PRO 06 Tenet Architecture for Tiered Embedded Networks

PRO 06.1 Overview
This project revisits the architectural foundations of the sensor network systems built and deployed by CENS.

Sensor network researchers originally envisioned large networks of tiny wireless nodes with simple gateways to the Internet. Because of energy and network constraints, the tiny nodes themselves would collaboratively process data in-network in complex, application-specific ways. In reality, though, the sensor networks we deploy mostly perform continuous data acquisition, and incorporate little or no on-mote multi-node data fusion. We believe that two factors can explain this development. First, the introduction of masters: 32-bit CPU-class nodes for which power can be engineered. Masters are an integral part of every network we deploy, and rather than acting as mere Internet gateways, they participate in the functionality of the network. (Yet our current architectural principles take little advantage of them!) Second, the unexpectedly high complexity of mote-based multi-node data fusion makes implementing such functionality a bad tradeoff. By not optimizing for the system as a whole, we are missing the opportunity/have overlooked to build a software architecture that promotes on-board mote processing of its local time series in an adaptive and efficient manner.

Tiered data-collection networks are therefore here to stay. Unfortunately, only the original architectural principles are available to sensor network designers. When designers follow those principles the resulting systems are fragile and overly complex. Even worse, they are difficult to repurpose; and the master nodes on which network health depends remain underutilized.

An architecture is needed to guide the construction of scalable, evolvable, and replicable sensor systems that will serve the vast array of applications currently awaiting deployment. The Tenet project is developing such an architecture.

PRO 06.2 Approach
Many current large-scale sensor network deployments are tiered. The lower-tier, composed of motes, contains sensing and actuation functionality and enables infrastructure-less instrumentation of physical spaces and artifacts. The upper-tier, consisting of 32-bit nodes, masters, is free of energy constraints and provides increased network and computational capacity, enabling large-scale deployments.

The Tenet architecture prescribes a functional separation between motes and masters, with the goal of reducing overall system complexity. The architecture asserts that it is still desirable for the motes, which should be optimized for low power operation, to do local aggregation, compression, and even filtering of its time series data. However, cross node aggregation, filtering, and processing is best done by the master.

Fundamentally, Tenet constrains collaborative multi-node in-network processing to be performed on the master nodes. In-network aggregation and fusion is inherently somewhat centralized in that data from multiple nodes is sent to a common node to be processed. The masters in a tiered architecture are natural fusion and aggregation points from the perspective of capacity (CPU, storage, bandwidth and energy). Constraining aggregation to be performed at masters results in a simpler architecture relative to one that allows aggregation on topologically convenient, but resource-constrained motes. In Tenet, motes are tasked by applications running on masters, and can implement simple logical elements such as thresholds and compression, but any further computation takes place only on masters. Finally, masters can collaborate with one another to implement distributed applications for tracking, detecting spatio-temporal events, or (as in this proposal) multi-robot coordination (Figure 1).

Figure 1. A Tiered Sensor Network

PRO 06.3 System(s) Description and/or Experiments
The current Tenet system consists of several components: a tasking library which supports the composition and execution of small programs called tasks; a routing subsystem which uses a multi-sink version of a standard tree routing protocol for routing data from motes to masters; a task dissemination subsystem which ensures the reliable delivery of tasks from any master to all the motes, a transport subsystem which provides end-to-end reliable transmission of sensor data from motes to masters, and a time synchronization component which ensures that all the motes maintain a globally synchronized time.
Recently we have designed and implemented two versions of Tenet using the threads primitive, called TOSThreads, that has recently become available in TinyOS. Using threads allow Tenet to be robust even in applications with long running CPU intensive tasks. Tenet-T is a reimplementation of Tenet using the TOSThreads library that spawns one thread to service each Tenet task and replaces Tenet's original task scheduler with the TOSThreads thread scheduler. The main difference between Tenet-T and Tenet is that TOSThreads supports preemption. This allows each thread running a Tenet task to execute its tasklets one after another without regard for how long each of them might take. In the original Tenet, the Tenet task scheduler has to interleave tasklets from multiple tasks in order to keep them all responsive; individual tasklets run to completion and cannot perform long running computations. Tenet-T removes this limitation at the expense of a small increase in code size. Tenet-C is a reimplementation of Tenet that significantly increases the expressivity of the tasking language, yet does not require drastic modifications to the overall system. In Tenet-C, the user writes a C program, instead of a data-flow task description, and compiles it into a dynamically loadable binary object. The Tenet-C kernel is identical to that of Tenet-T except that it dynamically link and load application binaries. However, Tenet-C’s API is significantly smaller because the functionality provided by many of the Tenet tasklets are already provided natively by C. Figure 2 provides a pictorial overview of these modifications. We have validated through experiments that our design correctly support timing-sensitive applications in the presence of long-running computations.

**PRO 06.4 Accomplishments**

**Software:**
We released the threaded version of Tenet (Threaded-Tenet) implemented on top of the TOSThreads and TinyLD library in TinyOS 2.1 software in June 2009. The instruction to download and use the software can be found at http://enl.usc.edu/software.html.

We have also updated Tenet 2.0 release to Tenet 2.1. This release includes robust version of many experimental features we developed in the past such as RCRT congestion control protocol.

**Deployment:**
In the past, we have implemented and deployed Pursuit Evasion Game and seismic monitoring network on a suspension bridge. Most recently, we have extended Pursuit Evasion Game into a multi-agent pursuit Evasion Game in which multiple robotic pursuers collectively determine the location of multiple evaders, and try to corral them.

**PRO 06.5 Future Directions**
Our work for next year will be focused on deployment of Tenet as well as evolution of the architecture and its components to allow Tenet to be used in a variety of settings.

We plan to deploy Tenet for seismic sensing of buildings using the imote-2 based platform. We are currently porting Tenet to this platform and designing the software to manage and coordinate the deployments.

To enable energy-efficient deployments of Tenet, we designed AEM to duty-cycle the radio. Although we have experimented with AEM in the in-door laboratory setting, we have not yet deployed Tenet with AEM in the field. We plan to take that step and use the lessons learnt to further improve the design of AEM.