# Scalability in SNAP Mesh Networks

by David Ewing, Chief Technology Officer, Synapse Wireless

It is projected that 100 billion new “uniquely identifiable objects” will be connected to the Internet by 2020 (source IEEE Computing). That’s only 8 years from the time of this writing for what is being called the Internet of Things to massively overtake the Internet as we know it today. Whether or not the predictions hold true, it is certain that the technology of interconnected low-power wireless embedded devices has come of age and is beginning to make its way into products which touch our daily lives. Just as microcontrollers are all around us, largely unknown and unnoticed by most of humanity, networked embedded devices will provide a new layer of information and interactivity with the physical world while remaining mostly “invisible”. This means that embedded networks will have to function without the administrative requirements we associate with “networking” today, even while growing to a scale much greater than our current Internet.

Over the past 5 years a number of technologies have emerged to address embedded networking. At the physical layer there are license-free “ISM” band radio technologies e.g. those in the 2.4GHz spectrum used by WiFi and Bluetooth. The IEEE 802.15.4 specification for low-power embedded communication also shares this band. A number of higher-layer protocols have been used to build networks atop these radio technologies, but they have consistently struggled with performance in systems of non-trivial size (>50 nodes). There are several underlying reasons for this:

- Most protocols have been developed in Academic settings, where it is not economically or logistically feasible to deploy hundreds or thousands of devices. Protocols were often developed and optimized around networks with fewer than 10 devices.
- Mainstream embedded development tools require physical connection to devices in order to develop and debug software, re-program FLASH memory, etc. For a distributed computing system of non-trivial size, this becomes a barrier to progress in protocol development.
- The traditional view of embedded devices is that of a “fixed function” product. Extending that idea to networking has led to a static “profile-driven” approach which doesn’t easily adapt to the demands of large-scale applications.

The nature of embedded networks is that **successful implementations create increased demand** leading to an ever-expanding scale of the network itself. For example, a retail “digital signage” application might initially provide real-time pricing and advertising displays to 50-100 locations within a store. Once that mesh network is functional, the infrastructure is in place for a myriad of additional embedded applications. For example asset- tracking tags for retail merchandise can be introduced, adding thousands of devices to the network in each store. Once a ubiquitous network is available, the ROI for security, marketing analytics, and other functions makes the expansion in scale inevitable.

At Synapse Wireless, we have spent a number of years developing and refining the SNAP Network Operating System in real-world commercial applications. SNAP provides the core building blocks which we have found to be essential to successfully constructing large-scale embedded networks:

- **Routing** – the ability for any device to address any other device on the network
- **Scripting** – every device has a virtual machine (VM) supporting remotely programmed scripts
- **RPC** – devices interact using asynchronous Remote Procedure Call messages

This white paper addresses some common questions regarding *scalability* in SNAP Mesh Networks.

## Network Size

*What is the maximum number of nodes a SNAP network can support?*

The number of devices which can participate in a single network is primarily limited by available **address space**, physical-layer **bandwidth**, and network **segmentation**.

### Address space

Each SNAP node is addressed by its IEEE MAC address, which is a 64-bit globally unique number. With 64-bits, we have 1.8x$10^{19}$ addresses as a fundamental limitation. According to University of Hawaii estimates, all the world's beaches combined account for about 7.5x$10^{18}$ grains of sand – so we have at least enough possible SNAP addresses to remotely monitor and control every grain of sand on the planet! We typically constrain the MAC on low-bandwidth links to a 24-bit unique ID for about a 17 million node limit on a single wireless mesh. Clearly, the challenge of scale in these networks is not due to address-space limitations.

### Bandwidth

SNAP can be carried over any physical layer which allows data to be transmitted from one device to another. The protocol itself makes very few assumptions about the underlying transport layer. The term bandwidth here refers to the capacity of the physical layer to transport data packets. A given RF technology will support N packets per second. For example, consider the Synapse RF200 2.4GHz module which has a data-rate of 2Mbps.

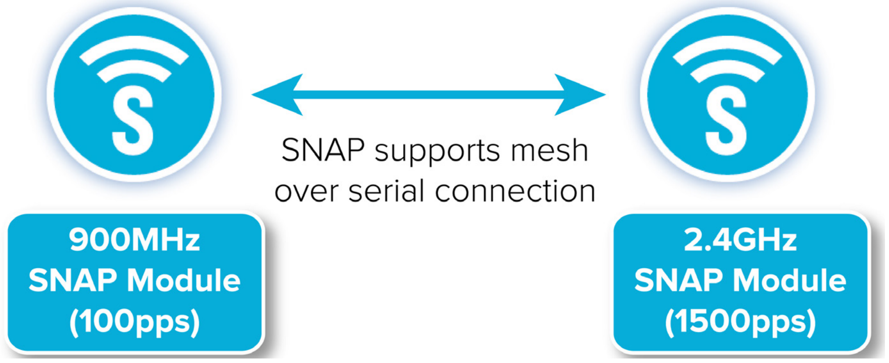
$$\frac{2{,}000{,}000 bits/sec}{8\ bits/byte} \times \frac{1pkt}{128bytes} = 1953\ pkt/sec$$

This yields 1953 packets per second (pps). Reducing that figure by about 25% to allow for radio and processing overhead gives us 1500pps as our network bandwidth budget. How does this impact network size? Say we have a network of sensors, each of which needs to be monitored once every 10 seconds. Using efficient SNAP RPC functions, we are able to pack the complete sensor data report into a single packet, so each device needs to send just one packet every 10 seconds. So our network can grow to 1500x10 = 15,000 devices before we have reached our calculated bandwidth limit of 1500pps.

At that point, if we want to grow the network further we have several options:

1. Change to a *higher bandwidth* physical layer
2. Decrease the *reporting rate* of our sensors, or
3. *Segment* the network.

SNAP's portability among physical layers provides a path for moving to higher bandwidth RF platforms. One of the key advantages of the VM approach is that it protects the software investment in a project even while the physical layer requirements may change due to performance, cost, or other market forces. The routing capability inherent in all SNAP devices also allows mixed networks to be trivially constructed. For example, an existing 900MHz network with low data-rate capabilities can interoperate with a new line of higher-performance devices at 2.4GHz simply by connecting two SNAP devices together to form a bridge.


SNAP supports mesh over serial connection
900MHz SNAP Module (100pps)
2.4GHz SNAP Module (1500pps)

Additionally, the scripting capability of SNAP nodes is routinely used to decrease the reporting rate of devices in the network. Treating embedded nodes as general-purpose computing devices, in contrast to the fixed-function profiles they are relegated to in other network architectures, allows these devices to intelligently determine when communication is absolutely required. The most efficient network transaction is the one which takes *zero bandwidth* – because it didn't have to be sent! For instance in the example above where sensors reported every 10 seconds, they could be programmed to report only when the sensor-reading changes by >5%. Pushing this basic application intelligence to the edge of the network can easily gain 10x performance increase (one-tenth  the packets required). So in our example if we have a relatively slow-moving sensor such as a temperature / humidity probe, the number of devices supported will increase from 15,000 to over 150,000 devices on a single network.

## Segmentation

There are a number of reasons for dividing a network up into multiple segments. The "backhaul" of mesh data over a LAN or the Internet is often done for simple connectivity reasons, for example to join multiple buildings  on a campus into a single logical network. Interconnecting multiple network segments extends the range of SNAP mesh well beyond that of a standalone low-power RF network. Likewise, this segmentation also multiplies the scale of the network beyond that of the independent wireless physical links. So in our previous example  with a limit of 15,000 sensor nodes, we could segment the problem into multiple networks – say on different RF channels – and join them together logically (using SNAP Connect, which we will discuss later) into one large-  scale sensor net.

SNAP is designed to function as a "meta network" which can be routed across existing transport layers such as Ethernet LANs or the public Internet. Interconnecting SNAP mesh network “islands” creates a larger connected mesh, and mitigates the bandwidth constraints of low-power RF technology. Thousands of SNAP nodes are connected this way today in our SNAP Lighting system, which uses a high-availability Cloud service as a routing hub for devices all over the world. The number of devices connected to this system is growing daily, and it is one of the premiere examples of the Internet of Things at work. Through segmentation, we can create SNAP networks of unlimited size.

## Network Span

*How many “hops” can a packet travel across a SNAP network?*

In SNAP, communication from one node to another can make use of intermediate nodes to “hop” the packets across the mesh between them. In general this happens “transparently” to the devices themselves, as they only have to specify the destination address and RPC function while the underlying protocol takes care of the rest. This common use-case is called “unicast routing.” Current SNAP implementations on 802.15.4 limit unicast routing to 30 hops, which is a deliberate tradeoff to keep route-formation messages small and efficient.

Naturally the routing protocol can be extended to allow “deeper” meshes, at a cost of using more network bandwidth to form and maintain the routes.

SNAP allows customized routing protocols to be implemented using a built-in multicast RPC capability. This is frequently used for networks exceeding the unicast hop limit, and provides a virtually unlimited number of hops. Multicast packets have an 8-bit Time to Live (TTL) field providing up to 255 hops across the mesh using just the built-in RPC functions. Using SNAPpy scripting, this can be extended to an unlimited network depth. For example, a node script can receive and rebroadcast RPC messages based on application-specific parameters   such as variables which track sequence numbers, time-to-live, etc. For example, using application-layer scripts  an embedded node with 16-bit integer TTL values could easily support a network depth of 65535 hops.

### Latency

We showed earlier that the limiting factor on the scale of a SNAP network is not the logical address-space, but rather the bandwidth requirement per given unit of time. Similarly, the maximum hop-limit is determined not  by an encoded TTL field, but instead by the time-responsiveness or latency requirements of the system. When a SNAP mesh network has established a unicast route, the latency for each hop is typically less than 1ms. So an acknowledged packet traveling all the way across a 30 hop mesh network would incur 60ms of latency due to  the time to get the data to the target node plus the time to get the response back from the target node.

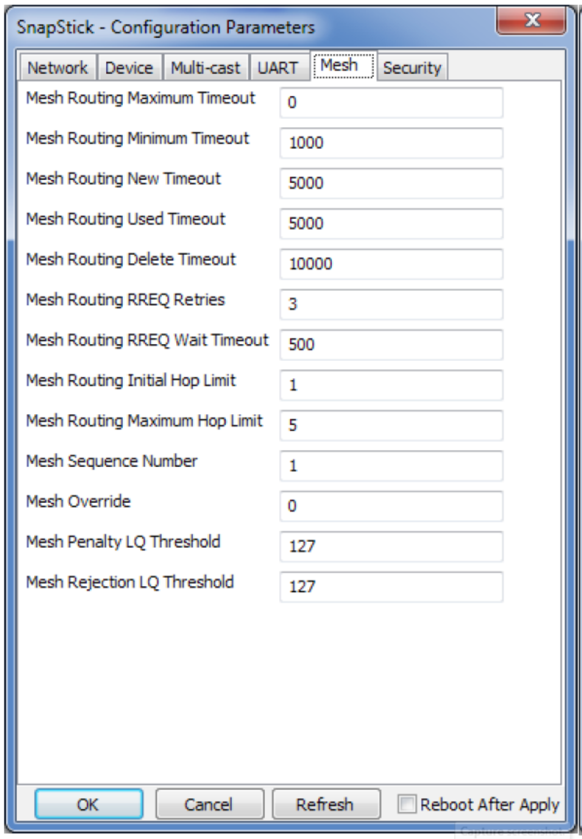
### Routing

As a Network Operating System, SNAP takes an “opportunistic” approach to routing. It does not mandate a specific routing protocol; only that each node is capable of directly invoking RPC functions on any other node using SNAP network addresses. If the transport layer provides a routing capability, as is the case for TCP/IP, we use native routing for packet delivery. The fundamental routing scheme adopted by SNAP is an ad hoc approach based on the DYMO standard (draft-ietf-manet-dymo). Using unique optimizations to this “on-demand routing” approach, SNAP maximizes the effectiveness of limited node RAM to distribute efficient route-table caches across the network. Routes are formed as-needed, and forwarding-tables are updated dynamically with the latest information as connectivity on specific links within the network changes. This routing algorithm is very effective for mobile devices, and environments where RF connectivity is inconsistent. To deal with the dynamic effects of RF fading, which can severely degrade routing performance in typical mesh networks, SNAP supports Link Quality (LQ) based routing parameters.

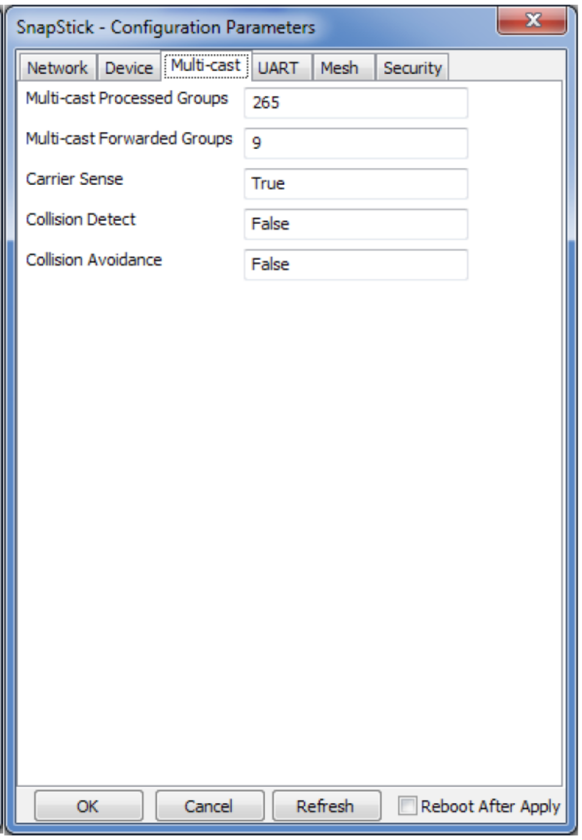
A key function of SNAP's routing algorithm is management of the individual route caches stored in RAM on each mesh node. The goal is to maximize the speed of route formation by minimizing the number of route requests broadcast over the air – all while using the lowest possible number of route-cache entries. SNAP's cache management is quite efficient with typical mesh traffic, supporting hundreds of peer-to-peer routed nodes on platforms with as little as a 10-entry cache size like the Synapse RF100 (Freescale MC13213 w/4k RAM). On platforms with greater resources SNAP supports larger route-caches. For example on the Synapse RF200 (Atmel ATmega128RFA1 w/16k RAM) cache sizes of up to 100 entries can be supported. Here again, the advantage of portability to multiple platforms is demonstrated.

Route formation and maintenance relies on SNAP's network multicast capability. While the hop-latency is inherently very fast (<1ms), additional time delay is deliberately added to each hop in the multicast case. This delay is a "random jitter" factor, which is essential to reduce the effects of packet collisions in dense mesh networks. This means that formation of unicast routes may require 10-20ms per hop. Combined with the goal of constraining route-maintenance messages to single-packet size, this timing constraint influences the limitation of unicast routing to 30 hops or less.

***Synapse Portal screenshots:*** SNAP nodes are set for high-performance routing by default, but to tune large-scale networks for maximum performance, many routing configuration parameters are available...



| | |
|---|---|
| Mesh Routing Maximum Timeout | 0 |
| Mesh Routing Minimum Timeout | 1000 |
| Mesh Routing New Timeout | 5000 |
| Mesh Routing Used Timeout | 5000 |
| Mesh Routing Delete Timeout | 10000 |
| Mesh Routing RREQ Retries | 3 |
| Mesh Routing RREQ Wait Timeout | 500 |
| Mesh Routing Initial Hop Limit | 1 |
| Mesh Routing Maximum Hop Limit | 5 |
| Mesh Sequence Number | 1 |
| Mesh Override | 0 |
| Mesh Penalty LQ Threshold | 127 |
| Mesh Rejection LQ Threshold | 127 |



| | |
|---|---|
| Multi-cast Processed Groups | 265 |
| Multi-cast Forwarded Groups | 9 |
| Carrier Sense | True |
| Collision Detect | False |
| Collision Avoidance | False |

## Interoperability

*How does SNAP connect to the Internet efficiently?*

It is illustrative to first consider how messaging is handled between individuals on the Internet today. If you  want to communicate with me, you'll need my email address. Alternatively you might know my Twitter   "handle" or IM username. Certainly you do not need my IP address to send me a message, even though we are communicating over the Internet! In fact there's no telling which computer or mobile device I might be using, and that device may not have a public IP address anyway. So we commonly use "meta" networks with their own addressing schemes (email addresses, etc) which "ride on top" of the Internet. SNAP devices communicate over the Internet in the same way – using SNAP addresses to find each other regardless of the networking topology underneath.  Dating back to the earliest days of the Internet, RPC protocols have been supported. SNAP over TCP/IP uses standard RPC protocols such as XMLRPC, as well as RESTful interfaces with HTTPS to communicate with Cloud-based services.

Since SNAP has RPC as one of its core building blocks, the RPC layer is the natural transition-point between wireless SNAP mesh and TCP/IP.  Synapse provides hardware and software implementations of a technology called SNAP Connect which performs the bridging function between low-power SNAP networks and TCP/IP. This is done in an extremely lightweight and efficient manner, since with SNAP bridging is reduced to an RPC "transform/routing" operation. By contrast other mesh networking approaches require packet fragmentation and reassembly and/or application-specific coding at the bridging interface.

Efficient transparent bridging to TCP/IP and the uniform implementation of RPC across all nodes allow SNAP devices to appear on the Internet as full-function computing hosts. This allows system software to be developed in a consistent way from network servers all the way down to the embedded devices themselves.  The SNAPpy virtual machine provides an ideal tradeoff of speed vs. space in embedded applications. The cost of high- performance microcontroller cores has dropped dramatically, with advanced 8-bit architectures and sub $1.00, 32-bit ARM cores being on the market for several years now. But FLASH memory is still at a premium for these devices, and the virtual machine – while requiring a few more compute cycles than native code – is extremely efficient in code space.  An embedded application written with the SNAPpy VM will typically consume 1/3 the FLASH space as the same application written in C and compiled to machine binary code. The processing speed  of VM code is more than sufficient to keep pace with radio communication on typical low-power mesh  networks, and when high-performance functions are required, SNAP provides a means for native binary code to be embedded within the script and uploaded to the nodes over the air. The technical capabilities are  impressive, but by far the biggest advantage of the VM is the boost in software development productivity it affords. The ability to use traditional "distributed computing" methods for system software development is a major factor in scaling SNAP networks for real-world applications. Without this capability, the software complexity of even moderate-sized systems overwhelms the development capacity of most commercial organizations.

## Conclusion

SNAP was designed to overcome numerous problems faced by embedded mesh networks. SNAP's "distributed computing" approach has allowed the underlying packet-communication layer to be tailored for a wide variety  of vertical market applications, ranging in scale from simple dedicated pairs to networks of thousands of devices. It is clear to us that – as was seen through the evolution of the "Internet of PCs" – a mature operating system which provides a solid platform for application development and deployment is a fundamental enabler for the large scale "Internet of Things."