THE SNAP! TOOLKIT FOR DEVELOPING SENSOR NETWORKS AND APPLICATION TO XC SKIING

BY

MATTHEW B. DEBSKI

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degrees of Bachelor of Science in Electrical Engineering and Computer Science and Master of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2000

© Massachusetts Institute of Technology 2000. All rights reserved.
THE SNAP! TOOLKIT FOR DEVELOPING SENSOR NETWORKS
AND APPLICATION TO XC SKIING

BY
MATTHEW B. DEBSKI

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degrees of
Bachelor of Science in Electrical Engineering and Computer Science
and
Master of Engineering in Electrical Engineering and Computer Science
May 18, 2000

ABSTRACT

Athletes, health professionals, animal specialists, meteorologists, and other investigators increasingly
employ small, human-scale sensor networks in their disciplines. The sensor networks used by each
field and study have unique requirements.

Regardless, these networks all collect and store data, and often display or transmit them. The Snap!
toolkit takes advantage of the similarities among sensor networks to make prototyping them fast and
easy. Simultaneously, it acknowledges their inherent differences, remaining flexible in order to
accommodate the needs of different systems.

Snap! specifies the behavior of the nodes on a sensor network and describes the logical structure of
each node and of the network as a whole. The Snap! design places emphasis on the reusability of
hardware and software, rather than on the creation of an optimal solution for a given application.

One Snap! platform and system has thus far been constructed and tested: a cross-country ski system
to gather information on the environment and from the skis and skier. The data collected from the
ski system make it possible to quantitatively describe the skier's stride and the effects of a ski's design.

Thesis Supervisor: Michael J. Hawley
Title: Assistant Professor of Media Arts and Sciences
Alex Dreyfoos Jr. (1954) Career Development Professor of Media Arts Sciences
ACKNOWLEDGMENTS

The Snap! toolkit is the work of Andrew Wheeler, Matthew Lau, and myself. Countless hours spent discussing the merits of flexibility, power specifications, layers of abstraction, and what to order for dinner taught me a great deal about taking an idea from a hazy proposal through the clarifying specification to crystallization by implementation.

Thank you especially to Andy for relentlessly motivating the design through the question, "Why are we building this?" and then thoroughly analyzing the answer. Working with a friend on a project of this magnitude can often strain bonds, but we have managed to keep them from breaking.

Thank you to Herr Doctor Professor Michael Hawley, who has asked, "So, what do you want to do?" since my freshman year. He always provides a fresh perspective on any topic, be it research, job search, or reindeer racing. Mike constantly presents new challenges and opportunities, ready for those with the gumption to attempt them.

Thank you to Christopher Newell, Charlotte Burgess, and Kristin Hall for easing the pains of Snap! development by putting up with unreasonable requests and making the outside world conform to the needs of the Snap! development team.

Thank you to Ben Brown for his work on software and hardware development of transducer components for the SWAT systems.

Thank you to Wendy Ju, without whom the SWAT systems would have been bundles of electronic components duct taped to a ski; she provided the mechanical expertise needed to successfully package SWAT.

Thank you to all members of PIA with whom I have collaborated over the past five years. Conversations with people of all backgrounds and persuasions about films, how people perceive, "really cool stuff," and the dominance of the Cartesian coordinate system will be one of the things I miss most about MIT.

Thank you to Nelson Minar for invaluable advice on how to pursue research, both generally and within the rule-breaking confines of the Media Lab.

Thank you to Sandia Ren for comfort, encouragement, and advice throughout the past two years. Her patient listening to my ponderings and clear perspective on what is actually important have kept me on track in a number of ways.

Finally, thank you to my parents, Bernard and Carol Debski. Their guidance, support, and love have led to whatever I have accomplished or hope to achieve in the future.
# Table of Contents

1 Overview ........................................................................................................... 11
   1.1 Introduction .................................................................................................. 11
   1.2 Motivation .................................................................................................. 12
      1.2.1 Event coverage .................................................................................. 13
      1.2.2 Product engineering ......................................................................... 13
   1.3 Thesis structure ......................................................................................... 14

2 Background ...................................................................................................... 15
   2.1 A History of Personal Information Architecture sensor networks .......... 15
      2.1.1 MarathonMan .................................................................................. 15
      2.1.2 GeoPak ............................................................................................ 16
      2.1.3 BioPak ............................................................................................. 16
      2.1.4 C2C .................................................................................................. 17
   2.2 Need for Snap! .......................................................................................... 18
   2.3 Related work ............................................................................................ 19
      2.3.1 IEEE 1451 ...................................................................................... 19
      2.3.2 LonWorks and other industrial networks ....................................... 20
      2.3.3 Smart Dust and µAmps .................................................................... 21
      2.3.4 FitSense ............................................................................................ 21
   2.4 Summary .................................................................................................. 22

3 Design ............................................................................................................... 23
   3.1 Functional description .............................................................................. 23
   3.2 Design considerations ............................................................................. 24
      3.2.1 Scalability ........................................................................................ 24
      3.2.2 Extensibility ...................................................................................... 24
      3.2.3 Low power ....................................................................................... 25
      3.2.4 Fault tolerant ................................................................................... 25
      3.2.5 Plug-and-play .................................................................................. 25
   3.3 Snap! architecture .................................................................................... 25
      3.3.1 The Snap! packet ............................................................................ 26
      3.3.2 Opcode descriptions ....................................................................... 27
      3.3.3 Variable description ....................................................................... 29
      3.3.4 Required variables ....................................................................... 30
      3.3.5 Bootstrapping ............................................................................... 32
      3.3.6 The Snap! node ............................................................................. 33
   3.4 Summary .................................................................................................. 37

4 A First Implementation .................................................................................... 39
   4.1 Preliminary information ........................................................................... 39
      4.1.1 PIC 16F876 ..................................................................................... 39
      4.1.2 The inter-integrated circuit (I²C) communication specification ...... 39
   4.2 Network interface module ....................................................................... 40
      4.2.1 Wired ............................................................................................... 40
4.2.2 Wireless................................................................................................................... 41
4.3 Input/Output module.............................................................................................. 42
  4.3.1 Hardware........................................................................................................ 42
  4.3.2 Software......................................................................................................... 43
4.4 SWAT: Skiing With Advanced Technology....................................................... 45
  4.4.1 SWAT motivation......................................................................................... 45
  4.4.2 Composition............................................................................................... 47
4.5 Summary ............................................................................................................ 51

5 Results and Evaluation ....................................................................................... 53
  5.1 Field Test: Iceland........................................................................................... 53
    5.1.1 What went wrong.................................................................................... 53
    5.1.2 What went right..................................................................................... 54
    5.1.3 Lessons learned....................................................................................... 56
  5.2 Field Test: Norway........................................................................................... 56
    5.2.1 What went wrong.................................................................................... 57
    5.2.2 What went right..................................................................................... 57
    5.2.3 Analysis of collected data ....................................................................... 58
  5.3 Conclusions ..................................................................................................... 62
    5.3.1 Snap! weaknesses.................................................................................. 62
    5.3.2 Snap! contributions................................................................................. 63
  5.4 Future work....................................................................................................... 64
    5.4.1 Extension of SWAT............................................................................... 64
    5.4.2 Full implementation of all opcodes and variables............................... 65
    5.4.3 Addition of planned peripherals......................................................... 65
    5.4.4 Migration to other platforms ................................................................ 66
    5.4.5 Use for other applications ..................................................................... 66

A Appendix ............................................................................................................. 67

Bibliography ........................................................................................................... 69
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Wired athlete</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>MarathonMan system</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>GeoPak system</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>BiPak system</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>Snap! packet format</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>Opcode meanings</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Snap! node</td>
<td>33</td>
</tr>
<tr>
<td>8</td>
<td>Input/output module examples</td>
<td>35</td>
</tr>
<tr>
<td>9</td>
<td>First implemented networks</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>Snap! node implementation</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>Iceland SWAT system</td>
<td>48</td>
</tr>
<tr>
<td>12</td>
<td>Norway SWAT system</td>
<td>49</td>
</tr>
<tr>
<td>13</td>
<td>Norway SWAT system diagram</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>SWAT system</td>
<td>51</td>
</tr>
<tr>
<td>15</td>
<td>Accelerometer directions</td>
<td>54</td>
</tr>
<tr>
<td>16</td>
<td>Iceland SWAT system acceleration</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>Iceland SWAT system heart rate</td>
<td>55</td>
</tr>
<tr>
<td>18</td>
<td>SWAT browser</td>
<td>58</td>
</tr>
<tr>
<td>19</td>
<td>Ski measurements</td>
<td>59</td>
</tr>
<tr>
<td>20</td>
<td>Skier stride as shown by toe angle</td>
<td>60</td>
</tr>
<tr>
<td>21</td>
<td>Stride analysis with toe angle and boot flex</td>
<td>60</td>
</tr>
<tr>
<td>22</td>
<td>Ski motion through acceleration</td>
<td>61</td>
</tr>
</tbody>
</table>
Section 1

Overview

1.1 Introduction

Imagine the following three scenarios. To commemorate the warriors who saved their infant king in 1205, thousands of cross-country skiers race each year in the Norwegian Birkebeiner, a 58-kilometer cross-country ski marathon through several mountain passes. Biologists studying elite athletes who complete the race in as a little as 2 hours 45 minutes have placed sensors on the skis, boots, and bodies of the a few expert skiers. They hope to learn more about the stresses that the full-body exertion required by cross-country skiing places on the human body. Attempting to complete the race themselves, some of the researchers also wear the equipment. Analyzing the differences between the elite skiers and the researchers will aid the novices in their own skiing efforts, as well as further illuminate how the elite skiers have incorporated crucial elements of technique into their now-natural movements. The results: none of the novices has yet mastered the heel-plant and alterations in stride that allow the elite skiers to move so nimbly up the mountains. Additionally, one of the researchers shows excessive slip in stride, testament to the lack of ski base preparation so crucial with waxable racing skis.

A team of kineticists from the same laboratory work to study of the effects of daily walking exercise among workers with sedentary jobs. The researchers hope to monitor changes in metabolic rate as the participants go from almost no daily exercise to between thirty and forty-five minutes of walking each day. Each of the participants dons a chest strap, a slightly hefty belt-pack, and an ear clip before going for their daily walk. The equipment in the fanny pack records the data over the sixty-day trial period for later analysis by the scientists.

Meanwhile, conservationists in Rwanda work to save the mountain gorillas living in Virunga National Park. Refugees from the surrounding war-torn areas continue to poach the gorillas, threatening the survival of the species. The park officials have created a sensor network that includes several digital cameras, solar battery chargers, and communication equipment. In well-hidden locations throughout the park, each station monitors several square miles of forest. At night, the infrared abilities of the cameras allow for the continuing protection of the gentle beasts. The data are relayed every hour to park headquarters or whenever the system detects that a poaching may be in progress. The quick reporting of these situations allows the Park authorities and environmental groups to intervene before a slaughter occurs or apprehend the criminals, preventing further damage by the same group.
The three systems presented in the imagined scenarios above differ considerably. The Birkebeiner configuration requires equipment and packaging that can withstand the cold, wet conditions present when mounted close to the snow as well as motion and shock challenges. The relatively constant data streams require high-bandwidth connections. The walking study requires low maintenance and low power equipment. It may not demand as much bandwidth as the skiing experiment, but the connection must be reliable to make the study worthwhile. The African setup must run for an extended time without attendance and ideally be self-powered. It must perform image analysis of sufficient complexity to determine that it should alert authorities to a potential poaching.

Despite their significant differences, the three sensor networks described in the previous section perform the same fundamental functions. Data are collected from a network of diverse sensors, stored, in some cases analyzed locally, and finally transmitted to a system capable of more powerful computation, for monitoring and study. The researchers depicted above are not interested in building sensor networks; they are studying elite athletes, improving quality of life, and saving a disappearing species. The sensors each group must use often exist, but as an inharmonious jumble of devices. Often, connecting them requires a Herculean engineering effort. Outsourcing the construction is often too expensive given the short length or small budget of the study. The Snap! toolkit takes advantage of the commonality among the systems to reduce the effort required to network the devices.

The Snap! toolkit is designed to make constructing small, human-scale sensor networks fast and easy. Snap! standardizes and specifies the common functions of nodes on sensor networks, such as reporting data and announcing presence. Snap! permits enough flexibility at the node level to allow almost any type of sensor to be attached to the network. It also describes the essential characteristics of the communications link between the nodes. A shelf of components built following the Snap! specification would allow the creation of a customized sensor-network in a matter of days or weeks, instead of several months. Snap! is not the ideal solution for all sensing applications and does not try to be. Emphasis has been placed on hardware and software reuse and rapidity of design, not suitability for particular size, power, or bandwidth requirements. For instance, Snap! would not be an appropriate solution for developing a "sensor shirt" intended for constant wearing as a marketed product. However, during development of the shirt Snap! would be a valuable asset in determining an appropriate suite of sensors and final networking setup. Using a Snap! platform, the data gathering problems mentioned above can be plausibly addressed.

1.2 Motivation

In addition to the imagined scenarios, two broad areas currently demand the solution that Snap! offers.
1.2.1 EVENT COVERAGE

Conversations with two different companies involved with athletic endeavors provided motivation for the Snap! toolkit. Quokka Sports, a media company dedicated to providing its viewers with total sports immersion, already presents viewers with coverage from the driver's seat of race cars, from the buoys of yacht races, and from the site of the 2000 Olympics. Each of these events has its set of sensor network requirements. Wind speed measurement is a requirement for yacht racing but may be useless inside an enclosed stadium. In addition to consistent coverage of the events mentioned above, Quokka frequently features special one-time events. These have included coverage of a multi-day cross-desert foot race, rock climbing, and the special challenges associated with covering the EcoChallenge, a multi-day trek including biking, hiking, kayaking, and other outdoor skills. If the sensor networks they used could be adapted quickly to the changing nature of the events covered, Quokka and other broadcasters like it could offer their viewers greater contact with the athletes involved. Furthermore, the existence and reuse of the desired sensors would increase the chance that a given measurement was taken. A sensor toolkit would reduce the engineering effort required to place existing sensors into new configurations as well as design new sensors for special event coverage.

1.2.2 PRODUCT ENGINEERING

L.L. Bean sells a variety of outdoor sports clothing and gear. It has found that user feedback is invaluable in both evaluating its current products as well as designing the next generation of sporting equipment. The company currently relies primarily on subjective telephone or journal reports from field testers. The reports vary, for better or worse, according to the individual tastes, preferences, and abilities of the tester. In some cases, the company has developed methods for objectively testing their products. For instance, product developers test sleeping bags by placing temperature sensors on subjects, attaching them to laptop computers, and monitoring conditions throughout the night (D. Bennell, L.L. Bean Product Research & Testing Manager, personal communication, December 1, 1999).

Similarly, customer feedback also influences the designs of snowboard manufacturers. From the effectiveness of a new binding system to the effects of a deeper sidecut on turning control, conditioned riders report their opinions on the effects of a change in design. However, when attempting to monitor
the equipment objectively, the weight and bulk of the sensing and telemetry systems prevented the testers from using the equipment naturally, resulting in invalid testing data (P. Maravetz, Burton Snowboards, Sports Product Development Lecture, January 18, 2000).

Product design, from everyday apparel to the most advanced athletic gear, could benefit from monitoring variables that are relevant in the particular design. While the technology exists for all developers to design custom systems, few companies actually monitor their equipment in the field. They take most of their measurements in a lab. A sensor toolkit would allow designers and testers to choose among existing sensors needed for a particular product test. A toolkit would allow a designer to incorporate a new, non-standard sensor into the network without modification to the other sensors or to the existing sensor network. The result: products that take into account objective and subjective data that perform to an overall more satisfying level.

1.3 Thesis structure

This thesis is composed of five sections: Section 2 illustrates previous work that inspired the development of the Snap! system, as well as concurrent and previous research that influenced the design of the Snap! toolkit. Section 3 describes the design goals and constraints of the Snap! system, and details its design. Section 4 depicts the first Snap! platform as well as the specifics of the first implementation in a cross-country skiing system. Section 5 discusses the results obtained from the cross-country ski system and evaluates the success of the Snap! toolkit in achieving its goals. Section 5 also presents future research directions for Snap!
Section 2

Background

2.1 A History of Personal Information Architecture sensor networks

The Personal Information Architecture group (PIA) began putting bodies online in 1997 as part of its Black Boxes project. The evolution of these systems provided impetus and experience for Snap!'s development.

2.1.1 MARATHONMAN

PIA designed its first system, MarathonMan, for the 1997 Boston Marathon. It comprised a heart rate monitor, a Global Positioning System receiver (GPS), an accelerometer for measuring cadence, and a body core temperature monitor. A separate device with its own microcontroller collated the data from all the sensors. This device sent the data to a palmtop computer, which both recorded and sent them via cellular modem to the Internet. The data from the palmtop could be analyzed in the event that the Internet link disconnected any time during the event.[30]

![Figure 2: MarathonMan system](image)

**FIGURE 2: MarathonMan system** PIA first developed MarathonMan to enable online body monitoring. The system featured a main collating board that knew how to interpret the signals of each sensor. The collator polled each constituent sensor, then sent the values to a palmtop computer that stored the data and transmitted them via cellular modem to the Internet.

Learning from successes and failures in the Boston Marathon, the system was redesigned for the San Francisco Marathon later the same year. In addition to the sensor set used in Boston, the runners also carried cellular phones, adding the recording of real-time subjective data to the sensor readings. The PIA
team also modified the MarathonMan system for use by the United States Army. The Army used the system, with more rugged packaging and without wireless communication, to monitor soldiers during field training exercises at Fort Bening.\[14\]

2.1.2 GEOPak

Following the MarathonMan project, PIA teamed up with the 1998 United States Yale/NASA/MIT Expedition to Mount Everest. PIA developed two different systems for the expedition: GeoPak and BioPak.

Climbers carried the GeoPak systems in their backpacks to monitor changing conditions as they ascended Mt. Everest. Each GeoPak consisted of a custom designed controller board attached to five sensors: a GPS receiver, a photoresistor for measuring light levels, a thermistor for measuring temperature, a humidity sensor, and a barometric pressure sensor. The controller board also flash memory on-board for persistent data logging, as well as a connection to short-wave radio for data transmission.\[17\]

![GeoPak system](image)

**FIGURE 3: GeoPak system** GeoPak recorded environmental conditions on Mt. Everest in 1998. The GeoPak board integrated controller and memory. The controller knew the protocols of each sensor. It polled each sensor, stored the values on-board, and relayed them via shortwave radio to a remote monitoring station.

2.1.3 BIOpak

The BioPak system monitored the physiological conditions of the climbers. Each BioPak fit entirely on the body of the climber, with no apparatus external to the wearer's clothing. Each BioPak contained four sensors: a heart rate sensor, a blood oxymeter, a skin temperature sensor, and a body core temperature sensor. An interface board connected to each sensor and to a PC serial bus. The interface boards
converted the individual sensor signals to a common protocol. A custom-designed BioPak controller board also connected to the PC bus. The BioPak controller was attached to a memory board and a short-wave radio. The controller polled each interface board, logged the data, and transmitted them. [13]

![Diagram of BioPak system](image)

**Figure 4: BioPak system** BioPak monitored physiological conditions of climbers on Mt. Everest in 1998. BioPak sensor boards knew how to communicate with the transducers to which they were attached. All the sensor boards communicated via a uniform protocol with the master controller. The master controller stored the data in off-board flash memory, and sent the data via shortwave radio to a remote monitoring station.

2.1.4 C2C

During the summer of 1998, PIA worked on C2C, a project to track a team of cyclists as they pedaled from Seattle, Washington to Washington, DC. The system comprised a heart rate sensor, GPS receiver, environmental temperature sensor, skin temperature sensor, bike speedometer, and cadence sensor. The sensors were attached to a custom controller board that collected data from each of the sensors. The controller collated the data and sent them to a personal digital assistant (PDA). The PDA uploaded the data to a personal computer at the end of each day for permanent storage and analysis.[6]

2.2 Need for Snap!

The systems described above all encompassed goals and component sets common to many sensor networks.

- Sensors - each system incorporated a variety of sensors. Some transducers had extremely basic signals: the thermistor, photoresistor, and barometric pressure sensors output a voltage that varied linearly with the measured variable. Other sensors communicated their data via a standard serial
communication method such as 1C or RS-232. Another class used pulse width modulation as an output. Each of the described systems used sensors with at least two different interface types. Whatever the output of a component, it must have been translated at some point in the system into a human-interpretable form.

- Logging - all of the systems presented above had non-volatile memory for storing the collected data. In some instances, the onboard system memory was also the terminus of the data stream. The memories varied from those present in palmtop computers and PDAs to custom designed boards specific to the requirements of the given system. In all of the above cases, the data are not analyzed on board the system in any way.

- Data transmission - in several of the cases above, real-time remote monitoring of the data was a primary system goal. The communication method designed into the system was based on the infrastructure of the locale in which the system was deployed. The communication systems were the components most prone to error, which made logging within the system an absolute necessity to mitigate data loss.

Despite the commonalities between the systems, PIA essentially designed each system from scratch. Some of the changes were unavoidable. The marathon runners demanded that the systems be comfortable and lightweight. Otherwise, no useful data could be collected because the equipment would affect runners' performance. The Fort Bening systems used the same hardware and software as the San Francisco marathon, but repackaged it for the expected shocks of combat simulation. The delicacy with which the runners handled the systems could not be expected of the Marines. The systems used on Mount Everest had to be built to withstand the harsh mountain environment. The conditions required that the included components have greater ranges of operation. The GeoPaks had to operate for several days without a change of batteries. The climbers could then leave them in a remote location to monitor the weather without devoting precious time and energy to their maintenance.

On the other hand, the system redesign was extremely expensive in terms of time spent developing hardware and software. The architectures of the systems differed remarkably. The software interfacing with each of the sensors had to be modified to fit the particular system. For instance, the sensors used in the C2C project were sufficiently different that the extensible properties of the BioPak system could not be readily adapted for its needs. Thus, despite the fact that the systems differed only minimally in their function, each required a massive engineering effort to complete, with little reuse of code or hardware. This problem of reinventing the wheel as new system requirements arose motivated the creation of a sensor toolkit.
2.3 Related work

The domain of inquiry surrounding sensor networks is absurdly large. Research into sensor networks includes those on satellites orbiting the earth photographing the habitats of manatees, spread across the continent recording the current weather conditions, dispersed as dust detecting tanks crossing a battlefield, and on a single person running a marathon. There are dozens of sensor protocols and means for creating sensor networks.

Descriptions of sensor networks vary along several axes. These include:

- Remote sensing versus local sensing: are the data collected exclusively for use in the network or are they exchanged with another system?

- Distributed processing versus central processing: do the nodes on the network perform computation beyond that required to record sensor data or do they simply package the data and send them over the network link?

- Local control versus central control: are the nodes responsible for coordinating their activity among themselves or does a central node dictate operation of each individual node?

- Tens of nodes versus billions of nodes: does the network consist of few or many nodes?

- Microwatts versus kilowatts: do the nodes run off the power they can harness from the environment, from batteries, from power lines, or from another source?

- Guaranteed communication versus sporadically available communication: are the nodes connected via a reliable link, such as Ethernet, or a less reliable link, such as ad-hoc, line-of-site network?

The purpose of each sensor network constrains or determines the choice made along these axes, as well as others. The Snap! design gives the system designer great leeway in all of the axes. A toolkit that is too limiting ceases to be useful. The technologies presented in this section either make specific choices along these axes that provided insight when designing Snap! or attempted to enable sensor network construction via different means than those Snap! uses.

2.3.1 IEEE 1451

The Institute of Electrical and Electronics Engineers (IEEE) has been developing a set of standards for creating digital networks of smart sensors. This set of standards, labeled 1451, comprises four parts. Only one has been officially approved, with the other three proceeding slowly. The four parts progress hierarchically from the interface among several different analog and digital transducers (1451.4), through
the collation of the signals (1451.3), to the final digital presentation of the sensor data (1451.2), and concluding with the communication between the devices on the sensor network (1451.1).[22]

The sanctioned 1451.2, "A Smart Transducer Interface for Sensors and Actuators,"[35] describes the interface between a sensor module, which consists of a single transducer with one or more channels, and a device that controls the sensor module and is also attached to the sensor network. The standard is divided into two parts: the physical interface and the definition of a transducer electronic datasheet, or TEDS. The TEDS contains device identification and characterization data, as well as a means with which to specify the translation from the transducer's raw output to a meaningful representation.[7]

The 1451 standard set, if successful, will reduce the burden of the sensor network designer. No longer will he struggle with the outrageous number of protocols defined by each sensor manufacturer. With respect to the goals of the Snap! system, however, 1451 is overkill. The standard presents too many requirements to the system protoyer. For example, 1451.2 specifies the exact number of data lines and the timing diagrams for the signals on the lines to each transducer. The TEDS contains data fields that may only be useful in exceptional circumstances. The time it would take a node designer to implement the 1451.2 standard prevents it from being used to prototype networks quickly and easily.

Additionally, 1451 includes standards for interoperability within the nodes themselves. In order to allow the system designer maximum flexibility, Snap! describes the external behavior of nodes, and only minimally within nodes. The system designer determines the implementation details on the node level. The interoperability to be gained by incorporating the standard into Snap!, or simply building 1451-compliant networks is not a necessary design feature of Snap! However, Snap! includes some of 1451's end-to-end description, as discussed in section 3.

2.3.2 LONWORKS AND OTHER INDUSTRIAL NETWORKS

Echelon Corporation [10] markets a sensor networking solution based on its NeuoronChip and LonTalk protocol. The network possesses several desirable qualities. The nodes themselves are dynamically reconfigurable, the network protocol is media-independent, and an entire suite of external applications exists to monitor the network remotely. Example applications of the network include controlling aircraft or automating buildings and homes.

LonWorks and other industrial networks, such as CEBus and CAN bus, all have properties that allow for flexibility along several of the description axes. Often, however, any specific network contains a constraint that is undesirable in a toolkit that seeks to be human-scale and as flexible as Snap! endeavors to be. That is, each of the networks is an implementation of something that could be built using the Snap!
toolkit. Each of the networks, though, represents a potential use for Snap! Thus, the Snap! design considered the various capabilities of each specific network. If Snap!’s design fundamentally prevented it accomplishing what an existing implementation could, the design had to be reassessed.

2.3.3 Smart Dust and µAmps

A research group at the University of California-Berkeley is producing Smart Dust. [16] Its goal is to encapsulate a transducer (often a MEMS device), input/output and control circuitry, and wireless, optical communication in a single unit smaller than 1.5 mm³. The Smart Dust project demonstrates what can be done when sensors are pushed to their physical size limits. The group envisions the dust being spread over an area via micro-air vehicles or being attached to a computer user’s fingertips as a new interface device.

A group at MIT is also developing technologies for small, short-range, wireless sensor networks. The µAmps project focuses expressly on the problems associated with powering the nodes and communicating within such a new network paradigm.[8]

Both of these groups are conducting research that pushes the bounds of the sensor community. Undoubtedly, they must answer questions similar to those faced by Snap! during its development. However, the significant differences between the goals of each system leave a wide chasm between the groups when answering the questions. While it is important to be aware of developments in these groups, as well as the dozens of other research groups conducting investigations into all areas of sensor research, the differing goals between non-overlapping systems must be kept in mind both during development and evaluation of a given system.

2.3.4 FitSense

FitSense [11] has developed the BodyLAN, a wireless digital link connecting sensors that are worn on the body. Their current array of sensors, some of which were used in the MarathonMan project, includes a body core temperature monitor, devices to measure environmental data, speedometers and odometers, motion-sensors, blood oxygen content sensors, energy expenditure estimators, and other vital sign monitors. The system is plug-and-play, so new sensors can be added to the network without reconfiguration of the other devices. The FitSense system overlaps with some of the goals of the Snap! project. Of existing sensor networks, the FitSense system most closely matches the physical size that the Snap! system addresses.
On the other hand, the Snap! system is designed to make the creation of new sensors easy, in addition to permitting plug-and-play of existing sensors. The Snap! system is designed to work with an extremely wide array of sensors, some which may not even exist yet. Snap! also allows for the dynamic reconfiguration of its nodes' behaviors. The FitSense system functions in only one mode and its ease of extensibility cannot be determined. Finally, the Snap! system is completely open, whereas FitSense is a corporation and motivated to retain its proprietary technology. The FitSense system presents a good solution for sensor networks with needs satisfied by the nodes that FitSense offers and the predetermined network configuration.

2.4 Summary

This section reviewed the several projects within the Personal Information Architecture group that have involved capturing and recording data from bodies and other small areas. It highlighted specific characteristics of the architectures of each of the projects. Finally, the section concluded with a survey of related research into sensor networks. The next section will explain fully the goals of the Snap! system, its design constraints, and the design chosen to meet the goals and requirements.
Section 3

Design

3.1 Functional description

The Snap! toolkit takes advantage of the similarities among sensor networks to make prototyping them fast and easy. Simultaneously, it acknowledges their inherent differences, remaining flexible in order to accommodate the needs of different systems. Snap! specifies the behavior of the nodes on a sensor network and describes the logical structure of each node and of the network as a whole. The Snap! design places emphasis on the reusability of hardware and software, rather than on the creation of an optimal solution for a given application.

Each sensor network has its set of requirements. Some might demand a few sensors that wait for a single master to poll them, for ease of design and lack of network collisions. Others may require that sensors only transmit when a significant event has occurred, in order to minimize network traffic, storage space, and power requirements. Another network may require that its sensors report to a single collator, which then makes the readings available to any monitor to request them, in a publish-subscribe paradigm. There are many more examples. The Snap! toolkit is designed to allow system designers to configure the network to whatever archetype is required. Furthermore, the toolkit does not require that its nodes be redesigned for such changes. In fact, the nodes may be changed even while operating, allowing the designer to change his configuration in response to changing power availability or network reliability.

Similarly, the Snap! toolkit is intended to be independent of a particular microprocessor or controller. Each node is only required to implement the behaviors of a Snap! set forth in the Snap! specification. This is important because the processing required on different nodes may vary from network to network. For instance, a temperature node may require only a small microcontroller to do analog to digital conversion, turn the value into a meaningful number, and support the other Snap! functions. Another node may consist of a high-powered processor and plentiful memory to convert the bitmap output of an attached CCD camera into the JPEG image file format. Both nodes may be a part of the same Snap! network.

In addition to sensors, the Snap! toolkit is also designed to allow actuators and other output devices to be present on the network. An actuator is any kind of output device. It may be something that generates motion or that displays or transmits information. A Snap! network may include light-emitting diodes (LEDs), liquid-crystal displays (LCDs), and electric motors. Other examples of output devices are long-
distance transmitters, such as cellular phone modems or short-wave radios, and flash memory cards for persistent storage of data.

The Snap! toolkit is independent of the actual network connectivity. The toolkit places some basic requirements that the network must have, but the choice of the actual protocols and media connecting the nodes is not specified. Of course, the quality of the network connecting the nodes will have an impact on the overall performance of a sensor network built with Snap!; a designer should not expect to transmit several images per second over a network with a bandwidth of 2.4 kilobaud!

3.2 Design considerations

In order to satisfy the functional goals of the Snap! toolkit, with the above factors in mind, the following considerations were taken into account during its design.

3.2.1 Scalability

A sensor network should be scalable. A minimal sensor network may contain as few as two nodes: a sensor and an actuator or recording device. A simple heart rate sensor with a wrist-mounted display is one such example. At the opposite end of the spectrum, a complete body-monitoring network may consist of fifty or one hundred sensors. These could include seven different electrodes for an electrocardiogram, various skin temperature monitors, a heart rate sensor, and blood pressure detector. Additionally, on-body sensors for detecting local environmental conditions, such as temperature, global coordinates, atmospheric pressure, and humidity may be part of the system. The Snap! toolkit should be able to accommodate both sensor network designs. It is not necessary, though, that the toolkit support thousands of sensors. While such large networks are conceivable, Snap! does not address these extraordinary cases. The design can avoid unneeded complexity and complication by keeping the number of sensors allowed to a reasonable number.

3.2.2 Extensibility

One of the most important features of the Snap! toolkit is its extensibility. The addition of new components to a system must not require that it be redesigned. New modules conforming to the Snap! specification must be compatible with existing modules. The network protocol must be capable of being dynamically extended. The Snap! toolkit cannot anticipate all sensors that currently exist and certainly not all modules that might ever become part of a network. The Snap! toolkit should not strictly specify hardware to be used, as it may become obsolete as more powerful components become available.
Overall, the design should allow extensions without affecting components already painstakingly designed and debugged.

3.2.3 Low Power

The Snap! toolkit design must acknowledge that many likely applications will have limited power supplies. One example is a camera placed in a remote location that responds to a certain trigger condition, such as obstruction of a light sensor to monitor rarely seen animal species. When triggered, it records environmental data along with an image. Remote environmental monitoring systems would be expected to run for months or years on a single power supply, and certainly without power lines. Snap! should not require that any piece of hardware constantly draw high currents for computation or communication.

3.2.4 Fault Tolerant

The Snap! toolkit should allow the design of systems that are fault-tolerant. Each sensor network is vulnerable to different faults. However, the Snap! toolkit should not contain specifications that make it particularly likely to fail given common uses of sensor networks. Common sensor network failures include failure of a central control point, intermittent network connectivity, and the breakdown of a subset of nodes. When a system designer decides that a failure mode is unlikely, his network may permit it. For instance, a designer may be convinced that one node is particularly bulletproof and choose to have a central controller. However, Snap! must not impose a common failure mode on the system designer.

3.2.5 Plug-and-Play

Similar to extensibility is a requirement that the Snap! tool kit provide facilities for a plug-and-play network. It should be possible for new modules to be added to an existing sensor network without reconfiguring the network. This type of auto-detection by the network and its nodes allows for further ease of expansion of capabilities. A sensor network that necessarily requires reconfiguration when adding or subtracting components inhibits the ability to quickly create prototypes.

3.3 Snap! architecture

Most of the design emphasis in the Snap! architecture is placed on the nodes. Only a very lightweight specification is required of the network. All the nodes on a Snap! network should implement the Snap! specification, even if they are not all composed of identical hardware. The basic element of communication among the Snap! nodes is a Snap! packet. Understanding its elements first will make later explanations clearer.
3.3.1 The Snap! Packet

The Snap! packet contains several fields:

<table>
<thead>
<tr>
<th>Destination address</th>
<th>Source address</th>
<th>Opcode (big-endian)</th>
<th>Length</th>
<th>Opcode-dependent payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 bytes</td>
<td>2 bytes</td>
<td>1 byte</td>
<td>2 bytes</td>
<td>&quot;Length&quot; bytes</td>
</tr>
</tbody>
</table>

**FIGURE 5: Snap! packet format** All Snap! packets contain a 2-byte destination address, a 2 byte-source address, 1 opcode byte, 2 bytes of length, and a variable-length payload.

- **Address fields** - the two address fields contain network independent addresses. Each node on the network has exactly one Snap! address. The Snap! address may be the same as that used by one of the networking layers, but it is not required that the network stack use that address. One address within the Snap! address space must correspond to a broadcast address. A packet with the broadcast address is intended for all the nodes on the Snap! network. The destination address is the address of the node for which the packet is intended. The sender address is the address of the node which first generated the packet.

- **Opcode** - the opcode describes the contents of the packet and the intended effect on its recipient. The payload contents also have different meanings depending on the opcode. The opcode values and the corresponding payload definitions are described later in this section.

- **Length** - the length is a two-byte value that indicates the length of the payload. It is a big-endian value and may be zero if there is no payload.

- **Payload** - interpretation of the payload contents depends on the opcode of the packet. The payload contains a number of bytes equal to the length specified in the packet. If the length is zero, there is no payload. In most cases, the payload consists of 8-bit ASCII (American Standard Code for Information Interchange) characters. This choice was made to assist in debugging, standardization, and ease of display. The requirement that much of the communication be ASCII-encoded often results in the node performing multiple conversions between a native format and ASCII. However, since a major goal of Snap! is to reduce the time it takes to create sensor networks,
the reduction in debugging time balances the decrease in performance from greater network traffic and multiple format conversions on the node.

3.3.2 Opcode Descriptions

The opcode must be one of the six in the list below. Some of the opcodes involve operations on the variables or values of a node; descriptions of variables and value classes follow in a later section.

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
<th>Payload contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgment</td>
<td>Short packet for communicating status</td>
<td>One byte acknowledgment code</td>
</tr>
<tr>
<td>Report</td>
<td>Response to a request or spontaneously generated to communicate current state</td>
<td>Name of variable reported and value of the variable</td>
</tr>
<tr>
<td>Request</td>
<td>Request another node's variable values</td>
<td>Name of variable requested</td>
</tr>
<tr>
<td>Set</td>
<td>Changes or assigns a value to a variable on another node</td>
<td>Name of variable to set and new value of the variable</td>
</tr>
<tr>
<td>Find</td>
<td>Discovers nodes on the network that meet specified criteria</td>
<td>A query string to determine which nodes respond</td>
</tr>
<tr>
<td>User-defined</td>
<td>Included to allow expansion of networks containing nodes whose behavior cannot be encapsulated by the existing opcodes</td>
<td>Dependent on user specification</td>
</tr>
</tbody>
</table>

FIGURE 6: Opcode meanings A packet's opcode determines its function

3.3.2.1 Acknowledgment:

The acknowledgment opcode indicates that the payload contains a single-byte acknowledgment code. The acknowledgment codes include:

- SUCCESS - indicates an operation was successful
- FAILURE - indicates an operation failed
- SYNC - used to loosely synchronize nodes for simultaneous recording
- ERROR - indicates that the last operation generated error
- PING - used to request that nodes receiving the packet generate a PING_RESPONSE acknowledgment packet
- PING_RESPONSE - generated in response to a PING acknowledgment packet

Acknowledgment packets are generated in response to set, find, and some acknowledgment packets, as well as to request packets when the node cannot properly respond to the request. Acknowledgment packets may also be spontaneously generated by devices attempting to determine the composition of the network, those that are attempting to coordinate simultaneous measurement, during a node's
bootstrapping, or according to one of its behaviors. Acknowledgment packets describe the node's operating state.

3.3.2.2 Report

The report opcode indicates that the payload contains a variable name followed by the node's current value for that variable. In a report packet, the payload consists entirely of ASCII characters. The name of the variable and all data fields reported terminate in a null character (0). If a variable value cannot be represented by ASCII characters, such as an image or audio stream, the payload contents may be the byte stream. In this case, the implementer must decide how to delimit fields. A node generates a report packet in response to a request packet, as described in the request opcode description, or spontaneously, according to one of its behaviors. A node receiving a report packet is not required to respond in any way.

3.3.2.3 Request

In a request packet, the payload contains the name of a variable. The name is 8-bit ASCII encoded and terminates with a null character. The node receiving the request packet responds either with a report packet containing the requested variable name and its current value or an acknowledgment packet with the acknowledgment code set to ERROR. A node generates a request packet to facilitate recording, transmitting, or displaying data, internally, to influence its own behavior, or to determine information about the nodes or the network overall.

3.3.2.4 Set

A set packet's payload contains a variable name followed by the value that should be assigned to the specified variable. The variable name and data fields use 8-bit ASCII encoding and are null-terminated. If the named variable comprises more than one field, the data for each field should be specified. If only a small subset of a multi-field variable needs to be set, it may be best to move down the variable hierarchy to set only the desired portions. A node receiving a request packet should respond with an acknowledgment packet with the code set appropriately:

- SUCCESS, if all of the fields in the named variable were set to the values specified.
- FAILURE, if one or more of the fields in the named variable could not be set. In this case, the node should not alter any of the fields.
- ERROR, if the named variable does not exist or contains a different number of fields than included in the set packet, the data specified is illegal, or one of the fields of the named variable is read-only. In this case, the node should not alter any of the fields.

The intention is for the node to respond with FAILURE if the set packet is legal, but current
circumstances prevent the node from acting on it. Such a response may cause the sender to resend the packet after a certain delay. An ERROR response indicates that the set packet is inherently flawed and the sending node may need to obtain more information about the destination node before trying again. A device for configuring a network will generate set packets, as well as other nodes managing the network's behavior.

3.3.2.5 Find

The first field of a find packet contains a query string, which is not well-defined in the first version of Snap! The query string can reference any node variable. Each node to which a find packet is addressed parses the query string and determines if it satisfies the query. If the node matches the string, it should generate an acknowledgment packet with the acknowledgment code set to PING_RESP. The find packet allows nodes to discover the make-up of a specific network, allowing nodes to collude to perform complicated sensing tasks.

3.3.2.6 User-defined

The user-defined opcode is included for the expansion of the Snap! system beyond the needs and uses foreseen during its design. The payload of a user-defined packet must contain the number of bytes specified in the length field. User-defined packets may be generated by any node for a variety of reasons. A node is not required by the Snap! specification to respond to a user-defined packet. The implementer will decide how a node actually responds to a packet with the user-defined opcode.

3.3.3 VARIABLE DESCRIPTION

A node's variables characterize its static and dynamic behavior. Variable names and values are the only data that are communicated in packets. A node is required to implement a prescribed set of variable names. The required set allows network nodes to learn about the capabilities and variable parameters of other nodes. Each sensor and actuator node will additionally choose to implement another set of variables. This set will be customized according to the functionality of the individual nodes.

The variable names follow a hierarchical naming scheme, similar to that used in the Unix file system [32] and the Java package structure.[12] In a variable name, the most general classification comes first and a period (.) separates the hierarchical levels. For example, gps may be the general name referring to the entire set of global positioning system data, while gps.lat and gps.lon refer to latitude and longitude specifically. The hierarchy may be arbitrarily deep. Grouping variable names into a hierarchy allows nodes to request and report data in chunks, reducing network overhead. The hierarchy also allows for a logical organization that helps make data analysis and presentation more meaningful.
3.3.4 REQUIRED VARIABLES

3.3.4.1 Datasheet information: ds

The datasheet contains information that is pertinent to all the transducers and actuators on a node or the node itself. IEEE 1451, as well as a general trend among manufacturers to include electronic data sheets on their products inspired the inclusion of the datasheet. Electronic datasheets facilitate plug-and-play and extensibility of the network by allowing nodes to dynamically determine characteristics of other nodes. The nodes do not need to be pre-programmed with information about other nodes before being connected.

The datasheet variables are:

\( ds.id \): a numeric identifier that is unique among all nodes in existence

\( ds.meas \): a listing of all the measurements that a node may make and the control variables of an actuator. This includes all the levels of the hierarchy. The value for a GPS node may be \( \text{gps, gps.lat, gps.lon, gps.spd, gps.alt, gps.time} \) while a node reporting temperature and barometric pressure as isolated values may define \( ds.meas \) to be \( \text{temp, barom} \).

\( ds.warmup \): among all the sensors and actuators, the longest time until a sensor or actuator is ready after powering on the node in milliseconds.

\( ds.sampling \): among all the sensors and actuators, the longest time that a sensor or actuator value remains constant.

3.3.4.2 Metadata: mt

The metadata variable contains information about the variables and the system software. This variable classification is not completely specified.

\( mt.snapver \): The version of the Snap! toolkit that the node implements. This allows a node to determine the behavior of software on other nodes.

3.3.4.3 Behaviors: bhv

The behaviors of a node are what allow it to fit so flexibly into a variety of network configurations. Behaviors are small scripts or programs that run on the node. If no behaviors are specified, the node will simply respond to incoming packets, reacting according to the Snap! specification for each type of packet.

Behaviors can vary in functionality and scope depending on the underlying hardware. A very simple microcontroller may limit the behaviors to testing current values against thresholds to decide whether a report packet should be generated. A heart rate node may only report when the heart rate goes above 75% of the maximum value or below 75% of the average. This could reduce network traffic and possibly
the power consumption of the node itself. Nodes with more processing power may be able to implement
higher level languages such as Java or C.

High-level languages and powerful host processors allow the execution of mobile code or the habitation
of software agents on the sensor network. These techniques may allow the system to detect very
complicated phenomena. For instance, a process may move among the sensors on person, keeping track
of the various sensor readings it encounters. It may discover that adrenaline in the blood stream,
combined with elevated heart rate, blood pressure, respiration rate, and skin conductivity, indicates that
the person is exhibiting the hallmark signs of a fight-or-flight response. This may cause the system to set
off an audio alarm as well as transmit a warning over the long-range communications device. Such a
behavior is possible even with threshold recording on a single, lower-powered processor. However, such
an arrangement makes the system vulnerable to a single point-of-failure.

The behaviors are represented as variables on the nodes and can be set and read like any other variable.
The language in which the behavior is written will determine what encoding scheme is used when
performing these operations. For example, an interpreted language may use packets with ASCII contents,
while a compiled implementation may set behaviors via packets containing bytecodes. The behaviors can
be changed dynamically and nodes may have a default behavior to be executed if no other is set.

The behaviors variables are:

\[
\text{bhv.1} \leq \text{val} \leq \text{bhv.num} : \text{Each of the numbered variables may contain one of the behavior}
\text{scripts or programs described above.}
\]

\[
\text{bhv.lang.1} \leq \text{val} \leq \text{bhv.lang.num} : \text{Contains the language used to specify the corresponding}
\text{behavior.}
\]

\[
\text{bhv.supplang} : \text{Contains a list of the behavior languages that are supported}
\text{by the node.}
\]

\[
\text{bhv.num} : \text{Contains the maximum number of behaviors that the node can support.}
\]

\[
\text{bhv.cyctime} : \text{Contains the time between execution of the behaviors, in milliseconds. This is not a}
\text{guarantee. Rather, when the node is not busy it will attempt to evaluate all the behaviors this often.}
\text{This variable may be set high to conserve power or set low to ensure constant monitoring.}
\]

### 3.3.4.4 Measurements

Measurement variables refer to those variables added for a particular node. The examples \text{temp},
\text{barom}, and \text{gps} have already been given. In this discussion, \text{measName} refers to any particular
measurement variable implemented on the node. If the value of a variable with more than one field is
requested, the values for each of the active fields should be reported. Active fields are those that are
specified in the format of the multi-field variable. If a variable doesn't make sense for a given transducer, the value is "N/A."

**measName.format**: This is one of the most important variables. The format variable defines which of the variables lower in the hierarchy are active in any action involving `measName`. This allows a designer to choose to use only some of the capabilities of a node without reprogramming it. For instance, `gps` may consist of `gps.lat`, `gps.lon`, `gps.alt`, `gps.spd`, and `gps.time`. The designer may decide that only latitude and longitude are needed for his purpose and that he would like time to be reported in a separate packet. Therefore, he sets `gps.format` equal to `lat,lon`.

**measName.desc**: Contains an ASCII description of the transducer.

**measName.unit**: Contains an ASCII description of the unit of variable and may also contain a formula to convert a raw reading into a more meaningful value.

**measName.max**: Contains the maximum possible value of the variable.

**measName.min**: Contains the minimum possible value of the variable.

**measName.resolution**: Contains the resolution of the sensor.

**measName.warmup**: Contains the time until the sensor or actuator can accept or report a reading after power in, in milliseconds.

**measName.sampling**: Contains the time that a sensor reading remains constant before another value can be read or set.

**measName**: Contains the current value of the sensor or actuator.

**time**: Each node keeps an internal timer that is mostly used for internal timing functions. Referencing this variable from a behavior provides a way to do time-based reporting.

The variable hierarchy may be extended as data is passed along through the sensor network. A designer may cause a node that collates data to create a new data structure that groups various sensors or actuators together. For example, a designer may create a system that has temperature, barometric pressure, and humidity sensors on separate nodes. Another node queries these three as well as a GPS node. The collating device may decide to leave GPS in its own hierarchy, but create a new data type, called `environ`, that pools the environment readings. This aggregation of data allows the designer to create new abstractions that are more representative of the various network constituents.

### 3.3.5 Bootstrapping

The Snap! design does not explicitly define a bootstrapping behavior; each network may vary. For example, in a Snap! network configured as a master-slave design, each non-master node may wait to be polled before sending data or making communication. For a network on which each node spontaneously generates packets, a node may wait a pre-defined time before sending its reporting format and capabilities. The network designer must decide the bootstrapping which makes the most sense given the particular configuration.
3.3.6 THE SNAP! NODE

The Snap! node comprises two modules: the network module and the input/output module. The physical separation of the two modules allows different network hardware to be attached to the same input/output module. A wired network can be changed to a wireless network by switching which network modules are attached to the input/output modules. The logical separation at this level allows designers leeway to manage sensors, actuators, and other resources as they see fit. At the same, the chore of network communication is separated to allow isolation of fault and development.

![Figure 7: Snap! node](image)

**FIGURE 7: Snap! node** A Snap! node contains two modules: the input/output module, which interfaces to transducers, and the network module, which implements the network stack needed to communicate with other nodes.

3.3.6.1 The network interface module

The network interface module is responsible for communicating with other nodes on the sensor network. It contains hardware and software necessary for these communications. Example networks include TCP/IP on twisted-pair Ethernet, I2C serial communication on a two-wire bus, and a custom-designed protocol using a wireless communication chip. The network interface module is entirely responsible for implementing the chosen protocol. The input/output interface module provides no support.

To the input/output interface module, the network interface module presents a very simple interface. The network interface module receives packets from other nodes on the network in the implemented network protocol and translates them into the standard packet format. This standardized format is then passed to the input/output module. The network interface module also receives messages in a standard format from the input/output module and converts them into the format used on the implemented network for transmission to other nodes.
The network presented by the network interface module to the input/output module must allow any node on the network to address any other node. It must also provide a greater than zero probability of any single network communication succeeding.

### 3.3.6.2 The input/output module

The input/output module is where most of the Snap! architecture is defined. The input/output module receives packets from and transmits packets to the network module in the Snap! packet format. While the interface between the two modules may allow the packet to be exchanged in parts, the input/output module is unaware of any networking headers or address scheme. It is only cognizant of the Snap! addresses as discussed in the Snap! packet definition. The input/output module contains the hardware and software interface to the transducer or actuator.

Across instances of a specific hardware platform, many of the Snap! variables will be implemented in a very similar way. The writing of the majority of code for a particular architecture will only have to be done once. Other input/output modules using the same architecture can then take advantage of the library code. The means by which an input/output module reads its sensors or sets its actuators will be unique to each sensor and actuator array. Therefore, custom sensor and actuator code must be developed for each node. Additionally, the tables or functions that store the variable values must be updated from node to node.

The wide range of possible input/output modules gives the Snap! architecture its openness. As long as the input/output module reacts to the packets it receives from the network module as defined in the previous section on packets, it can successfully function as Snap! node. Of course, the way a Snap! module responds to report and set packets, as well as its behaviors and set of measurement variables, determines what role it plays in a network. Some example classes and cases of input/output modules follow.
### Input/output module class | Example
---|---
Sensor | GPS receiver  
        | Heart rate monitor  
Actuator | Audible buzzer  
         | Water valve  
Display | Liquid crystal display touch panel  
        | Personal digital assistant  
Logger | Hard disk drive  
        | Flash memory device  
Gateway | Shortwave radio  
        | Cellular modem

**FIGURE 8: Input/output module examples** Input/output modules may take many different forms

### 3.3.6.2.1 Sensor input/output modules

This class is the most obvious example. Most of the nodes on a given network will have sensor input/output modules. A sensor input/output module may be connected to one or more transducers. In order to facilitate reuse of the input/output module and its accompanying resources, the measurements the transducers make should be conceptually related. There is nothing in the Snap! specification that requires this. However, it will be difficult to reuse the nodes that are painstakingly debugged and engineered if they contain configurations of transducers that are specific to a certain application. Such a design would defeat a large part of the purpose of the Snap! toolkit.

An example of a monolithic sensor input/output module is one connecting to a GPS receiver. Although the measurement variable `gps` may comprise many fields (gps.lat, gps.lon, gps.alt, gps.spd, gps.time, possibly even more), there is only one serial port connection to the actual receiver. This is a useful input/output module that will be incorporated into many Snap! networks.

A module containing transducers for wind speed, barometric pressure, temperature, and humidity moves towards the other end of the composition spectrum. The designer may choose to group the readings of all the transducers under the measurement variable `weather`, comprising weather.temp, weather.barom, weather.windsp, and weather.humid. Despite the grouping, the node actually communicates with four different transducers and conceivably four different interfaces. The conversion between the raw values of the transducer measurement and the reported values involve different formulae. Despite the differences in the interface complexity between the nodes with single and multiple transducers, they both appear the same to other Snap! nodes.
3.3.6.2.2 Actuator input/output modules

Actuator modules include modules that make a change in the environment. An actuator module may contain multiple transducers, organized in the same manner as those outlined in the discussion on sensor modules. One example of an actuator module is a motor that allows water to flow into a bed of plants. Another is an LED that illuminates to indicate that a target heart rate has been exceeded.

3.3.6.2.3 Display input/output modules

A display module is similar to an actuator module and there is no need to create a marked distinction between them. A display module may be used primarily for communicating information about the state of the network. An LCD on a sailboat that shows wind speed, current heading and speed, the positions of various sail controls, and the number of minutes into a race is a display module. A personal digital assistant that can be inserted or removed from the network in order to configure the individual nodes or verify the correct functioning of the network is another display module.

3.3.6.2.4 Logger input/output modules

Logger modules are used to store the data on a network. Logger modules either are the primary means of collecting the data from a system for later analysis or a backup in case a gateway module fails. A logger may be a flash memory device, a disk drive, or any other persistent medium. A logger might even keep a printed record of the data. A logger frequently functions by recording the data it receives in report packets. It may either wait for them to be generated by the nodes or it could actively poll the nodes using request packets.

3.3.6.2.5 Gateway input/output modules

A gateway module is used for transmitting data to and receiving data from a source external to the Snap! network. The gateway may be used for real-time monitoring of the sensor network, or to allow data to be analyzed by machines with greater computational capability with results returning to the Snap! network. A cellular modem and short-wave radio link are both examples of gateway modules.

The behavior of logger, gateway, and display modules may vary from that of a typical Snap! node. It may be enough for a system designer to configure the nodes to send report packets to a logger, without forcing the sensor nodes to query the logger as to the maximum value of its datastore measurement variable. On the other hand, a sensor network with extremely variable composition or with a particularly volatile network connection may have the nodes query the presence of gateway and logger nodes. It may be necessary to discover how much more data they can store or transmit. The exact input/output interface and the implementation and selection of measurement variables is at the discretion of the
individual node designer. The designer should keep in mind that for maximum reuse he should make the interface as general as possible. There is a constant tradeoff, though, between reusability of the modules and speed and ease of their development.

3.4 Summary

This section has explained the Snap! specification. It provided a functional description of the Snap! toolkit, elaborated the guidelines used in decision making during the design process, described the packet format and variable structure, depicted the separation of responsibility between the input/output module and the network module, and gave examples of different types of network modules and input/output modules. The next section will illustrate an implementation of the Snap! toolkit and a system built with it.
Section 4

A First Implementation

This section details the first realization of a system implementing the Snap! toolkit. The same printed circuit board contained both the network and input/output modules. However, the design allowed a system designer to cut the board in half, physically severing the two modules. Connectors could then be soldered to provided pads, allowing the two modules to function independently from their original fabrication. A separate power supply board attached to a single node. The supply provided regulated power to the nodes from battery input.

4.1 Preliminary information

4.1.1 PIC 16F876

The first implementation of Snap! used the PIC 16F876 from Microchip as the processor for the network and input/output modules.[19] The 16F876 is an 8-bit RISC processor. It has 8096 words of flash program memory, 368 bytes of RAM, and 256 bytes of EEPROM data memory. The '876 features 3 timers, a 10-bit analog-to-digital converter (ADC), a synchronous serial port (SSP) which can be used for I²C serial communication, a universal synchronous-asynchronous receiver transmitter (USART) which can be used for RS-423 communication, and many digital input/output pins. Custom Computer Solutions [9] produces a C compiler that targets many PIC microcontrollers, including the '876. Use of the compiler to allowed development in a readable and maintainable high-level language.

4.1.2 THE INTER-INTEGRATED CIRCUIT (I²C) COMMUNICATION SPECIFICATION

The first network modules used the Philips Semiconductor I²C networking specification to communicate.[25] I²C uses two signal lines, one each for clock and data. All devices are on the same two-wire bus. The specification allows either 7- or 10-bit addressing, supporting 127 or 1023 devices. The specification reserves a general call address to which all supporting devices respond in addition to their own. The I²C bus is a multi-master bus, allowing more than one device to control it. I²C detects contention when two devices attempt to control the bus simultaneously and defines an arbitration procedure to prevent data loss. The I²C signal lines are each normally pulled high through resistors. A device wishing to generate a low signal on the bus drives the line low; to generate a high signal it stops driving the line and allows it to be pulled high. Many manufacturers supply devices that communicate using I²C.
4.2 Network interface module

The first Snap! toolkit contained two separate network modules. Inadvertently, the need for the toolkit was reemphasized, as the two network modules were designed several weeks apart. The ease with which the two networks interoperated provided an indicator of the toolkit’s success.

**FIGURE 9: First implemented networks** A completely wired I²C bus and a hybrid I²C/wireless network comprised the first two implemented network interface modules.

4.2.1 WIRED

4.2.1.1 Hardware

The first network was based around the PIC 18F876 microcontroller. The 18F876's built-in I²C port, attached to a bus connecting the network modules of the Snap! nodes. This connection was augmented with a power line and required ground reference. The addition of a power line permitted the nodes to receive their power from the network bus. This meant that only one power supply board was needed for nodes that were proximate. The network modules communicated using a clock rate of 100 kHz. The general call feature was not enabled for this implementation. The network module attached to a logger input/output node usually had the network pull-up resistors, though any network module could have had them.

The network module linked to the input/output module with five lines: power and ground, I²C clock and data, and a traffic request line. Power and ground lines allowed an entire node to be powered from a single source, be it the network bus or a power supply board. The network module communicated with the input/output module via an I²C bus separate from that used for inter-node communication. The PIC on the network module used two digital input/output pins and software routines for this second bus. The interrupt request line was used by the input/output module to indicate that it had a packet to send.
Two light-emitting diodes (LEDs) and an additional connector for serial output, both controlled by the PIC, provided debugging information.

4.2.1.2 Software

With respect to the inter-node I2C network, the network module was normally in slave mode. In this mode, it waited to recognize its address on the network. Once it recognized its address, the module saved the entire packet to a buffer. When the packet was complete, the network module transferred it to the input/output module. The network module was always the master on the node's internal bus. In the first Snap! implementation, the network module could only buffer one packet at a time. It would not listen to the inter-node network until it had sent the packet to the node or a time-out had occurred.

The input/output module transmitted a complete packet to the network module in several transactions. When the input/output module had a portion of a packet to send to another node, it would raise the traffic request line connecting the two modules. Raising the traffic request line triggered an interrupt in the network module. The network module addressed the input/output module on their I2C bus and read the packet portion from the input/output module. When the input/output module had sent a complete packet, the network module entered the master mode and sent the packet on the inter-node I2C bus. The network module could only buffer one packet from the input/output module. It would not react to the traffic request line being raised until it successfully sent the current packet or a time-out occurred.

4.2.2 Wireless

The creation of a wireless network module eliminated cumbersome wires. A PIC 16F877, similar to the 16F876 described above, connected to the inter-node I2C network described above. The 16F877 also connected to a RF Monolithics TR1000 wireless transceiver.[31]

The network using the wireless network modules was actually a hybrid. A collection of nodes spaced only a few inches apart communicated via the wired I2C network described above. Each network module contained a table listing the nodes that were local to it. When a network module had a packet to send, it checked the destination address against the table to discover if the other node was local. If so, it sent the packet normally. Otherwise, the packet was sent to the wireless network module.

Upon receiving a packet from the I2C network, the wireless network module transmitted it. The module first determined whether any other wireless module was currently transmitting. When none were, the module began by sending a series of synchronization bytes to train receivers to the correct timing. Following these, it sent a start-of-packet byte followed by a network identification byte. The network ID
byte allowed several wireless networks to operate in proximity. Any wireless module receiving a different network ID differing from its own ignored the rest of the packet. The transmitter then sent the packet, followed by cyclic redundancy check (CRC) byte for simple error detection and noise rejection. The wireless network module then entered receive mode, waiting for data either from the I2C network or the wireless network.

The wireless network module was usually in receive mode. When the module began receiving synchronization packets, the circuit adjusted its timing. After receiving the start-of-packet byte and verifying the network ID, the module received and buffered the entire packet. The module then computed the CRC for the packet and checked it against the received CRC. If they matched, it transmitted the packet over the inter-node I2C bus to the destination node. Otherwise, it discarded the packet as corrupt. The wireless network module then returned to receive mode.

![Figure 10: Snap! node implementation](image)

The first Snap! implementation combined both input/output module and network module on one printed circuit board that could be cut in half to interchange components.

**4.3 Input/Output module**

**4.3.1 HARDWARE**

The PIC 16F876 also formed the core of the input/output module. The input/output module connected to the network module as described above. An 8-kilobyte serial EEPROM resided on the same bus that connected the two modules. The EEPROM supplemented the '876's paltry 368 bytes of RAM, allowing more data to be collected from sensors before transmitting packets.
The input/output module provided an expansion connector that connected to its transducers and actuators. The expansion connector provided power and ground lines directly from the input/output module as well as a switched power line controlled by the PIC. The jack contained pins connecting to the PIC: two for the USART, four analog-to-digital channels, one to the external interrupt pin, and one standard digital input/output pin. The USART, analog-to-digital, and external interrupt pins could all be reconfigured as normal digital input/output pins if their additional functions were not needed.

The input/output module also had two LEDs controlled by the PIC for displaying debugging information.

4.3.2 SOFTWARE

The software for the input/output module consisted of several core functions organized in libraries. These library functions made up the bulk of the code that resided on each input/output module. Additionally, each node contained specialized code for its particular transducers and actuators.

4.3.2.1 Network module interface code

One group of functions provided an interface to the network module, allowing discrete packet portions to be sent to the network module. The input/output module also implemented an interrupt routine for PIC traffic. When addressed by the network module, the PIC would vector to the routine. According to the function dictated by the network module, the routine would send the traffic that the input/output module had ready or receive a packet from the network module. After receiving a complete packet, the routine would set a flag to notify the main loop that a packet had been received.

4.3.2.2 Variable implementation

The first implementation of Snap! did not include all the variables specified in the design. The subset of implement variables comprised ds.id, ds.meas, bhv.0-2, measName, measName.format, and time. The variable values were stored in structures on the PIC, either in RAM or in the internal EEPROM. These structures were set when the PIC was first programmed, but could be dynamically altered through set packets or behaviors. Only the initial variable values had to be changed, so the code surrounding the accessing of these variables did not. One of the PIC's internal timers was used to implement the time variable.

The sensor nodes in first Snap! implementation used two broad types of measurement variables: those that contained a single value and those that contained an array of sampled values. When a variable with a single value was accessed, the input/output module would take a single reading from the transducer,
perform any necessary conversion and calculation on the value, and report it. In some cases, the transducer data was cached. When the variable was accessed, the current time was checked against the time at which the data was cached. If there was insufficient difference between the two times, the cached data was used; otherwise a new measurement was taken and reported. Variables with arrays of values operated differently. When the variable was accessed, the module began taking samples from the transducer at the configured sample rate. The node stored the samples in the external EEPROM. After completing the sample session, the input/output module incrementally read the stored samples from the EEPROM, creating and transmitting a packet for each sample.

Following initialization and bootstrapping, the input/output node entered an infinite loop. While in the loop, it would respond to incoming packets and run its behavior scripts.

4.3.2.3 Packet response

Upon notification through a flag that a packet had been received, the input/output module responded as described in the design specification. As noted above, only a limited set of variables were implemented. The modules ignored operations that referenced any variables they did not implement.

4.3.2.4 Behavior implementation

The first implementation of the Snap! toolkit contained a relatively simple behavior script. A behavior consisted of a triggering condition and a specification for a packet to be generated if the triggering condition was met. The triggering condition comprised a comparison opcode and, depending on the opcode, one or two operands. The supported opcodes consisted of two parts. One half specified a basic comparison: greater than, less than, equal, or not equal. The second half indicated what the operands were: two current variable values, a variable value and the value of the same variable when the behavior was last triggered, a variable value and a constant, or the difference between the current variable value and the value of the same variable when the behavior was last triggered and a constant. The packet specification consisted of a destination address, packet opcode, and variable name.

Each input/output module had the capability to run two or three behaviors. Unless other operations, such as responding to a packet or executing another behavior interfered, the behaviors ran once each second. By setting a behavior to trigger on the time variable differing from its last successful trigger by a certain amount, an input/output module reported automatically at regular intervals. This was the default behavior for the input/output modules in the first implementation of Snap! The report intervals varied according to the rate at which the transducers were expected to change, i.e. short intervals for a GPS receiver, longer intervals for temperature.
4.4 SWAT: Skiing With Advanced Technology

The implementation of the Snap! toolkit described above, without a system to apply it to, was little better than the specification in terms of demonstrating the usefulness of the Snap! concept. Cross-country skiers and their skis presented an intriguing and valid challenge for the Snap! toolkit.

4.4.1 SWAT MOTIVATION

Cross-country skiers and their skis represent a complex, dynamic system. This alone makes them a useful target for testing the implementation of a sensor network built using the Snap! sensor toolkit. Cross-country skiing, which has been used as a means of transport since pre-historic times, and draws a large recreational following today, has been seriously neglected in studies. A survey of articles on the topic of skiing in general reveals few papers. Most focus on the frictional and vibration dynamics of downhill skis on a short test run.

Currently, ski designers and equipment testers rely on laboratory tests and reports from people using the equipment in the field. While both of these data sets are useful, combining quantitative results from wired people using instrumented skis in the field provides much more precision than is currently possible. Additionally, data that may simply not have been available to designers before, such as whether a tester's tiredness is due to an inherent design flaw in the equipment or the skier's level of exertion the previous day, will help eliminate guess work. As described below, two domains of inquiry, as well as the challenge of measuring data not previously recorded, provided an ample test for the design requirements and functions of the Snap! toolkit.

4.4.1.1 The athlete

Cross-country skiing presents athletes with a range of distances similar to those found in long distance running events. Olympic distances range from five kilometers to the greater-than-marathon distance of fifty kilometers. As with any sport, dedicated athletes practice for hours each day in a quest for excellence. Several training aids currently exist to help maximize the effectiveness of this training. Heart rate monitors may be worn to roughly indicate the exertion level of an athlete. Training on machines can provide certain quantitative measurements of force and technique. However, no system yet exists to gather useful data about the athlete as he or she trains in the field.

Work done at Harvard University on the energetic cost of skiing relied on roller skiing and treadmills during experimentation. The conclusions state:
Although propulsive forces during roller skiing and snow skiing differ because of effects of wind resistance and the friction between the ski and the skiing surface, our results show that metabolic rate values depend on the application of ground force regardless of the amount of force required. Portable systems that measure forces and propulsive periods from the skis and poles could be used to estimate metabolic rates during skiing when collection of expired gasses is impractical. [5]

This example concretely states that a ski system such as SWAT would be of direct use to researchers studying skiing.

One important aspect of cross-country skiing is the athlete’s technique. There are two main styles of cross-country propulsion. The traditional diagonal stride places the skis parallel to each other. The arm of the ski being pushed backward is placed forward, diagonally opposite of the ski. A much faster technique, under constant refinement, is the skating stride. A skating skier sets the skis in a V-position, using the edges to drive him forward. The poles are either both used simultaneously and symmetrically or in a manner that more closely resembles the diagonal method of poling. As athletes train and refine their own technique, data about how a change effects the overall physical being of the athlete would be useful in determining whether a change has a net positive or negative effect. The holistic measurement of overall time around a track can only indicate so much.

Coaches and athletes alike often have only limited biometric data available to them. While secondary effects, such as tiredness, increase in heart rate, and poor training times for a given day, may point to a problem, it may difficult or impossible to diagnose. Biometric data collected in real-time may be able to help a coach or athlete tailor a particular part of a workout for maximum effectiveness. A system capable of indicating that an athlete is pushing the aerobic threshold on an easy day, or not pushing hard enough on a hard day, may provide the extra information needed to take the athlete to the next step of competitive excellence.

4.4.1.2 Equipment

A wide variety of skis are available for all sorts of snow types: short, metal-edged skis for backcountry touring; waxless skis for warmer, wetter snow; and waxable skis for those who prefer to optimize their gliding and gripping surfaces. The ski itself has a characteristic camber that determines how hard the skier must press down on the ski to cause the "kick zone" to catch the snow and move the skier forward. Debate as to whether longer, slimmer skis or shorter, fatter skis provide better performance remains unresolved. In a single system, fixed variables include length, width, camber, rigidity, and materials of the ski, as well as the length and stiffness of the poles. Any single skiing excursion adds air temperature, temperature, humidity, and compaction of the snow, and the choice of waxes applied to the base of the
ski for both grip and glide. This list includes only the most important, readily observed variables of the system.\[18\]

Changes in any one of the above variables can have an effect on important observed measurements of the skiing system. Perhaps chief among these is the speed at which the skier moves forward. This singular measurement may be further influenced by the amount of the ski in contact with the snow, the degree to which the ski grips the snow versus sliding along it, and the ease with which the ski maintains its forward path.

Research done at the University of California-Berkley and Institute for Medical Physics in Austria both relied on instrumented skis to measure the ski as a dynamic system and the characteristics of a ski gliding on water.\[2,26\]

To those who design skis, the ability to directly measure the effects of the design variables on the performance of the ski will allow for improvement of the scientific design process. Skis on the market today have remarkably similar designs. It is unlikely that uniform ski designs are equally suitable for men, women, and children. For those who sell skis, detailed measurements will allow a better match to an individual when selecting equipment. Being able to dynamically measure how differing cambers effect the stride of a potential customer will offer a great deal of improvement of the current technique of passing a sheet of paper beneath the skis while the customer is in the store. Finally, such measurements will again prove useful to the athlete. Being able to look at the current speed of the ski may improve enjoyment of the sport as well as provide further feedback about technique and the day’s wax application.

Finally, the ski system represented a valid use of the Snap! system as a tool for prototyping. As detailed in the composition section below, the platform comprised sensors of both broad and limited use. Sensors such as the heart rate monitor, temperature sensor, and barometric pressure sensor would undoubtedly be used in many systems. This required that they be constructed for reuse and configurable to their current system. The sensors on the ski, such as the boot angle or snow temperature sensors, would probably not be reused. Thus, it was important that they be easy and quick to build, so as not to spend too much time on a one-time use component.

4.4.2 COMPOSITION

The cross-country ski system was built in two steps. The first incarnation traveled to Iceland to be field tested. After examining the results of the first trip, the platform was revised to correct deficiencies in the first system and add new functionality. The second system underwent field testing in Norway.
4.4.2.1 Iceland system

One complete Iceland SWAT system was built. The system communicated using only the wired, I2C network. The first SWAT system had sensors on only one ski and one boot. Each of the nodes generated report packets at certain intervals. The nodes addressed the packets to the logger module, which was set to a predetermined address. The sensors did not communicate with each other, but only to the logger. The Iceland SWAT system contained the sensor nodes shown in figure 11.

<table>
<thead>
<tr>
<th>Sensor type</th>
<th>Variable type (single reading or sampled) and reporting interval</th>
<th>Sensor information and description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors carried in backpack</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heart rate monitor</td>
<td>Single reading / 30 seconds</td>
<td>Polar OEM module and coded chest strap [27]</td>
</tr>
<tr>
<td>External temperature sensor</td>
<td>Single reading / 4 minutes</td>
<td>National Instruments LM135 [23]</td>
</tr>
<tr>
<td>Barometric pressure sensor</td>
<td>Single reading / 2 minutes</td>
<td>Motorola MPX5100AP [21]</td>
</tr>
<tr>
<td><strong>Sensors mounted on ski</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Orientation</td>
<td>Single reading / 30 seconds</td>
<td>Precision Navigation TCM2 provided compass direction, pitch, and roll of the ski [28]</td>
</tr>
<tr>
<td>Accelerometers</td>
<td>17 readings over 2.25 seconds / 30 seconds</td>
<td>Analog Devices ADXL210AQC - Two two-axis accelerometers were gave acceleration in three directions: parallel to the ski length, parallel to the ski width, and vertical [4]</td>
</tr>
<tr>
<td>Toe angle</td>
<td>16 samples over 2.10 seconds / 30 seconds</td>
<td>Interlink Electronics #402 force sensitive resistors and conversion circuit - Measured the of the boot to the ski base [15]</td>
</tr>
<tr>
<td>Boot flex</td>
<td>16 samples over 2.10 seconds / 30 seconds</td>
<td>Abrams Gentile Entertainment bend sensor and conversion circuit - Measured the angle where the boot flexed, between the ball of the foot and the base [1]</td>
</tr>
<tr>
<td>Heel pressure</td>
<td>16 samples over 2.10 seconds / 30 seconds</td>
<td>Interlink Electronics #406 force sensitive resistor and conversion circuit - Measured the amount of pressure applied to the heel of the boot</td>
</tr>
</tbody>
</table>

**Figure 11: Iceland SWAT system** The Iceland SWAT system comprised these sensors.
The Iceland system used a PalmIIIx personal digital assistant (PDA) [24] to log the data reported by the sensors. An input/output module was configured to send the packets it received over the serial transmit line in the expansion connector. The PDA’s serial port connected to the expansion connector. The PC-to-serial node achieved rough time stamping by sending a fixed string once every ten seconds. A terminal emulation program, Online, wrote the data received over the serial port to a file on the PDA. The contents of the file could later be downloaded to a personal computer for further analysis. The PDA contained 4 megabytes of storage, enough to store all the data recorded in one day.

<table>
<thead>
<tr>
<th>Sensor type</th>
<th>Variable type (single reading or sampled) and reporting interval</th>
<th>Sensor information and description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sensors carried in backpack</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Heart rate monitor</td>
<td>Single reading / 30 seconds</td>
<td>Polar OEM module and coded chest strap</td>
</tr>
<tr>
<td>External temperature sensor</td>
<td>Single reading / 30 seconds</td>
<td>National Instruments LM135</td>
</tr>
<tr>
<td>Barometric pressure sensor</td>
<td>Single reading / 30 seconds</td>
<td>Motorola MPX5100AP</td>
</tr>
<tr>
<td>GPS receiver</td>
<td>Single reading / 30 seconds</td>
<td>Trimble Lassen board with 28367-00 antenna - reported latitude, longitude, altitude, horizontal speed, altitude, and time of day [36]</td>
</tr>
<tr>
<td><strong>Sensors mounted on ski</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compass heading</td>
<td>Single reading / 30 seconds</td>
<td>Precision Navigation Vector 2X</td>
</tr>
<tr>
<td>Snow temperature</td>
<td>Single reading / 30 seconds</td>
<td>National Instruments LM135 - mounted on only one ski per pair</td>
</tr>
<tr>
<td>Accelerometers</td>
<td>75 readings over 3.75 seconds (20 Hz) / 30 seconds</td>
<td>Analog Devices ADXL210AQC - Two two-axis accelerometers were gave acceleration in three directions: parallel to the ski length, parallel to the ski width, and vertical</td>
</tr>
<tr>
<td>Toe angle</td>
<td>75 samples over 3.75 seconds (20 Hz) / 30 seconds</td>
<td>Interlink Electronics #402 force sensitive resistors and conversion circuit - Measured the angle of the boot to the ski base</td>
</tr>
<tr>
<td><strong>Sensors mounted on boots</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Boot flex</td>
<td>75 samples over 3.75 seconds (20 Hz) / 30 seconds</td>
<td>Abrams Gentile Entertainment bend sensor and conversion circuit - Measured the angle where the boot flexed, between the ball of the foot and the base</td>
</tr>
<tr>
<td>Heel pressure</td>
<td>75 samples over 3.75 seconds (20 Hz) / 30 seconds</td>
<td>Interlink Electronics #406 force sensitive resistor and conversion circuit - Measured the amount of pressure applied to the heel of the boot</td>
</tr>
</tbody>
</table>

**FIGURE 12: Norway SWAT system** The Norway SWAT system comprised these sensors.
4.4.2.2 Norway system

Two Norway SWAT systems were built. One system used the same network as the Iceland system, completely wired FC. The second system used the wired/wireless hybrid networking scheme. In the Norway system, each ski and boot contained sensor nodes. As with the Iceland system, the nodes addressed all their packets to a predetermined logger address. The Norway system contained the sensor nodes shown in figure 12.

In order to make the collected data more useful, the timing of the measurements from the ski and the boot (accelerometers, toe angle, boot flex, and heel pressure) was coordinated. The left ski accelerometer node served to synchronize the others: when its measurement variable was accessed, it first sent an acknowledgment packet, with its payload set to the SYNC value, to the other sensors. All the coordinated nodes waited 50 milliseconds after receiving the synchronization packet. Then they all took their samples as configured and reported. As long as the interval between two synchronization packets was less than 40 seconds, each coordinated sensor would only measure and report with the others. The coordinated nodes would report spontaneously if they did not receive a synchronization packet. While this reduced the usefulness of the collected data, it eliminated the left acceleration node as a single point of failure.

![Figure 13: Norway SWAT system diagram](image)

FIGURE 13: Norway SWAT system diagram Three different containers held the sensors for the Norway SWAT system: one on each ski, one on each boot, and one in the backpack.

The Norway SWAT system used a custom designed logger. The logger contained a PIC 16F874, similar to the PIC 16F876 described earlier in this section. The 16F874 connected to the inter-node FC network and a SmartMedia holder.[33] SmartMedia cards are about the size of two postage stamps, wafer-thin, and can store between 2 and 32 megabytes. Since SmartMedia cards use flash memory, they provide non-volatile storage. A 4 megabyte SmartMedia card stored data from a complete day of skiing. The logger
stored all packets addressed to it. The logging board could be removed from the network and the data transferred via RS-232 connection to a personal computer for further analysis.

4.5 Summary

This section detailed the first implementation of the Snap! toolkit, including the software and hardware components of both the network modules and the input/output modules. It also described the example platform that was used to test the Snap! toolkit concept. It elucidated the motivation for the Skiing With Advanced Technology system. The section detailed the Iceland and Norway SWAT systems. The next section describes the results of the Iceland and Norway field tests, evaluates how Snap! met its design goals and functionality, and draws conclusions about the Snap! research.

FIGURE 14: SWAT system The author models the SWAT system he wore in Norway. Notice the wire trailing from backpack to pants and boxes for power and ski sensors mounted on the skis.
Section 5

Results and Evaluation

Just as the implementation of the Snap! system without applying it to a specific sensor networking problem would have failed to demonstrate its usefulness, so the creation of the Skiing With Advanced Technology (SWAT) system would fail to demonstrate the success of Snap! without field testing. SWAT underwent two rounds of testing. The first round occurred in late February, 2000, in Iceland. After analyzing the results of this trip, the knowledge gained went into revising and enhancing SWAT. The second SWAT system was tested in mid-March, 2000. The previous section describes the SWAT systems in detail.

5.1 Field Test: Iceland

The Iceland field test comprised four separate data-gathering sessions. Professor Michael Hawley wore the system for all four sessions. No data were successfully gathered during the first two sessions, due to hardware and software errors described below.

After correcting the bugs, the last two sessions were conducted. The first test session was a short, ten-minute jaunt to verify that SWAT was successfully recording data. The second session was a 2 hour 15 minute trek covering about 5 kilometers. The trail was unbroken, but the instrumented skier did not break trail.

5.1.1 What went wrong

SWAT did not successfully log data during the first two data-gathering sessions. Errors in software prevented packets from successfully arriving at the logger. Additionally, vibration loosened connectors and batteries, causing the nodes to behave erratically. Professor Hawley also demonstrated one of the disadvantages of a wired system. During a long stride, one of Mike's skis crossed over the wires connecting the boot to the ski. The wires severed, preventing further data collection from the boot.

Of the sensors that made up the first implementation of SWAT, only two produced data that were both valid and useful: the accelerometers and the heart rate monitor (figures 16 and 17). The rest of the sensors had either hardware or software errors that were revealed during the field test. The data from the barometric pressure and temperature sensors lacked the most significant byte of a two-byte value, rendering their readings meaningless. The wires connecting the transducers in the boot to their node
became damaged before any readings could be recorded from them. The amplifier for the toe angle was not optimally tuned. This led to values that indicated whether the boot was off the ski or on it. The ski orientation node was originally included to provide auxiliary data for a GPS receiver. The data from the ski orientation sensor appears to be valid. Without a GPS receiver, however, and configured to report only a single sample every 30 seconds, the data collected from the node could not lead to any conclusions about the ski dynamics.

The data collected from the accelerometers revealed another deficiency in the system. The input/output modules on the first SWAT system were restricted to the RAM available on the 16F876. This limited the amount of data that the modules could collect and transmit in single sample session. The sampling frequency was insufficient for the rate at which acceleration and the boot angle were changing. Thus, while general characteristics of the ski stride are visible, such as when the skier began or finished a stride, more detailed analysis, such as slippage or distance traveled, is inconceivable.

Finally, a significant limitation of the first version of SWAT was that only one ski was monitored. This meant that very basic information about the skier's stride was not certain, such as if he was following the classic diagonal style, double poling, or skating. The skier's balance and distribution of effort could not be assessed. Overall, the data collected from the single-ski SWAT system was inadequate to provide anything more than a superficial description of the action of the skier and his equipment.

5.1.2 WHAT WENT RIGHT

Despite the deficiencies of the data collected, by the end of the first field test several of the Snap! nodes were functioning as designed. The system collected data reliably from all of the nodes connected to the network; the problem was with the transducers connected (or not connected) to the input/output modules. The log file revealed that at several points during the second test run some of the nodes reset, essentially dropping off the network and reappearing. As designed, the network continued to function correctly whenever these unplanned network reconfigurations occurred.

![Figure 15: Accelerometer directions](image)

The accelerometers recorded three orthogonal axes as labeled here.
FIGURE 16: Iceland SWAT system acceleration The Iceland SWAT system successfully recorded these acceleration data during one sample session, illustrating one stride. The low sampling rate and short sample session time make produce a low resolution graph.

The data collected from the accelerometers are not completely without merit. Figure 16 shows one sample group taken from the final test session. The large increase in lengthwise acceleration indicates that the skier began his stride with a rapid forward push. The deceleration as the ski came to rest was not as significant. The change in acceleration in the vertical direction is very slight. This indicates that the skier was most likely not planting the ski sufficiently or that snow conditions were such that a hard plant was unnecessary. A graph of heel pressure would have helped draw the conclusion that the latter conclusion was probably correct: the freshly fallen snow and near-melting temperatures made the snow rather sticky for the depicted sample session.

FIGURE 17: Iceland SWAT system heart rate The heart rate graph shows that Professor Hawley alternated between rest and the target aerobic heart rate zone for his age.
The heart rate graph in figure 17 gives an indication of when the group took breaks from skiing to either observe and document the surroundings or verify the correct operation of the equipment. The high end of the heart rate is around 135 beats per minute, or 75% of the skier's maximum heart rate. This rate, within the aerobic conditioning zone for Professor Hawley, indicates that the pace was appropriate for a short workout but not for three hours of sustained tour skiing. This may explain the frequency and duration of the rest stops.

5.1.3 LESSONS LEARNED

The Iceland field test provided the first shakedown of the Snap! implementation. It revealed several hardware and software bugs. Some were fixed in Iceland, others were noted as problems to be solved before the second field test.

Mounting sensors in proximity to snow presents several challenges, such as providing a waterproof case that also allows wires to pass through and keeping the electronics within their operating temperature range. The first SWAT system successfully met these challenges. The box containing the electronics was so bulky, however, that it was uncomfortable for the skier and occupied more space than necessary. Additionally, the wires forming the sensor network proved to be cumbersome to connect and disconnect. They were uncomfortable for the skier and required extra care to prevent breakage. This discomfort and vulnerability to failure provided motivation for developing a wireless network for the revision of SWAT.

5.2 Field Test: Norway

The Norway field test comprised two data gathering sessions. Two SWAT systems were ready for the test; they were identical except for the network modules and connectivity. Professor Michael Hawley wore a system that used the hybrid I2C/wireless network described in Section 4. Matthew Debski, the author of this thesis, wore a system that used I2C as the only connectivity. Both systems employed sensors on both skis and boots.

The data gathering sessions consisted of two treks, each 24 kilometers long. The marches were an out-and-back loop to a remote cabin for nighttime accommodations and repair of equipment. The terrain varied from flat to large hills, with one sustained ascent approximately 5 kilometers long. Snow conditions changed from almost pure ice to well-packed trail. Light conditions varied from full-daylight to aurora and moon-lit.

---

1 The target heart rate for an aerobic workout is 60%-80% of maximum. A rule-of-thumb formula for determining maximum heart rate is \((220 - \text{age in years})\) beats per minute.
Both Snap! systems functioned for both data gathering sessions.

5.2.1 WHAT WENT WRONG

While both Snap! systems successfully recorded information during both sessions, they did not perform completely as desired.

On the entirely PC system, problems with the cable between the boot flex and heel pressure transducer on one boot prevented collecting any valid information from that node. Additionally, the wire connecting one of the toe angle transducers to its input/output module severed.

The wireless system had a different set of problems. Analysis of the log files indicates that each of the local networks communicated successfully. Also, the networks on the skis and boots communicated among each other, as the packet used to synchronize the sampling sensors on the skis and boots was successfully received by the nodes. However, the distance to the backpack transceiver as well as skier's body being placed in the communication path prevented the data being received by the wireless node in the backpack. Since the logger was in the backpack, the data from the skis was not logged.

Finally, the second version of SWAT featured nodes were enhanced with an addition EEPROM chip. For nodes that sampled their transducers, this allowed a higher sampling rate and a greater number of recorded samples. The larger number of data, in turn, allowed finer analysis of the information returned from these sensors. On the other hand, analysis of the data reveals that the sensors were still undersampled. The sensors sampled at 20 Hz. Figure 22 shows the data collected from the length and height accelerometers. The waveforms appear choppy; the data from the height axis seems to show a frequency of about 10 Hz. The Nyquist criterion requires a sampling frequency of twice the signal being sampled for accurate reconstruction. The 20 Hz frequency just meets the criterion. To observe the more subtle details of the data, a sampling rate of 50 Hz should have been used. Whether the input/output and network modules would be capable of handling this rate is unknown.

5.2.2 WHAT WENT RIGHT

Many of the weaknesses in the section described above involved specific sensor nodes: for example, the group of sampling sensors was not set to sample quickly enough. A cable inside the boot module was not connected properly. Generally, however, the SWAT system functioned as designed.

A cursory analysis of the results from the two systems suggests that the SWAT system using the wired network, which managed to log the majority of its data, is the correct choice for future systems. In reality,
improvements in the wireless network design are likely to render the wired system unnecessary. The wired system is vulnerable to a variety of physical assaults: water, snow, dirt, and dust clog connectors; a sudden fall may sever or dislodge the wires; the cables generally interfere with a complete range of movement. Thus, even the limited success of the wireless system well indicates that the wired system may eventually be discarded.

5.2.3 ANALYSIS OF COLLECTED DATA

Following the Norway field test, a data browsing application was created. The application used the time-stamped series of GPS coordinates superimposed on a map to show the path of the instrumented skier. Choosing a specific point brought up a window showing graphs of the sampled data (accelerometers, boot flex, heel pressure, and boot angle) as well as displays of the static data at that point. Using the application, the characteristics of the skier’s stride could be monitored over the course of the entire test.

FIGURE 18: SWAT browser The application shown here allowed browsing of the data collected during the Norway field test. The background shows the path taken by the skier, as recorded by the GPS receiver. Choosing a point activates the foreground display, which shows graphs of the accelerometer, toe angle, boot flex, and heel pressure, as well as the single value time, heart rate, ski direction, barometric pressure, and temperature.
This discussion concerns the data from the point shown in figure 18. The same analysis can be applied to any of the points for which data was collected.

![Ski measurements](image)

**FIGURE 19: Ski measurements** The ski and boot sensors measured toe angle, boot flex, and heel pressure as depicted here.

5.2.3.1 The skier’s stride

Figure 20 shows the toe angle over a 3.75 second sampling session. As can be expected, the toe angles alternate between the times they each reach their maximum values. The maximum value of toe angle occurs when the striding leg is completely extended and the skier shifts his weight to the other ski. This weight shift allows the extended ski to lift up and the skier brings it forward. The stride rate can be determined by measuring the time between the two successive peaks from the ski. This particular sample shows that the strides are evenly spaced. However, the skier does not reach the same maximum value in each of the strides indicating that he may lack consistency in stride length.

5.2.3.2 Comparison of toe angle and boot flex

Figure 21 shows the relation between toe angle and boot flex from a single foot. As the skier extends his rear leg, the boot and foot begin to bend before there is an angle at the toe. This demonstrates the heel being lifted from the ski earlier as well. The boot flex also lags behind the toe angle somewhat after the front of the foot is flattened again. Further study of this graph, in combination with a video analysis, could help the skier optimize the length of his stride. Also, combining this graph with heel pressure could reveal if the skier is transferring his weight in the most effective way.
5.2.3.3. Ski movement

Figure 22 depicts the accelerometer data taken from the left ski during a single sample session. The large values correspond to the lengthwise movement of the ski. These correspond timewise to slightly after the maximum toe angle; after the leg is fully extended, it is brought swiftly forward during the stride. Velocity and position measurements can be computed by integrating the lengthwise curve. Both would be helpful.
in determining the amount of backwards slip the ski undergoes, an indication of a flaw either in waxing technique, choice of wax, or in the weight distribution of the skier.

The other two values are not perfectly constant. Acceleration along the width of the ski indicates that there is wobble or force being exerted other than in the intended direction of motion. Excessive movement in this direction, not indicated here, would indicate that the skier was wasting energy with an inefficient stride.

The final vector to consider is height. The acceleration due to gravity causes the constant offset that is seen in this direction. These data are especially interesting for two reasons. Most importantly to the technique of the skier, it indicates the force and timing of the lifting of the ski as it is brought forward. At the end of the forward movement, it also demonstrates the planting of the ski. Any upward force exerted while the ski is on the ground could lead to slippage in the lengthwise direction.

![Graph of ski motion through acceleration](image)

**FIGURE 22: Ski motion through acceleration** The acceleration data can be integrated to determine efficiency. The dashed lines demarcate the vertical resonation present as the skier lifts the ski to move it forward.

The series is also useful to the ski designer. As the ski is picked up, it begins to resonate. This is demonstrated by the fixed frequency of the waveform present in the height graph while there is any acceleration in the lengthwise direction. When the lengthwise acceleration is constant, so is the vertical acceleration. The ski designer can use the information about vibration to choose appropriate materials or evaluate different designs.
The collected data and the analysis presented illustrate the success of the SWAT system in accomplishing the purpose for which it was designed. It is possible to analyze stride components using time-synchronized videotape. However, the quantitative results obtained by the Snap!-enabled SWAT system previously required a customized measurement system. The ski vibration data and the possibility of precise slippage calculation demonstrate some of the value that the SWAT system presents to skiers and ski manufacturers.

5.3 Conclusions

The implementation of two versions of the SWAT system afforded an opportunity to truly evaluate the Snap! system's performance and meeting of the design goals. The design time for the Iceland field test included not only the design and construction of the individual sensor nodes and associated software, but also the time to implement the first version of Snap!. In the three weeks between the Iceland and Norway field tests, the first Snap! implementation was enhanced based on the lessons learned from Iceland, new transducers were created and their associated software functions written, and the wireless network was designed from scratch and integrated into the system. The fact that so much was accomplished in so little time is itself testament to Snap!'s success in meeting its design goals.

5.3.1 SNAP! WEAKNESSES

Overall, Snap! achieved the goals that it set out to accomplish. However, the system was not flawless. The incarnating of one Snap! version and its testing it in two systems revealed some general weaknesses in the design and concept.

5.3.1.1 Use of ASCII

Perhaps the single largest engineering flaw in the system was the choice of ASCII as the means of communicating between nodes. ASCII was chosen to reduce the time spent debugging. The ability of a Snap! constructor to view and understand the traffic on the network should help him determine where bugs occur. Network errors would result in garbled text; mistakes on the input/output module would result in valid characters but bad values.

In reality, the use of ASCII seemed to cause more harm than good. Good engineering practice dictates debugging components separately, then combining them and debugging the system. Thus, a designer should be able to debug the network, ensuring that the network modules are communicating successfully. Then, he can plug the input/output modules into the network modules and test the system again. The choice of ASCII attempted to create a short cut. Instead, the code on the input/output modules became unreasonably complicated. For instance, the data from the orientation sensor for the Iceland SWAT
system was encoded in ASCII. In order to store sufficient samples on the input/output module for transmission, as well as perform calculations on the values, the data was converted to integer values and stored. Then, to comply with the Snap! specification, the stored values were converted to ASCII again before transmitting them. A large percentage of code space was used for the conversion functions.

As has already been mentioned, the value of the report field can be used to carry binary data. Another alternative could be to expand the packet definition to include typing of the data in the report payload fields. This would allow the input/output module to report data in the format that is most natural for it, while still allowing other modules to interpret the data.

5.3.1.2 Overkill of Snap!

The Snap! system is designed to allow users almost unlimited flexibility in configuring their system. Even an implementation using the minimal PIC can change dynamically from a master-slave configuration, to publish-subscribe, or to autonomous reporting. The nodes can communicate with each other frequently or never. Network communication can be constant and guaranteed or only intermittent. Systems can contain redundant loggers, multiple displays, and gateways with different connectivity or a single logger and a dozen sensor nodes. The saying "Jack of all trades, master of none," may be applicable to Snap! The system, in providing the flexibility for all of these configurations, also means that it cannot specialize too highly in any single area. For the SWAT system, only one configuration was used. There was almost no need to place the system in any other configuration.

Nevertheless, it must be reemphasized here that Snap! for prototyping different sensor networks quickly. Thus, a user can place the nodes into several different configurations to determine which is the best for a particular application. If the designer then needs a more permanent specialized solution, he can design a custom solution to the problem. Further application of Snap! will reveal if its flexibility is an asset or a liability.

5.3.2 Snap! Contributions

The Snap! toolkit provides a means of building sensor networks rapidly. In some design aspects, the system departs from existing systems, while incorporating new ideas and drawing in good concepts from other systems. The result is a toolkit that fills a niche not adequately served by other networking technologies. The main contributions of Snap! are its flexible configurations, aggregating data format, and changeable behaviors.
5.3.2.1 Flexible configurations
The Snap! toolkit does not require that its nodes or modules consist of specific hardware. As long as the modules and nodes function as specified in section three, they form an implementation of the Snap! toolkit. The nodes can be implemented using Pentium processors, custom fabricated integrated circuits, or PICs, as in the example implementation. This allows the node designer great flexibility in choosing hardware appropriate for the particular node task.

5.3.2.2 Aggregating data format
The hierarchical data structure used to communicate across the network and within the node behaviors gives node designers the benefit of abstraction barriers and aggregation. As the data passes through different levels of the network, nodes can create new data types, consuming older ones. Including this idea at the packet level and using ASCII for communication allows easy implementation of a standard technique across engineering design.

5.3.2.3 Changeable behaviors
The dynamically changeable behavior of each node provides the largest contribution. Input/output modules can provide sensor fusion by communicating with other nodes. Nodes can react to the sensed data by changing report rates, setting actuators, or going to sleep. The changeable behaviors of the Snap! system are what truly give it the power and flexibility to meet its functionality requirements and design goals.

5.4 Future work
The current implementation of the Snap! toolkit is not complete. Some of the ideas discussed during the design and specification of the toolkit were not fully laid out in the specification. Also, the Snap! system, designed to be a toolkit for building sensor networks, has only been used to construct two complete systems, one of which was an improvement of the other. There are several areas which further development of the Snap! architecture can explore.

5.4.1 Extension of SWAT
SWAT served as the first implementation of the Snap! toolkit. As such, most of the work and research surrounding it was devoted to testing of Snap! itself. The purpose of Snap!, however, is to enable the construction of networks such as SWAT. Thus, given that a system for recording a great deal of information from a cross-country skier and his skis is available, it is possible to conduct experiments previously not possible. Section one described skiers in the Birkebeiner wearing systems designed to
compare elite athletes to novices, providing both with feedback on how to improve. SWAT makes such a study possible. Another extension would be to outfit several different brands and designs of skis with the SWAT system. Allowing skiers of different body types and skill levels to use the systems before purchasing skis would aid them in their choice of model and likely increase their enjoyment of the sport.

5.4.2 FULL IMPLEMENTATION OF ALL OPCODES AND VARIABLES

As noted in section 4, the first implementation of Snap! did not include all the variables laid out in the design; only a set needed to create a functional implementation was included. A thorough examination of the entire Snap! design would be aided by the addition of the extra variables. The extra variables allow the system to better adapt to changing conditions. They also make possible additional types of Snap! nodes that are not possible using the current limited implementation.

One of the most significant changes to the Snap! implementation is the inclusion of the Find opcode. The Find opcode is included in the description of the Snap! architecture in section 3. Allowing nodes to query the network to discover nodes of certain types or properties potentiates the building of more intelligent systems. Systems with self-discovery can be capable of dynamically reconfiguring themselves or altering their behavior based on their composition.

The addition of the Find opcode and the refinement of its definition reside in a broad area of research. Both Sun Microsystems's Jini [34] and the MIT Media Lab's Hive [20] include methods for discovering the behavior of nodes on a network. Work in this area would not only affect Snap! but would likely impact networks generally.

5.4.3 ADDITION OF PLANNED PERIPHERALS

The Snap! toolkit design does not dictate which particular nodes comprise a sensor network. The two SWAT systems consisted exclusively of nodes that either sensed or logged data. Implementing two other specific types of nodes would greatly expand the usefulness of almost any network that incorporated them: a long-distance communication node and a Configurator node.

A long-distance communication, or gateway, node would allow a Snap! network to communicate with other types of networks. One example is a cellular modem that sends the packets to a host on the Internet. Once received, the host could publish them on the World Wide Web for real-time viewing. The gateway should be bi-directional, allowing viewers to communicate to the Snap! network. Thus, a Snap! weather station in a remote location could indicate that its batteries were running low. Remote users could then decide whether to activate a secondary power source remotely or actually visit the station to
assess conditions. Coaches monitoring athletic performance could send feedback on how the athlete could best modify his tactics given current physiological conditions.

A Configurator node could be based upon a Personal Digital Assistant (PDA). The Configurator would attach to the Snap! network whenever its designer wished to view its current state. The Configurator would have the ability to access the variables of each node and display them. The Configurator could also download new behaviors in whatever language is appropriate for the Snap! implementation. Attachment could occur when first applying power to verify that all nodes are working correctly or during operation to determine the current readings reported by the sensors. The Configurator could also be left attached for a greater duration, serving as a general-purpose display to a network user or wearer.

5.4.4 MIGRATION TO OTHER PLATFORMS

The current Snap! implementation uses the relatively low-computation-power PIC16F876. Some of the implemented nodes on the SWAT system, such as the accelerometers, stretched the available RAM and code space to its limits. While the code on this node certainly could have been tightened, Snap! is supposed to facilitate rapid production of prototype networks. Forcing a Snap!-network designer to wrangle his code into a space that is too small does overlap with this goal.

Additionally, other input/output modules may require more computation power, such as one that interfaces to a digital still or video camera. Furthermore, a designer who wishes to explore the ability to use mobile code or emergent behaviors of the network modules may wish to use a behavior language that is more complicated than one capable of being implemented on a PIC.

Snap! is designed to be flexible yet powerful enough to be implemented on a variety of hardware systems. Implementing a second, more computationally powerful version of Snap! would provide a further test of the design, while also enhancing the collection of nodes available to network creators.

5.4.5 USE FOR OTHER APPLICATIONS

Using the Snap! toolkit to create other networks is the best test of its design. The toolkit is designed to facilitate the rapid construction of networks. If no networks besides SWAT are ever built using Snap!, then the system is almost useless. Certainly the construction of SWAT as a single-purpose system, without creating the general Snap! hardware and software would have taken less time. The greatest area of future work, then, is to use Snap! to build multitudes of sensor networks, determining their successes and failures, and pushing the architecture to new heights.
Appendix

Further information about the Snap! Toolkit for developing sensor networks, including implementation details and suggestions, is located at http://www.media.mit.edu/pia/Research/ESP/
Bibliography


http://wwwansi.org/

http://www.analog.com/

http://jap.physiology.org/

http://www.media.mit.edu/c2c/

http://www.edn.com/embapps/emb041.htm

http://www-mtl.mit.edu/research/icsystems/uamps/

http://www.ccsinfo.com/

http://www.echelon.com/

http://www.fitsense.com/


http://www.interlinkelec.com/


[33] Simple Technology: SmartMedia Cards.
    http://www.simpletech.com/flash/smartmed.htm

[34] Sun Microsystems. Jini Connection Technology.
    http://www.sun.com/jini/

[35] TC-9, Committee on Sensor Technology of the IEEE Instrumentation and Measurement Society. 
    IEEE 1451.2: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators -
    Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet
    (TEDS) Formats. USA. 1999.

    http://www.trimble.com/