Towards the Coexistence of Divergent Applications on Smart City Sensing Infrastructure

Rajiv Ramdhany, Geoff Coulson

School of Computing and Communications
Lancaster University, Infolab21, LA1 4WA
Lancaster, UK

{r.ramdhany, g.coulson}@lancaster.ac.uk

Abstract. Based on the trends of divergent sensor node software evolution and on-node multi-application coexistence observed from a real smart city WSAN deployment, we propose requirements for optimum exploitation of the infrastructure. In particular, we advocate the need to switch between concurrency models depending on usage context.

1 Introduction

Long-term Wireless Sensor and Actuator Network (WSAN) deployments as in smart city infrastructures/testbeds are becoming increasingly large-scale, multipurpose and 'infrastructural' in nature. Because of their multi-purpose nature and the major capital investments involved, we envisage that the owners/ operators of future infrastructural WSAN deployments will increasingly be driven to adopt agile, multi-stakeholder usage models. Economically-viable exploitation models require multiplexing concurrent experiments, applications and users on the WSANs and are characterised by *1)* the need to run *divergent* and *ever-changing* software configurations across their (heterogeneous) node base, and *2) multi-application sensing*, the sharing of sensing hardware or sensed data among multiple coexisting applications.

As a growing set of novel applications are requiring additional complexity and 'intelligence' to be pushed to the WSAN leaf nodes [1], we argue a simplistic concurrent-usage model based exclusively on sensor-data sharing or on coarse-grain scheduling of sensor nodes via group reservations (and node reprogramming) [2] [3] does not deliver the flexibility and efficiency required for an optimum, cost-effective infrastructure exploitation.

In this paper, we identify several requirements for enabling the coexistence the WSAN applications on sensor nodes and focus on the need for OS-level concurrency. The execution model of current WSN operating systems embodies a particular concurrency strategy usually based on one of the following: split-phase FIFO task scheduling, synchronous/asynchronous events, cooperative multithreading, time-sliced preemptive multithreading or real time scheduling. The choice of a concurrency model often implies a trade-off between the degree of concurrency, scheduling predictability and resource usage, made necessary by the resource paucity in the host embedded devices. For example, event-based execution models are more energy-efficient and conservative at memory utilisation [4] than multithreading but result in unpredictable scheduling behavior when event handlers are long-lived. Preemptive multithreading models, in contrast, are more deterministic and offer better event processing capabilities in terms of meeting processing deadlines; but they incur memory overhead and higher energy consumption [4]. Due to ever changing applications and software configurations in smart city WSANs, we argue that this trade-off

is itself *dynamic* and *divergent*. Concurrency models have associated costs/benefits and tend to suit particular WSAN usage scenarios and node roles. Hence, not all nodes need to commit permanently to a single concurrency model. Instead, we advocate *i)* concurrency strategy selection for every IoT node based on its tasks assignment and role (e.g. application data fusion point or packet forwarder) and, *ii)* switching between concurrency models when WSAN nodes participate in different applications serially or simultaneously.

2 Motivation and State-of-the-Art

We consider the usage model of a real-life smart city WSAN, in occurrence the SmartSantanter platform, to explain our position. The SmartSantander platform [2] is a multi-purpose city-scale experimental research facility in support of typical applications and services for a smart city. It comprises of a large number of Internet of Things (IoT) devices which are deployed in several urban scenarios, and federated into a single testbed for the purpose of concurrent experimentation and service provisioning. In addition to testbed services for experimentation, a number of smart-city services are currently deployed, namely: city parking control & management, environmental monitoring, on-vehicle mobile sensing, transport vehicle tracking, traffic congestion monitoring and precision irrigation for urban parks. An analysis of these concurrent services/experiments reveals that the tasks assigned to IoT nodes usually involve: *1)* sensor data sampling, *2)* data collection from neighbouring nodes *3)* data processing (e.g. temporal aggregation, compression or signal processing) *4)* data encryption/decryption before processing/forwarding, or *5)* routing data to sink nodes. Further, the functionality that need to be deployed on sensor nodes is itself application-specific and diverse, for example conditional triggers for sensor readings, routing behaviour triggered by frequency-analysis of sensor data, sophisticated data aggregators and compression to name but a few. Also evident is the need for cohabitation of applications on nodes (for instance, environmental monitoring, parking control and experimentation sharing repeater nodes).

Our analysis has identified the following requirements for achieving concurrent sensor-node use and gauged their current level of support in the state-of-the art.

- **Dynamic local sensor-node reconfiguration -** the capability to dynamically augment/adapt sensor node software by instantiating/removing software modules using runtime dynamic linking. Various degrees of support for in-situ software update are offered by modular WSN OSes. Contiki [5] only supports one updatable module at any one time loaded on top of a fixed kernel. Lorien [6] supports runtime addition, removal and replacement of software components more naturally.
- **Support for managed sensor node software evolution**. A systematic ongoing, efficient and non-disruptive means of determining and executing software compositional changes on WSAN nodes such that different nodes can evolve along different lines to support the software configuration that meet the needs of different stakeholders. OS-level architectural reflection and a suite of lightweight protocols are used in [6] to track/report software configurations, and perform component dissemination and instructed configuration updates on behalf of users.
- **Application Isolation.** With multiple applications sharing a node, it is necessary to isolate access to node resources (hardware, memory and network) to avoid applications

corrupting each other by overwriting unprotected memory locations or race conditions on sensors or network interfaces. Since hardware support for virtual memory (personal address spaces) is inexistent on current WSAN MCUs, the common approach for memory isolation is through the creation and policing of software protection domains as proposed in [7] or sandboxing via on-node virtualisation as in the Maté VM [8]. Hardware access isolation can be provided through *virtualised access*: SenShare [12] provides a compile-time Hardware Abstraction Layer (a set of interfaces used by applications) which redirects hardware access requests to a runtime comprising of virtual hardware controllers that serve access requests to the underlying hardware. Network traffic isolation is achieved by maintaining per-application network overlays as in Lorien [6] and SenShare [12].

- **OS-level Concurrency.** This involves scheduling the processor between tasks from coexisting applications and/or experiments efficiently so that they meet their processing deadlines. Concurrency in TinyOS [9] is realised through events, and the split-phase operation of tasks (deferred computation/calls) and event handlers. Tasks run to completion and are only preemptible by events (triggered as a result of hardware interrupts); they must therefore be kept reasonably short as not to delay other time-sensitive tasks such as network stack packet-processors. At the other end of the design space, concurrent tasks are implemented as threads as in Mantis OS [10]; applications as well as system components such as stacks may spawn threads which must be scheduled. Context switching between threads is performed by a thread scheduler, which follows a priority-based algorithm with round-robin semantics.

Different concurrency models are adopted by existing WSN OSes but none provide optimal scheduling/resource-consumption under the full range of operating conditions encountered in multi-purpose WSANs. Run-to-completion event-driven systems such as TinyOS are efficient in terms of memory usage (single stack usage) and energy consumption (more time spent in idle mode than other schemes) [4]. But they force programmers to model their algorithms as finite state machines, split-phase operations and short tasks. Long-running tasks may cause other parts of the OS to perform poorly as they cannot be preempted. At high packet arrival rates, for instance, packet processors are unable to dequeue packets from the radio module's buffer fast enough since their execution cannot be prioritised over other tasks. The average execution time of task is variable and this non-deterministic scheduling behavior is further exacerbated under high load.

Task prioritisation and time-sliced preemption are accepted notions in contemporary OSes to achieve low execution-delay variation and fairer scheduling between application tasks although this comes at a price. Preemptive multithreading enables independence between tasks [4] but requires extra programmable memory for capturing and restoring execution context when threads are switched. Each thread has its own stack which typically has to be over-provisioned as stack-space consumption of threads is hard to predict. Moreover, locking mechanisms are required to serialise concurrent thread access to shared resources, introducing additional overhead. Context switching causes the processor to spend less time in idle mode, resulting in higher battery power consumption.

Cooperative threading support in WSN OS such as Tiny-Threads [11], rely on applications to voluntarily yield the processor, thereby providing concurrency without the costs of preemption. However, it places the burden of managing concurrency explicitly on the

programmer and the inexact science of placing yielding points in code cause highly-variable inter-yield intervals. Hybrid models attempt to combine the advantages of both events and cooperative threading. Contiki, for example, is primarily event-driven i.e. processes are implemented as event handlers that run to completion and share a common stack; but it additionally supports multithreading capability as an optional library that applications can link to.

3 Pluggable Concurrency Models for MultiApplication Sensing

Due to divergent evolution of sensor node software and the multi-application coexistence, we believe the choice of single network-wide concurrency model to be sub-optimal. In particular, we recommend the adoption of different concurrency strategies to suit each sensor nodes' operating context and application requirements, and enabling switching between them as requirements change. To this end, we propose the following set of dynamically loadable context-dependent concurrency schemes for smart-city usage scenarios and provide implementations in Lorien, our reflective component-based WSN OS.

- Split-Phase FIFO Task Scheduling. This concurrency scheme is suitable for usage scenarios where single-application tasks run on nodes, are short-lived and do not require scheduling guarantees. As tasks are executed serially on the main thread, race conditions do not occur and programmers need not use synchronisation primitives. The MCU spends time either executing tasks and interrupt handlers or sleep duty cycling thereby consuming less power than if context-switching. Split-Phase FIFO Task Scheduling is preferable on dedicated single-application leaf nodes for example, in parking sensors nodes buried in the asphalt or nodes.

- Cooperative Fibers. When a sensor node becomes a data fusion point or is assigned more CPU-intensive data processing tasks such as Huffman compression of data samples or pattern matching, yield points can be introduced though cooperative fibers to allow the CPU to switch to other tasks. A fiber waiting for the radio hardware module to become available after a transmission can hence voluntarily yield to allow other tasks to execute. To ensure that suspended fibers are resumed, task *continuations* are captured in the form of execution context (including the stack pointer) and restored, using the setcontext family of functions: `getcontext`, `setcontext` and `swapcontext`.

- Preemptive Time-Sliced Multithreading. This concurrency model is suitable where task independence and prioritization are required for performance and responsiveness. As an example, repeater nodes also participating in environmental monitoring scenarios must prioritise packet forwarding over sensing tasks to avoid inducing network delays. When sudden high activity is encountered by a node that requires timely processing of tasks (during a flash flood or volcanic eruption), this concurrency scheme allows tasks to be scheduled for more frequent sensor data sampling, processing and transmission. For instance, to enable data collection when Disruption Tolerant Networking (DTN) gateway nodes on top of buses in SmartSantander come into contact opportunistically with isolated sensor node islands, data aggregation and transmission must be prioritized over other tasks as to reliably send data while the bus is still in radio range. Our preemptive thread scheduler for Lorien is priority-based and uses round robin semantics to ensure a fair distribution of CPU time slices. Binary and counting semaphores are supported to synchronise access to shared resources such as device drivers. When a thread blocks on a device driver,

it is moved from a ready-list to a semaphore list. It is removed when, upon receiving a hardware interrupt, the device-driver posts a semaphore to release the thread. Power-saving is handled by a thread called idle-task which is scheduled when no threads are active.

4 Work in Progress and Conclusions

To date, we have provided implementations for the aforementioned concurrency models; their relative suitability to various usage scenarios is under investigation. Further, we are currently exploring the use of proxy interfaces to make switching between concurrency models be non-invasive for applications. Also under investigation are the relative benefits of mixing cooperative and preemptive multithreading in the sensor node context. In this paper, based on the trends of divergent sensor node software evolution and on-node multi-application coexistence observed from a real smart city WSAN deployment we have proposed requirements for optimum exploitation of the infrastructure. In particular, switching between concurrency models depending on context is recommended and a number of dynamically loadable concurrency models proposed.

5 References

1. R. Shimizu, K. Tei, Y. Fukazawa, S. Honiden. Case studies on the development of wireless sensor network applications using multiple abstraction levels. In *SESENA 2012,* June 2012.
2. L. Sanchez, J.A. Galache, V. Gutierrez, J.M. Hernandez, J. Bernat, A. Gluhak, T. Garcia. SmartSantander: The meeting point between Future Internet research and experimentation and the smart cities. In Future Network & Mobile Summit (FutureNetw), 2011 , June 2011.
3. G. Coulson, B. Porter, I. Chatzigiannakis, and T. Baumgartner. Flexible experimentation in wireless sensor networks. Commun. ACM 55, 1 (January 2012), 82-90.
4. C. Duffy, U. Roedig, J. Herbert, C. Sreenan. An Experimental Comparison of Event Driven and Multi-Threaded Sensor Node Operating Systems. PerCom Workshops '07. March 2007.
5. A. Dunkels, B. Gronvall, T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. *Local Computer Networks, 2004.* Nov. 2004.
6. B. Porter, U. Roedig, G. Coulson. Type-safe updating for modular WSN software. *DCOSS 2011*, June 2011.
7. N. Weerasinghe, G. Coulson. Lightweight module isolation for sensor nodes. In *MobiVirt 2008*. ACM, New York, NY, USA.
8. P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. *SIGARCH Comput. Archit. News* 30, October 2002.
9. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGPLAN Not.* 35, 2000.
10. S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. *Mob. Netw. Appl.* 10, August 2005.
11. W. P. McCartney and N. Sridhar. Abstractions for safe concurrent programming in networked embedded systems. In Proc. of *SenSys'06*, 2006.
12. I. Leontiadis, C. Efstratiou, C. Mascolo, and J. Crowcroft.. SenShare: transforming sensor networks into multi-application sensing infrastructures. In Proc. *EWSN'12*.