Real-Time demands of the IoT

Dr. Johan Kraft - CEO and Founder, Percepio
Originally published in New Electronics 09 August, 2016 www.newelectronics.co.uk

Delivering ultra-low power operation in IoT nodes means they will spend most of their time in a dormant mode, waiting to exchange data with a gateway. Could employing an RTOS help deliver a more reliable user experience?

Connectivity is ubiquitous and the ability for electronic devices to exchange data using various methods and protocols existed long before the Internet. It has been made more relevant in recent years because it underpins the IoT, leading to a significant growth in technologies that provide or support connectivity. Adding basic connectivity between two devices isn't difficult, but extending that to an infrastructure like the Internet does take some careful design.

In this context, 'careful' is synonymous with 'complex', which could make building-out the IoT a challenge that just keeps getting bigger. However, while 'careful' may be an intangible quantity, complexity is something the electronics industry manages very well.
If viewed as a system, the IoT requires a systemic approach to design; technologies that scale, interoperate, can adapt to new scenarios and challenges, while all the time meeting constraints. Embedded systems are often described as constrained devices, which makes embedded design a major driver in the IoT.

Not just for PCs

Outside of the embedded community, operating systems are probably just seen as the software that runs our smart phones, tablet computers and, perhaps, the servers that host our websites. The devices we see as embedded — typically single-purpose, closed 'black boxes' that now proliferate our lives — may not be seen as even needing such an operating system. However the IoT, which will see literally billions of new devices being interconnected over the next several decades, is likely to require the benefits of not just an operating system but one that can offer real-time functionality.

Most operating systems offer a platform to build complex systems, providing key functions that typically includes connectivity. A real-time operating system (RTOS) also offers these ancillary functions but differs from other operating systems in one crucial area, its ability to offer deterministic execution. But why is this important in the IoT?
The IoT will comprise innumerable distributed networks, which will be managed by the infrastructure that is the Internet. Each of those networks will likely be managed by a local gateway, preserving the inherent hierarchy of communication networks. In turn, those

gateways will be responsible for managing communications between various nodes, each of which will very likely be an embedded (and therefore constrained) device.

It is also very likely that many of these devices will be sensor-based and possibly powered by either harvested energy or simple batteries. That makes them slaves to ultra-low power operation. Couple that constraint with the fact that they will very likely be connected using some form of wireless interface and it becomes clear that power conservation will be vital. Using an RTOS to manage these nodes makes sense, particularly if they need to wirelessly communicate with a gateway. All of the middleware needed to enable communications will already be available, in an RTOS that has been optimized for constrained devices. Furthermore, using an RTOS to manage those communications could prove to be the most effective way to deliver power-efficient operation.

It's all in the timing

Many IoT nodes, particularly those that need to work for multiple years from a single battery, will spend much of their time in a dormant state to conserve energy, waking periodically to transmit data to the gateway. If the gateway were only serving one node, this would be a simple process, but if the gateway is managing many tens or even hundreds of nodes, ensuring data is exchanged within specific time slots will require deterministic execution. As Figure 1 shows, each node may believe it is the only node on the network. The protocol used may not offer the ability for long exchanges that would consume valuable energy, but instead limit operation to a burst from the node which must be received by the gateway at a specific time (Figure 2). If the node and/or the gateway isn't prepared for the exchange, packets could be lost. That may not be serious once or twice, but if it happens repeatedly the integrity of the network could be compromised.

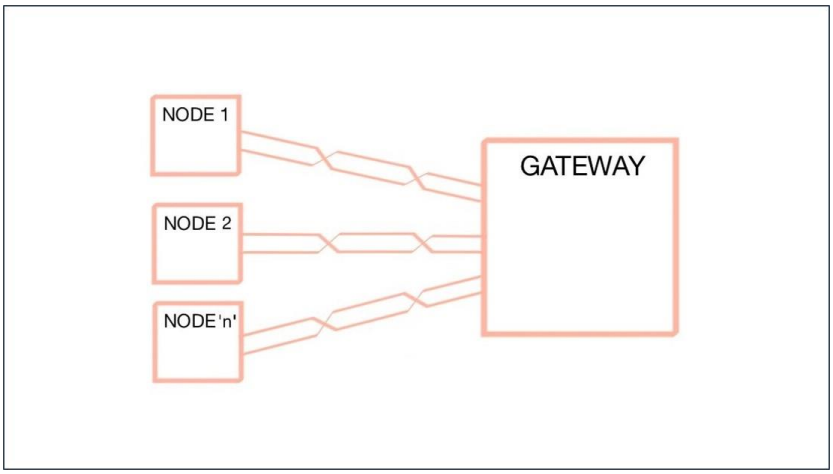

Figure 1: Nodes on the IoT may not be aware of other nodes on the same network, making their timing more critical.

Using an RTOS could guarantee synchronization between nodes and gateways, and there are many RTOS variants available that are now addressing this use-case, such as Micrium's µC/OS, ThreadX from Express Logic, and the very popular FreeRTOS — all of which have been ported to a wide range of the 16- and 32-bit microcontrollers that will provide the intelligence within IoT nodes.

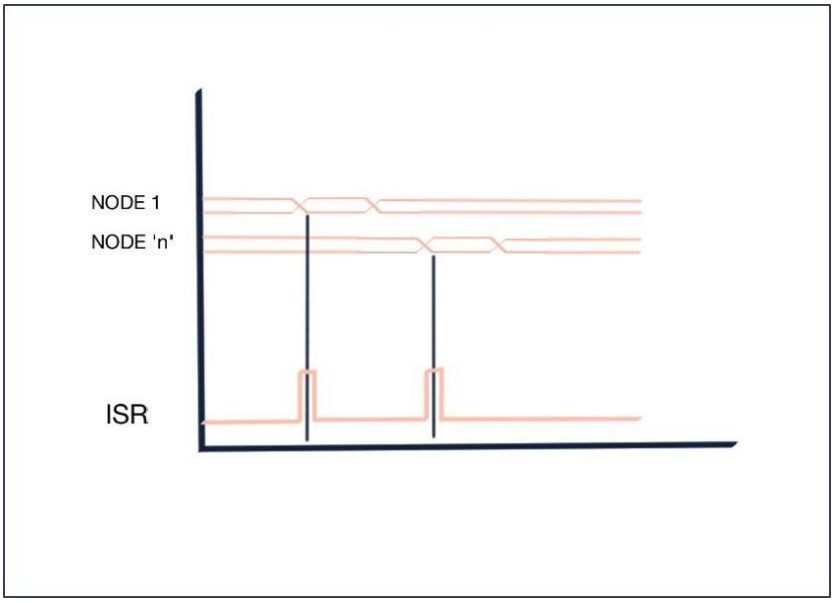

Figure 2: Gateways may need to service tens or even hundreds of nodes in a deterministic way to preserve the integrity of the network's operation

All of these RTOSs are robust and proven solutions, capable of delivering deterministic operation in embedded devices when the application code is written to conform with RTOS behavior (see below). However these and all other RTOSs are still subject to real-world anomalies that can impact performance. These anomalies, as described below, could have unforeseen consequences on devices and so it is important to understand their causes and the possible remedies.

Finding fault

While desktop operating systems are well known for their ability to handle multiple tasks at once, in reality the underlying processor will only be executing one thread at any time. Even multicore processors are limited in this respect, it is the job of the operating system to minimize the time it takes to switch between contexts in order to deliver a multitasking experience. In an RTOS, deterministic operation means task switches must be prioritized, to ensure that the task with the highest priority is always the one being executed. As one task terminates, the task with the next highest priority will begin, or be interrupted by a higher priority task. This prioritization is fundamental to the operation of an RTOS, but it can result in tasks being stopped or interrupted which, ultimately, means achieving real-time execution across all task priorities could be difficult.

While this is part of designing with an RTOS, there is also the presence of jitter. This is the small amount of variability in the time taken to execute tasks; a tolerance, in manufacturing terms. Most of the time, this jitter can be compensated for in the design through prioritization. However, each design is different and runtime events may conspire to render even the most diligently designed code prone to anomalies such as jitter.

Tracking down this kind of activity in embedded code is difficult, one that requires close inspection of the code's execution under real-world conditions. Debugging technologies are able to provide excellent insights into code execution; probes like Segger's J-Link and J-Trace support this type of debugging through breakpoints inserted into the code and the

ability to capture core activity through debug ports. Similarly, Micrium's µC/Probe is a Windows application that allows memory locations in the target microcontroller to be read from or written to during runtime. Both these technologies can be complemented by Percepio's Tracealyzer technology, which displays CPU activity on a per-task or per-interrupt basis, as well as mapping that activity over time. This provides an effective way of identifying timing variations and finding resource conflicts during runtime. Tracealyzer uses software-defined trace, which is very flexible and works on any processor. To record a trace, you only need to include Percepio's recorder library in your RTOS build, configure it and start the tracing. The performance overhead is only a few microseconds per event, and you can stream the trace continuously to the host computer via a debug probe, TCP/IP or other channels. The trace can also be kept in a target-side RAM buffer and uploaded on demand.

Building-out the IoT will require a wide range of technologies, many of which already exist and some that are yet to emerge. Employing an RTOS in IoT nodes could deliver major benefits by ensuring the reliable behavior of network nodes however large or small.

To find out more about Percepio Tracealyzer visit http://Percepio.com or get started immediately by downloading Tracealyzer for your RTOS of choice: http://percepio.com/download/