Peloton: Coordinated Resource Management for Sensor Networks

Jason Waterman, Geoffrey Werner Challen, and Matt Welsh
School of Engineering and Applied Sciences, Harvard University
{waterman,challen,mdw}@eecs.harvard.edu

Abstract

This paper makes the case that operating system designs for sensor networks should focus on the coordination of resource management decisions across the network, rather than merely on individual nodes. We motivate this view by describing the challenges inherent to achieving a globally efficient use of sensor network resources, especially when the network is subject to unexpected variations in both load and resource availability. We present *Peloton*, a new distributed OS for sensor networks that provides mechanisms for representing distributed resource allocations, efficient state sharing across nodes, and decentralized management of network resources. We outline the Peloton OS architecture and present three sample use cases to illustrate its design.

1 Introduction

Operating system designs for sensor networks [5, 8, 11] have focused primarily on managing resources for individual nodes. However, a sensor network is not merely a collection of nodes operating independently: a sensor network must *coordinate* behavior across multiple nodes to achieve high efficiency and data fidelity. Unlike conventional distributed systems, nodes in a sensor network do not span multiple administrative domains, nor does the network typically support multiple applications with different users. Rather, it is natural to conceive of a sensor network as a single programmable entity that operates in a coordinated fashion to achieve some high-level system goal.

Managing limited resources is a key challenge in sensor networks. To achieve high efficiency, it is necessary to orchestrate resource management decisions across the network as a whole. For example, consider a network to monitor seismic activity at a volcano [18, 19]. Nodes must decide how much of their limited energy to invest in sampling, storing, and processing local sample data; transmitting signals to the base station; and listening for and routing packets for other nodes deeper in the routing tree. The resource load on each node is a complex function of the activity level of the volcano, quality of the sensor data, and packet forwarding demand from other nodes. This problem becomes more complex when nodes are powered by solar cells, since the energy budget fluctuates. It is important to note that both resource *load* and resource *availability* fluctuate over time: an offline static solution cannot suffice.

Coordinating resource management in a sensor network has received considerable attention [1, 7, 18]. However, *programming* these complex distributed behaviors is still done in an *ad hoc* fashion, using bare-bones APIs provided by the node-level OS. Subtle changes to node-level behavior (such as the radio listening duty cycle or choice of routing path) can have a tremendous impact on the overall efficiency and data yield of the network. Existing systems provide few tools to assist developers in designing correct and efficient solutions.

In this paper, we argue the operating system design for sensor networks should enable coordinated and global resource management while providing the appropriate abstractions and mechanisms to support whole-network optimizations. We argue that by providing a rich set of intra- and inter-node resource management interfaces that *expose* resource availability, *share* state across nodes, and *allocate* resources across multiple nodes, it is possible to develop more efficient applications with far less effort.

We propose *Peloton*,[1] a new distributed sensor OS based on three architectural components. The first, *vector tickets*, is a programming abstraction representing the right to consume resources across a set of nodes for performing some operation, such as routing data. The second, *state sharing*, provides mechanisms for nodes to share state on local resource availability and coordinate activities. The third, *distributed ticket agents*, permit resource management decisions to be decomposed across nodes, clusters, and the network as a whole.

In this paper, we make the case for Peloton, contrasting our approach to existing systems. We describe the Peloton OS architecture and describe through three use cases how it can be used to implement energy-efficient and coordinated mechanisms for cluster-based routing, adaptive sensor duty cycling, and optimized reliable data collection. We conclude with a discussion of future research directions.

2 The Case for Peloton

Designing a sensor network to make efficient use of scarce resources while yielding high-quality data

[1]In a road cycling race, the *peloton* is a pack of cyclists that ride closely together in order to collectively reduce wind drag and thereby save energy.

presents a number of challenges. Not only are node resources limited, but small local changes in a node's operation can have a ripple effect throughout the network. Moreover, nodes are mutually dependent upon each other to relay data, maintain time synchronization, perform collaborative event detection, and maintain spatial sensor coverage. It is not enough to conceive of a network as a mere collection of independent nodes, yet that is the dominant programming abstraction supported by existing sensor network operating systems [5, 8, 11].

As applications increase in complexity, reasoning about the global effects of local changes to node behavior can be difficult. The most common form of resource management is simple duty cycling, in which a (usually static) period is assigned to each node to achieve a given target lifetime. This works fine for applications with simple periodic behavior and few configuration changes over time. However, applications with more dynamic resource requirements need more sophisticated approaches, involving adaptation over time as well as both local and global knowledge of resource availability.

To achieve the greatest efficiency, nodes cannot simply make local decisions on how to invest their resources. Rather, it is necessary to perform resource adaptations in a coordinated fashion, where nodes share information on their local state and work together to assign tasks and allocate node-level resources to achieve the greatest common good. Such coordination can be done either within local clusters of nodes, or network-wide. While network-wide coordination has the potential for greater optimization, this must be traded off against the higher overhead for communicating demand and availability information to a centralized controller.

Existing sensor node OSs provide little support for collective resource management. TinyOS [8] and SOS [5] provide only low-level interfaces for managing the hardware state of the node. Pixie [11] and Eon [15] provide greater control over node-level resource availability and tuning, but still focus only on individual nodes. In SORA [12], nodes perform purely local, decentralized tuning of their actions, but without any explicit coordination across nodes.

The need for coordination in a sensor network to achieve good resource efficiency has been recognized in the literature [1, 12, 7, 18]. However, previous approaches have been *ad hoc* in nature, focusing on point cases of specific problems, such as routing, tracking, or sensor coverage. Most of the proposed algorithms have only been studied in simulations, and the few implementations would have required substantial effort to build, given only low-level messaging support provided by the OS. As a result, general-purpose abstractions for coordinated resource management have yet to emerge.

Similar resource management problems arise in many other distributed systems, including Internet-based services [13, 16], grids [3] and clusters [2, 14]. Sensor networks present new challenges in this space due to the vastly different workloads; severe constraints on resources; and the need for low-overhead coordination mechanisms. Our design of Peloton takes inspiration from these previous systems, but is tailored for this new domain.

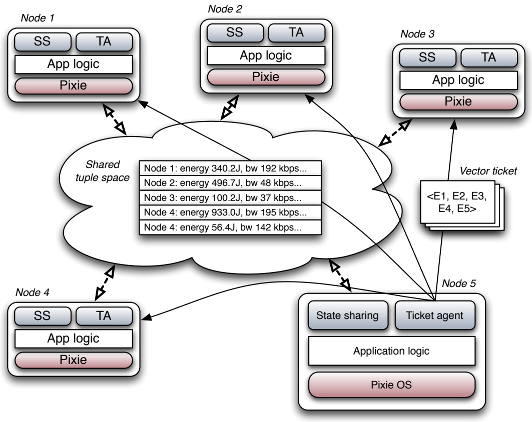

Figure 1: **A Peloton network consisting of five nodes.**

3 Peloton Architecture

Peloton builds upon our previous work on Pixie [11], a node-level OS for sensor nodes that focuses on enabling *resource aware programming*. In Pixie, application components request and receive *resource tickets* from the OS. A ticket represents a time-bounded right to consume some quantum of a given resource, such as energy, radio bandwidth, or memory. Tickets represent a flexible currency for resource management within the node and enable a rich space of policies for adapting sensor node operation to variations in load and resource availability. Pixie provides a resource allocator for each physical resource that estimates availability and responds to ticket allocation requests accordingly.

As an example, an application can tune the amount of data it attempts to transmit based on varying radio link conditions (e.g., due to interference or mobility) by requesting a bandwidth ticket in the desired amount. If the request cannot be satisfied under the current link conditions, a ticket with a smaller amount of bandwidth is issued, providing direct feedback on resource availability. Likewise, Pixie supports a range of energy scheduling policies to target a given battery lifetime by allocating energy tickets at a rate to conform to the schedule.

Resource tickets provide fine-grained visibility and control over resource usage. However, their use requires substantial application logic to request and manage tickets. For this reason, Pixie introduces a *resource broker* abstraction that mediates between applications and the underlying physical resources. Brokers serve as agents that apply policies such as prioritization, scheduling, and weighted fair queueing, thereby shielding application code from the details of managing tickets. For example, Pixie's bandwidth broker automatically manages ticket allocations across a set of application components to ensure that the highest-value data is transmitted when radio link conditions vary.

In Pixie (as well as similar systems, such as Eon [15]), all resource management decisions are made at the node level and there is no support for inter-node coordination. The key idea in Peloton is to extend the Pixie ticket abstraction to support *vector tickets* that represent a vector of resource requirements across a set of nodes for performing some desired operation. Likewise, Peloton introduces *distributed ticket agents* that perform commonly-used resource management policies in a coordinated way across nodes. The Peloton architecture is shown in Figure 1.

3.1 Vector Tickets

The key resource management abstraction in Peloton is the *vector ticket* (VT), which represents an allocation of resources across a set of nodes. A vector ticket $V = T_1, T_2, ...T_n$ consists of a vector of resource tickets T_i, each of which is a tuple $\langle n, R, c, t_e \rangle$. A ticket represents the time-bounded right to consume up to c units of resource R until expiry time t_e, at node n. An individual ticket can represent allocations of multiple resources. For example, transmitting a radio packet consumes both radio bandwidth and energy.

In Peloton, coordinated resource management is performed through the allocation and manipulation of vector tickets. As a simple example, performing a reliable download of a chunk of data from a given sensor node would require a vector ticket representing the energy and bandwidth consumed at the sensor node for reading data from flash and transmitting packets, as well as the energy and bandwidth used by intermediate nodes along the routing path to the base station. A single VT can capture the complete resource envelope of an operation spanning multiple nodes, providing a powerful mechanism for tracking and controlling resource allocations in a network-wide fashion.

VTs are allocated by *ticket agents* that track resource availability, possibly across a set of nodes, and distribute resource allocations to meet some desired policy. This allows resource allocations to be performed individually by nodes, collectively by a group of nodes, or globally by a base station. A VT is delivered to the nodes identified in the VT using an efficient local or global broadcast protocol such as Trickle [10]. In order to consume resources, a node must acquire a vector ticket, either locally (from the node's local ticket agent) or from a third-party agent, such as the base station or another node in the network.

As in Pixie, VTs are also used to track resource consumption, since all resource usage must be tied to a corresponding VT, as well as to provide feedback to applications in terms of resource availability. One strength of resource tickets is that they decouple resource allocation from usage; a ticket may be acquired at one time, and redeemed at a later time. Resource tickets are time-bounded as indicated by the expiry time t_e. This provides the ticket agent a measure of control over how many outstanding resource allocations have been granted. Tickets are not guarantees, only hints: a ticket may be revoked before its expiry time if conditions change.

VTs provide nodes autonomy in terms of *how* they consume resources allocated in the ticket. As an example, a global ticket agent might perform network-wide energy scheduling by tracking estimated energy usage across nodes, and doling out energy VTs to nodes in an attempt to meet a target battery lifetime. However, nodes are free to use the energy allocated in the VT as they see fit, such as for sampling data, relaying packets, or processing. This design strikes a balance between the high overhead required to enable fine-grain "micromanagement" of node operation versus complete autonomy without any coordination.

3.2 State Sharing

A critical requirement for coordination is the ability for nodes to efficiently share state. Peloton builds in mechanisms for node state sharing within neighborhoods, clusters, and across the network. Building upon previous systems [17, 20], Peloton provides a simple API whereby a node can publish local state into a shared tuple space, and read shared state from the tuple space. Nodes use this API to publish information on local resource availability (energy, bandwidth, memory, and storage capacity), which is used by ticket agents, described below, to implement resource allocation policies.

Peloton maintains a global but weakly-consistent view of the tuple space across all nodes. The update rate and freshness of data in the tuple space is a function of the topological distance between two nodes. For example, updates from direct radio neighbors are refreshed rapidly, while data from distance nodes is refreshed less often, and may represent a coarse time-windowed average of that node's state. A node can always request an direct update from another node to obtain its latest state if necessary.

This design provides both good efficiency (in terms of radio messages) and consistency within local neighborhoods. Delayed and aggregated state propagation to more distant nodes in the network is acceptable since consistency requirements typically diminish with topological distance; two nodes near each other in the network are more likely to require careful coordination. Of course, state sharing operations consume energy and bandwidth, and are accounted for using resource tickets.

3.3 Ticket Agents

Vector tickets and state sharing provide the underlying *mechanisms* to enable resource management in Peloton. Resource allocation *policies* are provided by *ticket agents*, which consume information published to the tuple space and allocate vector tickets accordingly. Peloton provides an extremely flexible model for ticket agents, which can be either centralized (e.g., at the base station) or decentralized (e.g., running on each node in the network). Decentralized ticket agents coordinate their decision-making using the tuple space to synchronize and share needed state.

Peloton's vector ticket model enables hierarchical resource allocation and delegation, since the node that allocates a vector ticket need not be the same as the nodes on which it is consumed. For example, nodes can use a leader election protocol [7] to form clusters, and nodes within each cluster delegate resource allocation authority to the ticket agent at the clusterhead. The clusterhead can receive frequent updates of node state and make fine-grained resource allocations locally. A central controller at the base station can coordinate cluster-level allocations by communicating only with clusterheads, allocating coarser-grained vector tickets for each cluster.

In this model, the base station acts as just another participant in the system, although it has substantially more computational horsepower and memory for tracking network-wide state. Moreover, Peloton naturally decomposes resource management authority across a multi-tiered network, such as one that contains both lightweight mote-class devices as well as embedded microservers [4].

Peloton generalizes the global resource allocation policies provided by Lance [18], which focuses on network-wide optimization of raw data collection in a sensor network. Peloton extends Lance to support a much more general form of global network control, both in terms of broadening the range of resources and actions that can be controlled, as well as distributing allocation authority into the network.

4 Application Vignettes

To highlight the benefits of the Peloton architecture, in this section we discuss three use cases that leverage the programming model.

4.1 Adaptive Cluster-Based Routing

A common approach to energy-efficient routing in sensor networks is to assign *clusterheads* within the network that take responsibility for receiving and forwarding packets for members of the cluster. Since clusterheads consume more energy than cluster members, it is necessary to rotate clusterheads periodically to balance energy load. In LEACH [7], the desired fraction of clusterheads is defined as a (static) design parameter. Nodes elect themselves clusterheads through a simple randomized leader-election protocol. Clusterheads define a TDMA communication schedule for each of the cluster members and collect, aggregate, and route aggregate data to the base station.

In LEACH, nodes pay no attention to their own energy reserves, nor that of other nodes in the network: it is assumed that the network will exhibit a uniform traffic load and thus remaining energy will be balanced across the nodes. However, this cannot accommodate varying traffic loads, link quality, and topology constraints caused by the spatial distribution of nodes. A better approach is to use information on available energy and radio link quality [6] to optimize clusterhead assignments.

Using Peloton, implementing an energy-aware variant of LEACH is relatively straightforward. Based on the energy consumption profile of nodes within a local neighborhood, each node can determine the probability with which it will elect itself as a clusterhead. Each clusterhead becomes the (temporary) ticket agent for the cluster members, assigning vector tickets for bandwidth and energy use to manage the cluster's communication schedule and resource consumption envelope.

This simple example highlights the value of an OS structure that makes distributed resource visibility and allocations explicit. Peloton cleanly separates the underlying details of resource management from the higher-level coordination logic specific to the routing protocol. Furthermore, Peloton's underlying abstractions make it easy to extend the protocol to consider tradeoffs such as the impact of forwarding load on nodes with poor radio links and variable traffic generation rates.

4.2 Adaptive Sensor Duty Cycling

Sensor duty-cycling is another common energy saving technique, but it must be implemented with care as the specific schedule affects data fidelity, network connectivity, and sensor coverage. Thus, there is a tension between lowering resource costs and overall value of data produced by the network. In many cases it is difficult or impossible to determine an appropriate duty-cycling schedule statically.

One example of a dynamic duty-cycling protocol is RACP [9], which tunes the sleep and wake cycles of individual nodes to maintain adequate sensor coverage while considering variations in nodes' energy availability. In RACP, a node can nominate itself as a *head* node, which sleeps until a predetermined wakeup interval. This requires that other nearby nodes act as *sponsors*, staying awake to maintain local sensor coverage at the cost of increased energy expenditure.

The RACP scheme maps nicely onto the Peloton primitives. Through local state sharing, nodes learn of each other's energy availability and spatial coverage. Entering a sleep state requires that a node allocate a VT to cover the energy requirements for the sponsor nodes that will stay awake on its behalf. Peloton's role is to decouple the shared mechanisms needed to coordinate resource management, while the specific policies are provided by the RACP ticket agent running on each node.

4.3 Energy-Efficient Data Collection

Another application enabled by Peloton is the collecting of high-data-rate signals from a sensor network. Performing a reliable transfer of a sample stream stored on a sensor node, which might consist of kilobytes of data, requires substantial bandwidth and energy resources. Our previous work on Lance [18] demonstrated the ability to optimize the overall utility of data collected from a sensor network while adhering to a node-level energy schedule. However, Lance relies on a centralized controller, running at the base station, that has global knowledge of the data stored and the energy profile of each node. This requires potentially high overhead for scheduling data transfers and introduces a single point of failure.

The use of Peloton's vector tickets and shared state abstraction opens up the possibility of decentralizing this process. One approach is to induce hierarchical control by electing clusterheads in the network in a manner similar to LEACH. The clusterhead would maintain a consistent view of the data stored by nodes within its neighborhood and perform a local optimization to determine which signals should be allocated energy and bandwidth resources. The clusterhead acts as the local ticket agent, managing vector tickets for each signal download operation. Once signals are aggregated at the clusterhead, multiple clusterheads can coordinate to schedule the transfer of the highest-utility signals to the base station. This two-tiered approach reduces communication overhead through local decision making within each cluster, and allows clusterheads to more accurately estimate relative data utilities across a range of signals.

These three examples are intended to be simple and intuitive, though they only scratch the surface of the kinds of coordination schemes that Peloton is intended to support. Peloton's design captures commonly-used mechanisms for managing sensor network resources. It is important to keep in mind that Peloton does not eliminate the need for careful policy design. Our hope is that by starting with the right abstractions, this will be much easier to do than it has been in previous systems.

5 Discussion and Conclusions

We believe that thinking of a sensor network as a coordinated ensemble, rather than simply as a collection of individual nodes, is necessary to achieve efficient allocation of scarce network resources. The need for coordination, both explicit and implicit, requires a rethinking of the OS architecture for sensor networks, which to date has been focused on managing node-level resources alone. We have described the Peloton OS architecture, which provides three essential mechanisms to enable coordinated resource management: vector tickets, state sharing, and distributed ticket agents. Through three canonical use cases, we have confidence that Peloton is an effective approach to structuring complex in-network resource management decisions.

We are currently implementing Peloton on top of the Pixie [11] node-level OS. Pixie provides much of the node-level machinery needed to track and allocate node resources. The key challenge is enabling efficient sharing within the network, as well as decomposing resource-management decisions across ticket agents within the network. We believe that Peloton will make it easier to develop collaborative applications that react to varying node state and resource conditions. We plan to deploy Peloton as part of a volcano-monitoring sensor network that will perform in-network processing of seismic data.

References

[1] N. Burri, P. von Rickenbach, and R. Wattenhofer. Dozer: ultra-low power data gathering in sensor networks. In *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, pages 450–459, New York, NY, USA, 2007. ACM.

[2] D. G. Feitelson and L. Rudolph. Gang scheduling performance benefits for fine-grain synchronization. *Journal of Parallel and Distributed Computing*, 16:306–318, 1992.

[3] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing*, number 3779 in LNCS, pages 2–13, 2005.

[4] O. Gnawali, B. Greenstein, K.-Y. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler. The TENET Architecture for Tiered Sensor Networks. In *Proc. ACM Conference on Embedded Networked Sensor Systems (Sensys)*, Boulder, CO, November 2006.

[5] C.-C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. SOS: A dynamic operating system for sensor networks. In *Proc. Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*, 2005.

[6] M. J. Handy, M. Haase, and D. Timmermann. Low energy adaptive clustering hierarchy with deterministic cluster-head selection. In *MWCN '02: 4th International Workshop on Mobile and Wireless Communications Network*, pages 368–372, Stockholm, Sweden, 2002.

[7] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proc. the 33rd Hawaii International Conference on System Sciences (HICSS)*, January 2000.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Proc. the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93–104, Boston, MA, USA, Nov. 2000.

[9] C.-f. Hsin and M. Liu. Network coverage using low duty-cycled sensors: random & coordinated sleep algorithms. In *IPSN '04: Proceedings of the 3rd international symposium on Information processing in sensor networks*, pages 433–442, New York, NY, USA, 2004. ACM.

[10] P. Levis, N. Patel, S. Shenker, and D. Culler. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, 2004.

[11] K. Lorincz, B. rong Chen, J. Waterman, G. Werner-Allen, and M. Welsh. Resource aware programming in the pixie os. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 211–224, 2008.

[12] G. Mainland, D. C. Parkes, and M. Welsh. Decentralized, adaptive resource allocation for sensor networks. In *NSDI '05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 315–328, 2005.

[13] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Design and implementation tradeoffs for wide-area resource discovery. In *HPDC '05: Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*, pages 113–124, Washington, DC, USA, 2005. IEEE Computer Society.

[14] V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, and E. Nahum. Locality-aware request distribution in cluster-based network servers. In *ASPLOS-VIII: Proceedings of the eighth international conference on Architectural support for programming languages and operating systems*, pages 205–216, New York, NY, USA, 1998. ACM.

[15] J. Sorber, A. Kostadinov, M. Brennan, M. Garber, M. Corner, and E. D. Berger. Eon: A Language and Runtime System for Perpetual Systems. In *Proc. ACM SenSys*, November 2007.

[16] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in Mariposa. In *Proc. the 3rd International Conference on Parallel and Distributed Information Systems*, September 1994.

[17] M. Welsh and G. Mainland. Programming sensor networks using abstract regions. In *Proc. the First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI '04)*, March 2004.

[18] G. Werner-Allen, S. Dawson-Haggerty, and M. Welsh. Lance: optimizing high-resolution signal collection in wireless sensor networks. In *SenSys '08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, pages 169–182, 2008.

[19] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and yield in a volcano monitoring sensor network. In *Proc. 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2006)*, Seattle, WA, November 2006.

[20] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A neighborhood abstraction for sensor networks. In *Proc. the International Conference on Mobile Systems, Applications, and Services (MOBISYS '04)*, June 2004.