence of mobile ad hoc sensor networks and computational grids. In our vision, mobile nodes are required to execute jobs that exceed their local computational capabilities and, consequently, are compelled to break those jobs into tasks and distribute them. There is a cost in transporting such tasks to nearby nodes, but we have shown, albeit in a limited (but nevertheless real) environment, that the data reduction that such tasks can effect reduces overall network contention and so average latency, particularly if network availability is taken into consideration in addition to computational load. We have started work on larger scale experimentation and node mobility.

7 References

[1] Contiki Operating System. `http://www.sics.se/~adam/contiki/`.

[2] Z. Abrams, H.-L. Chen, L. Guibas, J. Liu, and F. Zhao. Kinetically Stable Task Assignment for Networks of Microservers. In *Proceedings of the 5th annual ACM/IEEE International Conference on Information Processing In Sensor Networks*, pages 93–101, April 2006.

[3] P.-J. Chuang and C.-W. Cheng. On File and Task Placements and Dynamic Load Balancing in Distributed Systems. *Tamkang Journal of Science and Software Engineering*, 5(4):241–252, 2002.

[4] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali. Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems. In *Proceedings of the 4th annual ACM International Conference on Embedded Networked Sensor Systems*, pages 29–42, November 2006.

[5] Y. Xu and H. Qi. Distributed Computing Paradigms for Collaborative Signal and Information Processing in Sensor Networks. *Journal of Parallel and Distributed Computing, ACM*, 64(8):945–959, August 2004.