Wireless Technology to Control and Monitor Anything from Anywhere™

Synapse's SNAP Network Operating System

Today we are surrounded by tiny embedded machines – electro-mechanical systems that monitor the environment around them and issue commands to control other machines and systems. One such machine may be monitoring the ambient light and checking for motion, poised to activate a lighting system. Another might be observing the operating characteristics of a motor and controlling its speed. Yet another may be watching and adjusting the temperature and humidity.

Consider a collection of such machines in a mall, or an office, or a factory, or... well, anywhere, really. Individually, each of these machines is extremely useful. Imagine how much more useful they would be if they could communicate with each other, perhaps passing short messages back and forth saying what they were observing and what they were doing, thereby allowing their companions to make more intelligent decisions. Suppose the machines in one building could communicate with the machines in another building on the same campus. What if any group of machines could communicate with other group or groups anywhere in the world via the Internet? And suppose humans could converse with any and all of these machines from anywhere in the world by means of a standard Internet browser.

In fact, all of this machine-to-machine (M2M) and machine-to-human (M2H) communication is now possible using inexpensive, easy-to-use, state-of-the-art technologies from Synapse Wireless...

Introducing SNAP

Let's start with SNAP, which is a network operating system that can run on any computing platform, from tiny embedded controllers, to notebook and desktop computers (running the Windows, MAC, or Linux), all the way up to massive Internet and Cloud servers.

One of the key features of SNAP is that it performs full mesh routing using any and all available communications interfaces. Employing standard protocols, SNAP fully exploits the communications capabilities made available by the device upon which it is running. These protocols may include IEEE 802.15.4, WiFi, TCP/IP, Ethernet, USB, RS232, RS485, and so forth.

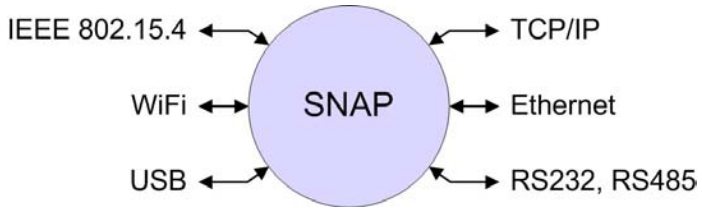

Figure 1. SNAP performs mesh routing using any and all available communications interfaces.

When an instance of SNAP is first invoked, it is informed as to which communications interfaces are available to it. The SNAP instance immediately starts "pinging" and monitoring these interfaces to see if any other SNAP instances are in the vicinity. As soon as two or more SNAP instances establish communication, they form a mesh network. As more and more SNAP instances are activated, they automatically integrate themselves into the network.

Self-forming, self-healing SNAP-based networks do not require a central controller (such a controller can form a single point-of-failure in conventional network solutions). All SNAP-enabled devices perform full peer-to-peer mesh routing. If one device fails for any reason, the other devices will automatically route messages around the failed unit.

Introducing SNAPpy

In addition to performing mesh-routing, the SNAP network operating system can also run user-created applications (in this context, the term "user" refers to whomever is developing applications to run on the network). These applications are captured in the high-level Python scripting language by means of an intuitive, easy-to-use development environment called Portal.

The term SNAPpy is used to refer to the combination of SNAP and Python. A SNAPpy application is captured in the form of script that calls a suite of small, parameterized functions. These functions can also call each other, and functions in one SNAP node may call functions in other nodes.

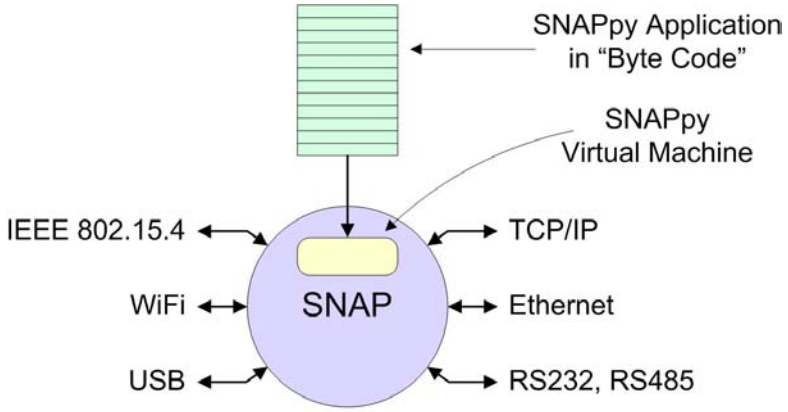

Figure 2. SNAPpy applications are executed by a SNAPpy virtual machine.

SNAPpy applications are automatically translated into what is known as "byte code", which may be downloaded into one or more SNAP nodes. As illustrated in Figure 2, each SNAP node includes a SNAPpy (Python) Virtual Machine, which executes the byte codes forming the SNAPpy applications. This provides extreme portability, because the SNAPpy Virtual Machine provides a layer of abstraction that separates the applications from the physical hardware. This means that a SNAPpy application executable will immediately run on any processor without requiring any modification or re-compilation.

SNAP-Based Wireless Networks

SNAP has been ported to a wide variety of low-cost, low-power microcontrollers that can be used in wireless modules [in some cases the microcontroller chip may also include the Radio Frequency (RF) functionality]. For the purpose of this portion of our discussions, we will assume that only the IEEE 802.14.4, USB, and serial (RS232 or RS485) communications interfaces are enabled. A SNAP instantiation in such a wireless module has a small footprint both in terms of memory size (only 45 kilobytes) and computing requirements (SNAP can run on 8-bit microcontrollers and higher). This means that only a single processor is required to handle both the wireless communications and to run any SNAPpy applications.

A Hardware Abstraction Layer (HAL) is used to interface SNAP to the outside world in the form of its microcontrollers' input and output pins, which are used to monitor sensors and control actuators as illustrated in Figure 3. The HAL allows SNAP to be quickly and easily ported to different microcontrollers as required.

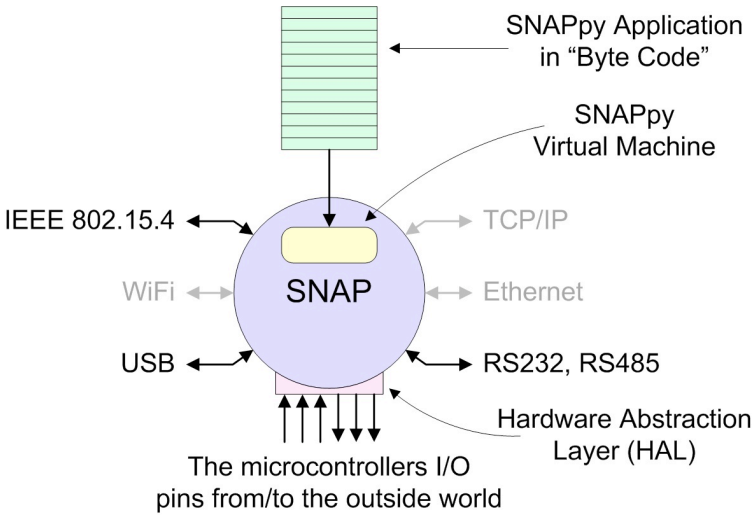

Figure 3. A Hardware Abstraction Layer (HAL) interfaces SNAP to the outside world.

For the purposes of simplicity, let's assume that we have four SNAP-based wireless modules (in reality, each SNAP-based network can support thousands or millions of nodes). Although these modules may have USB and/or serial ports, initially we will assume that nothing is connected into these ports. In this case, the only communications channel available to these modules is their IEEE 802.15.4 wireless interface.

As each node is powered-up, it automatically integrates itself into the network as illustrated in Figure 4. Some SNAP-enabled wireless modules have a line-of-sight (LOS) range of up to three miles. Should a node fail, the remaining nodes will automatically route messages around the failed node.

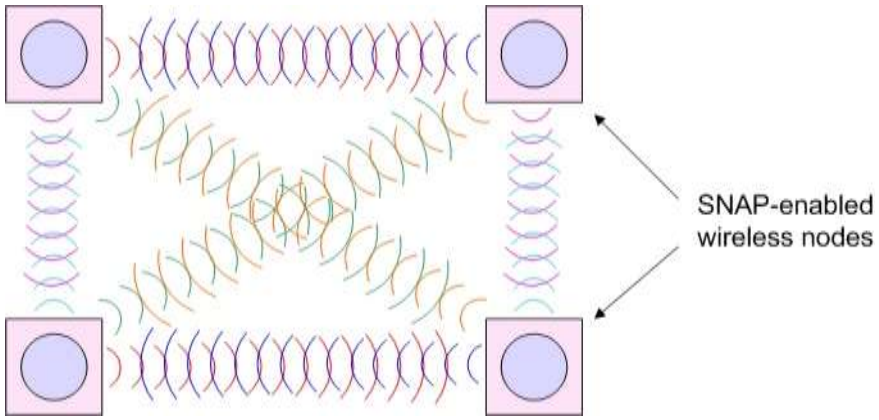

Figure 4. A simple SNAP-enabled wireless network.

As was noted earlier, SNAP-based networks do not require a central controller (such a controller can form a single point-of-failure in conventional network solutions). All SNAP-enabled devices perform full peer-to-peer mesh routing.

In order to keep network traffic to a minimum (thereby conserving bandwidth and power), SNAP provides complete support for remote procedure calls (RPCs). The term RPC refers to a messaging architecture where the name of a function and its associated parameters can be bundled up and broadcast over the network (this is much more efficient than the large, complex packets employed by conventional networks).

Functions in SNAPpy applications can be invoked on any node by any node or by any other device accessing the network. (As will be discussed later in this paper, RPCs can also be passed through the Internet, thereby allowing remote monitoring and control of the wireless network.)

Debugging applications is greatly facilitated by the fact that developers can quickly edit their SNAPpy application scripts, download them "over-the-air" into the wireless nodes, use RPCs to invoke individual functions in those nodes, and monitor any RPCs being made by any network node.

Introducing SNAP Cloud

Now, let's suppose that we wish to have two or more SNAP-based wireless networks communicate with each other via the Internet. The first step, or course, is to actually provide each of our networks with some form of access to the Internet; and one way to achieve this is by means of SNAP Connect E10 units. Each of these pocket-sized devices is a powerful, industrial-class, embedded Linux computer running an instantiation of SNAP and with wireless, Ethernet, and USB communications interfaces as illustrated in Figure 5 (in this example we'll assume Ethernet connections into the Internet).

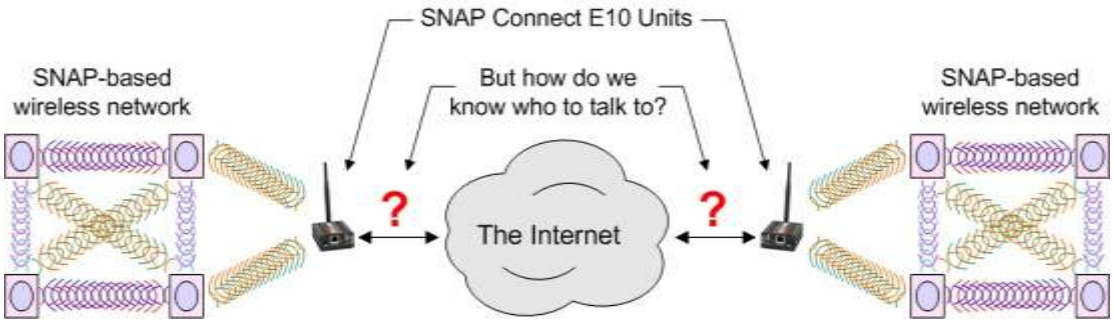

Figure 5. Connecting SNAP-based wireless networks into the Internet.

But there's a problem. How do we know who to talk to (and who NOT to talk to)? On the one hand we want our networks to be able to easily locate and communicate with each other, but we don't want to confuse the issue by having to specify complicated IP addresses and suchlike. On the other hand, we want our networks to be secure, and we certainly don't want our networks to be communicating with other networks or with unauthorized humans.

The point is that this problem is not unique to what we're trying to do. Consider Internet-based services like Skype, Facebook, and Twitter. These allow people to form communities and to communicate with each other without ever having to specify (or even know) an IP address. Let's take Twitter as an example. First of all you

create an account and specify the username and password you wish to use. Now you can Tweet away to your heart's content. Also, if you wish to follow someone else's Tweets, all you need to know is their username – then you simply visit their account and click the "Follow" button.

How is all of this achieved? In fact the Twitter website and software is hosted on what is known as a Cloud Server, which is a kind of distributed server in which additional virtual servers can be brought online (or taken offline) to accommodate the current demand. Similarly, SNAP Cloud may be visualized as one or more instantiations of SNAP running on a cloud server as illustrated in Figure 6. And what does SNAP do? As we now know, SNAP forms a mesh with any other SNAP instantiations it sees via any of its available communications interfaces.

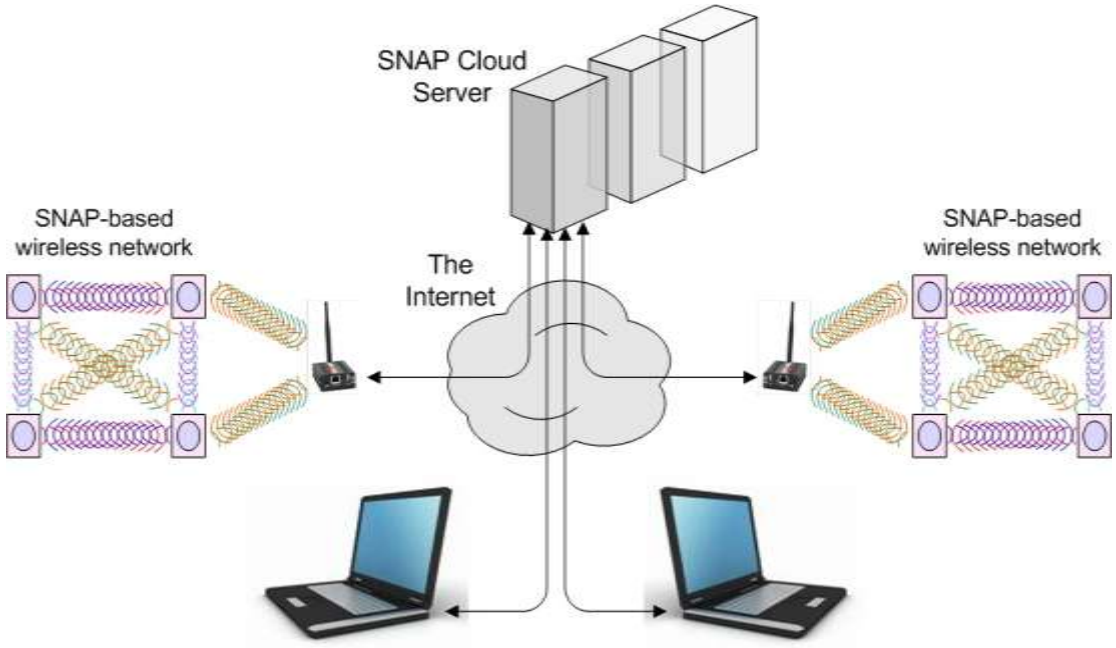

Figure 6. SNAP Cloud is the "magic" that makes everything work.

So, the way this really works is that when we connect a SNAP Connect E10 unit to the Internet, the first thing it does is to logon to the SNAP Cloud server and to provide the username and password that's associated with a particular "community" that the user has set up. (This information can be quickly programmed into the SNAP Connect E10 units by connecting them to a PC via their USB ports.) Any SNAP network who's SNAP Connect E10 has the same username and password can communicate with other networks in the same community. Similarly, any human can access any node on any SNAP-enabled wired or wireless network ... so long as they know the username and password associated with that community. Thus, a system administrator can now monitor and control networks located around the globe.

But wait, there's more, because any of the connections into the Internet shown in Figure 6 could have associated firewalls. These firewalls permit the units they are protecting to send messages out into the Internet, but they typically block messages coming from the Internet unless those messages are from known, trusted sources. Once again, SNAP Cloud provides the "magic" that makes everything work, thereby allowing any SNAP-

enabled node to communicate to any other node or nodes located anywhere on the Internet – irrespective of any firewalls – with minimal latency.

The end result is to allow even the tiniest embedded system to communicate with its peers anywhere in the world. In turn, this allows all of the machines to make more intelligent decisions that are based not only on what they know from reading their own sensors, but on the collective knowledge of the entire community.