Building Verifiable Sensing Applications Through Temporal Logic Specification

Asad Awan, Ahmed Sameh, Suresh Jagannathan, and Ananth Grama

Department of Computer Sciences, Purdue University, W. Lafayette, IN 47907

Abstract. Sensing is at the core of virtually every DDDAS application. Sensing applications typically involve distributed communication and coordination over large self-organized networks of heterogeneous devices with severe resource constraints. As a consequence, developers must explicitly deal with low-level details, making programming time-consuming and error-prone. To reduce this burden, current sensor network programming languages espouse a model that relies on packaged reusable components to implement relevant pieces of a distributed communication infrastructure. Unfortunately, programmers are often forced to understand the mechanisms used by these implementations in order to optimize resource utilization and performance, and to ensure application requirements are met. To address these issues, we propose a novel and high-level programming model that directly exposes control over sensor network behavior using temporal logic specifications, in conjunction with a set of system state abstractions to specify, generate, and automatically validate resource and communication behavior for sensor network applications. TLA+ (the temporal logic of actions) is used as the underlying specification language to express global state abstractions as well as user invariants. We develop a synthesis engine that utilizes TLC (a temporal logic model-checker) to generate detailed actions so that user-provided behavioral properties can be satisfied, guaranteeing program correctness. The synthesis engine generates specifications in TLA+, which are compiled down to sensor node primitive actions. We illustrate our model using a detailed experimental evaluation on our structural sensing and control testbed. The proposed framework is integrated into the COSMOS macroprogramming environment, which is extensively used to develop sensing and control applications at the Bowen Lab for Structural Engineering at Purdue.

1 Introduction

Sensor networks are integral to most DDDAS applications. They are often composed of large numbers of low-cost motes with wireless connectivity and varying sensing capabilities. Motes are characterized by limited resources including memory, CPU, network capacity, and energy budget. As DDDAS applications become widespread, there is increasing realization of the complexity associated with building robust sensor network applications. Much of this complexity stems from the need to implement reliable distributed coordination in dynamic environments, under severe performance and resource constraints. Driven by these underlying constraints, sensor network programming often involves low-level system details and communication mechanisms. Consequently, even simple applications pose challenging implementation problems, motivating high-level

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 1205–1212, 2007.

programming models and abstractions. With this overarching goal, we have developed a comprehensive sensor macroprogramming environment, COSMOS, that supports programmability, device heterogeneity, scalable performance, and robustness. Our macroprogramming environment integrates node and network operating system kernels with a language and compilation infrastructure that has been validated on a range of applications with varying performance requirements.

A critical component of our macroprogramming environment is foundational support for automatic synthesis and verification of distributed coordination mechanisms from formal specifications. While conventional programming paradigms leverage high-level abstractions for defining communication and synchronization protocols, these abstractions often do not provide mechanisms for enforcing requirements that maximize resource utilization and flexibility. In contrast, we present a high-level programming model for sensor network programming in which program behavior is expressed as invariants that directly capture performance and resource constraints. To this end, the paper makes two basic contributions – (i) it demonstrates the use of temporal logic for defining specifications that capture a large class of behaviors that arise in sensor programs; and (ii) it presents techniques for automatically generating efficient communication and other coordination actions from these specifications.

Recent work in sensor networks has resulted in a large number of communication protocol implementations available as reusable components [2]. Use of these components requires the programmer to interface local data sensing and processing at the nodes with available communication library components. These component implementations represent varying domain specific optimizations and protocol designs. For example, tree routing based aggregation, as used in TinyDB [6], allows collection of network data at the root node, while ring-gradient based aggregation, used in Synopsis Diffusion [7] supports similar functionality, but provides significantly enhanced resilience through the use of multi-path routing. This comes at the cost of possible data duplication and out-of-order data delivery. On the other hand, the Trickle [5] protocol provides dissemination of data from a single node to all the nodes in the network, which represents complementary functionality. Interestingly, a single application may require both aggregation/collection and dissemination on different distributed dataflow paths with varying constraints on robustness, in-ordered delivery, and latency.

Conventional sensor network application development, consequently, requires a developer to select and compose coordination mechanisms from available primitives to satisfy application specifications. For the gradient versus tree routing example above, factors that impact selection include (i) acceptable resource overhead to achieve the required level of resilience, and (ii) whether the aggregation functions are duplicate and order insensitive. The first consideration represents a tradeoff of resource consumption and resilience, while the second factor deals with cross-component conflicts, specified in terms of requirements on distributed dataflow semantics. Due to application needs and resource constraints, the developer is forced to account for low-level implementation details of communication mechanisms–making program development time and effort intensive. This paper addresses the problem by providing powerful formalisms and automated techniques for generating the necessary implementations that are guaranteed to satisfy user constraints.

2 Overview of Proposed Approach

We target three key aspects of sensor network programming – correctness, programmability, and performance. We do this by enabling the user to provide a high-level specification of aggregate system behavior for the application, and provide a program synthesis and compilation infrastructure to automatically generate efficient executables from these specifications. Specifications are expressed in temporal logic; formulae in this logic define invariants over system state. For example, the formula ($\Box$ *RoutingTableSize* $<$ *150Bytes*) expresses a constraint on a global system abstraction (*RoutingTableSize*) that must hold over all state transitions. Notably, this specification provides a separation of concerns: any routing implementation that meets this constraint is feasible under this invariant. We support the following abstractions:

(A1) **Member node groups abstractions.** These abstractions allow description of data source, forwarding and processing, and sink nodes that participate in distributed coordination. For example, to express the fact that all nodes are data sources, and that there exists a unique root in the network, we specify:

$$\begin{aligned}\Box((\forall s \in \mathit{Senders}) \Rightarrow\ & (s \in \mathit{Nodes}) \\ & \wedge(\exists r \in \mathit{Receivers}.r \in \mathit{Root}) \\ & \wedge(|\,\mathit{Root}\,| = 1))\end{aligned} \tag{1}$$

Here, *Senders* defines a set of data source nodes, *Receivers* defines a set of sinks, and *Root* defines a singleton set containing a distinguished receiver.

(A2) **Routing resource consumption abstractions.** These allow specification of control requirements on the overhead and performance of data routing and hop-by-hop forwarding. For example, to express the fact that there is no routing redundancy, we specify:

$$\begin{aligned}&\Box(\forall s, r_1, r_2 \in \mathit{Nodes}. \\ &\quad \mathit{Comm}(s, r_1) \wedge \mathit{Comm}(s, r_2) \Rightarrow r_1 = r_2)\end{aligned} \tag{2}$$

Comm defines a binary relation on nodes that expresses a communication relation among senders and receivers in the network.

(A3) **Link abstractions.** These abstractions provide control over properties of communication links (e.g., radio energy consumption, packet batching for congestion control, etc.). For example, the following formula expresses the constraint that the radio power on a mote should be turned off if the size of its input buffer is less than some threshold, and that the power should eventually be turned on if the buffer is non-empty (because of data inserted from the sensor end). Furthermore, the maximum size of the buffer should never exceed a specified constant (`MaxLen`). Observe there are several implementations that could satisfy this specification; one might resume power as soon as the threshold level is exceeded, another might periodically service the buffer, but may choose to drop packets to enforce the invariant on buffer size.

$$\begin{aligned}&\Box(\forall n \in \mathit{Nodes}. \\ &\quad (|\,\mathit{Buffer}(n)\,| < \mathit{Threshold} \Rightarrow \mathit{Radio}(n) = \mathsf{down}) \\ &\quad \wedge(|\,\mathit{Buffer}(n)\,| > 0 \Rightarrow \Diamond \mathit{Radio}(n) = \mathsf{up}) \\ &\quad \wedge(|\,\mathit{Buffer}(n)\,| < \mathit{MaxLen}))\end{aligned} \tag{3}$$

(A4) **Data management buffers.** These allow specification of properties relating to distributed data coordination and processing, and the control of associated resources (such as memory, service delays, etc.). For example, the following formula captures a time-ordered precedence among data received by nodes. Specifically, it obligates a data packet p_1 with an earlier timestamp than data packet p_2 (as defined by the *Epoch* function) to be either serviced or evicted before p_2. (We use $a \rightsquigarrow b$ to denote $a \Rightarrow \Diamond b$.)

$$\begin{aligned} \Box(\forall p_1, p_2 \in \mathit{Data}.\forall n \in \mathit{Nodes}.p_1, p_2 \in \mathit{Buffer}(n) \\ \mathit{Epoch}(p_1) < \mathit{Epoch}(p_2) \rightsquigarrow \\ (p_1 \notin \mathit{Buffer}(n) \wedge p_2 \in \mathit{Buffer}(n))) \end{aligned} \tag{4}$$

Observe that these specifications capture salient aspects of system behavior with respect to application requirements without fixing a specific set of implementation choices. Indeed, in practice, some of the specifications above may possibly have several implementations. We derive a feasible implementation from the composition of *all* provided specifications. This is complicated by the fact that an implementation choice made in response to one specification may conflict with other specifications. For example, we could satisfy the specification in Equation 1 using either a tree or gradient routing implementation. In the case of gradient routing, though, packet duplication conflicts with the invariant in Equation 2.

We automatically synthesize a valid implementation that satisfies user-defined specifications, based on invariants provided by library components. Unfortunately, not all user-provided specifications are directly supported by implementations. For example, the specification in Equation 4 imposes a precedence relation on received data. However, no library implementations in existing sensor network systems actually provide such functionality because data management is often tightly coupled to application semantics. Thus, mapping this specification to an implementation requires us to inject additional details.

The user does not specify how the given invariants and properties are met. The actions required to meet the user specified invariants are automatically generated by a *synthesis engine*. Since user specifications may be incomplete, it is possible that not all desired invariants can be satisfied based on user-provided specifications. Invariant violations that arise because of incomplete specifications are reported by the TLC model checker. Using these violations the synthesis engine generates refined specifications based on knowledge of the abstractions exposed by the system. The synthesis process terminates when a provably correct (i.e., model checked) specification is achieved. Finally, the automatically-generated specifications (`MCAutoGen`) together with the interface description provided by the user module are compiled to node primitives. The compiler infrastructure uses a combination of two approaches to generate native code. First, based on the user constraints over system state variables, it locates components from an annotated library that provide the required mechanisms to meet user requirements. Inter-component dependencies and conflicts are automatically resolved. Second, based on the detailed actions generated by the synthesis engine it automatically generates node primitives from TLA+ specifications.

Table 1. Invariants for each of the distributed coordination tasks of the seismic sensing application

Sys Abs	MaxFlow	HiResFlow	CtrlFlow	Total
Member	RcvrID=Root		SndrID=Root	2
Routing	FwdLinks=1, RtableSize≤150B MaintenanceInterval=10000ms		FwdLinks=Bcast RtableSize≤10B	5
Link	-		-	0
MsgType	[TimeSeq]	[NodeID, SeqNum]	[]	3
In-Net Buf	MLenInv, MOrderInv, MAggregateMax	HLenInv, HOrderInv	-	5
Rcv Buf	MLenInv, MOrderInv, MAggregateMax	HLenInv, HFlushInv, HOrderInv	-	6
Timer	MFBServiceRate, MRBServiceRate,	HFBServiceRate, HRBServiceRate,	CFBServiceRate, CRBServiceRate,	6

3 Evaluation

Our application testbed is a seismic-sensing application, currently operational at the Bowen Lab for monitoring a three story concrete structure. The application uses tens of nodes, to sense acceleration (using on-board accelerometer sensors) and displacement. Seismic activity generates high resolution data from the network, which is used to extract and study the frequency response. Our implementation of this application is as follows: time-windowed max acceleration value is extracted from the sensor network using in-network aggregation. This data is fed to a user provided controller function, which detects seismic activity and disseminates a trigger message to all nodes. The controller function runs on a unique node (root node) in the network. In response to the trigger, all nodes generate high-resolution data, which is forwarded to the root node, where a user provided FFT function evaluates the frequency response of the structure.

The application specification includes three distributed communication and coordination tasks: (i) `MaxFlow`, which involves collection and in-network aggregation of max sensor readings, (ii) `HiResFlow`, which involves collection of high resolution sensor readings, and (iii) `CtrlFlow`, which involves dissemination of controller trigger messages to all nodes in the network. The invariants on state abstractions for each of these flows are shown in Table 1. Since the max values and high resolution data flows from all nodes in the network to the root node, the user can share the system abstractions including member, routing, and link abstractions. On the other hand, `CtrlFlow` involves data dissemination from the root to all nodes and needs to be independently controlled. For `MaxFlow` and `HiResFlow` the application specifies that forwarding should be duplicate sensitive, and provides routing table size and topology maintenance resource constraints. On the other hand, for dissemination, the application requires that duplicates are necessary and that routing memory resource consumption should be low. For `CtrlFlow` the user does not require any constraint on buffering, except that its service rate should be high (`CXXServiceRate`). The user specifies buffer length and packet ordering for both the buffers of `MaxFlow and \verb`HiResFlow+. In addition, for `MaxFlow` in-network aggregation is specified. For `HiResFlow` the application

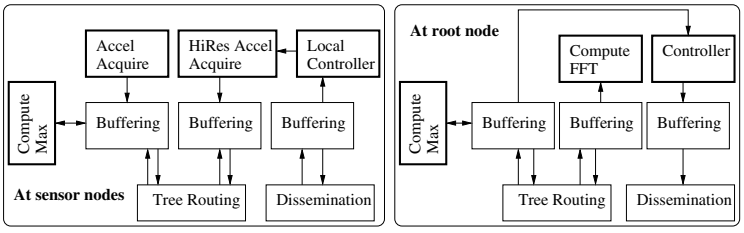

Fig. 1. Interaction of components in the synthesized seismic sensing application. Separate program fragments are generated for the sensor nodes and the root node. User functions are represented by thick border boxes.

specifies (using `HFlushInv`) that sequence numbers for data from each node should be consecutive (because several consecutive packets are required for FFT to generate a reasonable frequency analysis). For the entire application only 27 invariants are used, in all. The interactions of the components of the generated program are illustrated in Figure 1. The user provides only six functions, which implement data acquisition and processing.

Performance. We evaluate the performance of the synthesized program on a 10 node (plus one root), wireless sensor network testbed. We use Mica2 motes with MTS310 multi sensor boards [1] as sensor nodes. Mica2 has an ATmega128 micro-controller running at 7.37MHz, with a 128KB program ROM, 4KB data RAM, and Chipcon CC1000 radio (with a maximum through put of 12.3kbps). The MTS310 board supports a wide variety of sensors including a 2-D accelerometer, which is used in our experiments. The root node is a workstation with Intel Pentium 4, 1.70GHz and 512MB RAM, running Linux 2.6.17.7 kernel. The root is interfaced to the Mica2 radio network using a MIB510 [1] board connected to the serial port.

The key performance concerns in our application include (i) reliability of network communication, and (ii) congestion control, due to the heavy traffic when high resolution data is communicated to the root node. To evaluate the ability of our programming

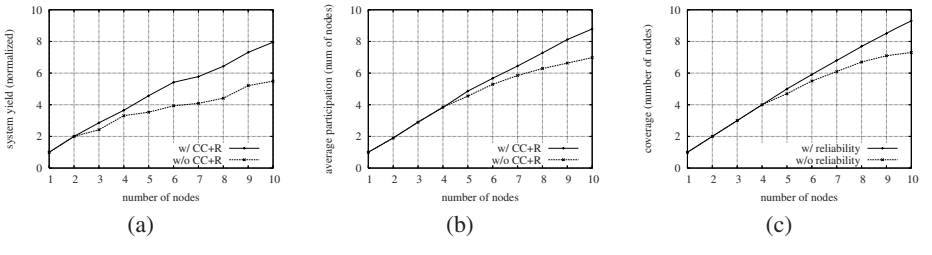

Fig. 2. Congestion control and communication reliability components are inserted into the synthesized application due to additional user invariants. The plots show that the use of these components increases system performance substantially.

model to meet user requirements we execute the application with and without the following invariants:

$$Inv \triangleq \wedge MH\_link.ReliabilityOvh \leq 1 \quad (5)$$
$$MH\_link.CongestionBatching \leq 4$$
$$Ctrl_i ink.ReliabilityOvh \leq 1$$

These invariants imply that the communication of max values and high resolution data (`MH_XX`) should use reliability, with a maximum overhead of one packet, and congestion triggered buffering of upto three packets. Similarly, reliability, with a maximum overhead of one packet, should be provided for controller messages. The performance results, for varying network size, for the synthesized application, with and without the above invariants, are shown in Figure 2. The evaluation includes the following. (a) System yield for high resolution data (normalized to the yield of a single node), where yield measures the amount of useful data received and processed at the root node. (b) Average number of nodes whose data was used in each max value calculation. Note that one max value is generated per epoch. (c) Coverage, in terms of the number of nodes that received the dissemination trigger message in the first refresh interval. Note that the dissemination protocol uses periodic refreshes to account for network dynamics. However, the response time increases with the number of nodes that receive the trigger in the first refresh interval.

The plots in Figure 2 show that by using congestion control and reliable forwarding, approximately 40% more yield of high resolution data, and 30% higher participation in max aggregation is achieved, when the network has 10 nodes. Similarly, using reliability for trigger dissemination achieves about 14% higher coverage in a network of 10 nodes. These results show that using simple high-level invariants, the user can substantially affect system performance without having to deal with low-level programming details. Furthermore, the invariants allow constraining the resource overheads involved in achieving desired performance.

3.1 Specifying Resource-Quality Tradeoffs

Most sensor network applications utilize in-network data processing. Nodes in a wireless sensor network dynamically form self-organized data routing topologies. Hence, the number of nodes that feed data to a forwarding node, where processing takes place, varies dynamically. Given limited buffer memory, applications should adaptively manage processing to cater to surges in input data rate or risk dropping messages due to overflowing buffers. However, developing such robust applications is difficult. Our language enables users to provide multiple data processing functions, which are selectively triggered based on buffer state, thus, allowing load conditioning. The user provides processing intensive functions (e.g., input signal convolution, FFT, or statistical analysis of input data), and their less processing-intensive counterparts that yield lower quality approximations. Following is a simple load conditioning invariant example.

$$LoadCondInv \triangleq \text{IF } Len(fb) < Threshold \quad (6)$$
$$\text{THEN } UseFunc(\text{“DataFFT”}, fb, fb[End])$$
$$\text{ELSE } UseFunc(\text{“AproxFFT”}, fb, fb[End])$$

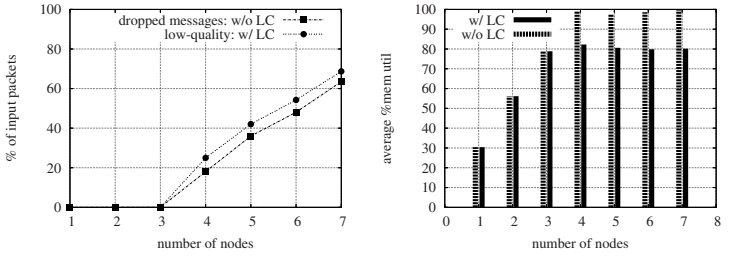

Fig. 3. Memory vs quality tradeoff. Load conditioning prevents lost data by using a faster computation, which yields approximate results, to maintain memory threshold.

Driven by resource limitations, users tradeoff quality with resource consumption. The plots in Figure 3 illustrates an evaluation of a system with and without the above invariant. Without load conditioning, input data is dropped when the input rate increases (as the number of data source nodes increase). With load conditioning the memory threshold is maintained, while high load conditions trigger low quality approximations. For example, with data from 4 sources, approximately 25% of output data is low quality while 75% is high quality.

4 Concluding Remarks

In this paper, we presented a powerful and novel formalism for development of verifiable sensing applications based on temporal logic specifications. We also present an automatic synthesis engine, that uses a model checker, TLC, along with low-level system abstractions, to generate code guaranteed to satisfy programmer specified invariants.

References

1. Crossbow Inc. `http://www.xbow.com/wireless_home.aspx`.
2. Cheng Tien Ee, Rodrigo Fonseca, Sukun Kim, Daekyeong Moon, Arsalan Tavakoli, David Culler, Scott Shenker, and Ion Stoica. A modular network layer for sensornets. In *Proc. OSDI'06*, November 2006.
3. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer Pister. System architecture directions for networked sensors. In *Proc. of ASPLOS-IX*, November 2000.
4. Leslie Lamport. *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. Pearson Education, Inc, 2002.
5. Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *Proc. of NSDI '04*, March 2004.
6. Samuel Madden, Michael Franklin, Joseph Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, March 2005.
7. S. Nath, P. B. Gibbons, S. Seshan, and Z. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proc. of SenSys '04*, November 2004.