WIND™

AN INTEL COMPANY

The Internet of Things in the Lab

Staging and Testing for the Real World

By Jakob Engblom

WHEN IT MATTERS, IT RUNS ON WIND RIVER

ABSTRACT

Developing and testing software and configuration variants for industrial Internet of Things (IoT) applications and systems is a challenge. The systems can be physically large, and often contain hundreds or thousands of nodes, which is tough to manage in a physical lab. Testing software that will run across thousands of nodes requires the ability to automate, inspect, and control tests, but automating tests across hordes of physical machines is not easy. These challenges can be overcome by using virtual platforms and simulations of wireless networks and the environment. This converts the difficult hardware into software simulations that can be created, configured, and controlled with ease. In this paper, we cover the techniques for IoT system simulation and testing that we have developed and discovered over the past year, and how to build scalable testing systems.

TABLE OF CONTENTS

INTRODUCTION

Over the past few years, IoT has gone from an interesting idea to mainstream technology that is a must-have for almost every company selling any kind of product. In this paper, we discuss how to better test software for industrial IoT systems and applications. By industrial IoT, we mean applications where a single entity deploys fairly large networks of many IoT nodes for some kind of business or professional purpose. Most consumer applications only involve a small number of devices, and have a somewhat different testing profile.

A typical industrial IoT system follows the hierarchy shown in Figure 1. We have a large number of *nodes* or devices that interact with their local physical world. The nodes are typically low-power, cheap devices that are used in hundreds or thousands of units on a single site or in a single setup. A number of nodes are connected to a gateway.

The *gateway* is a more powerful machine that can run heavy-duty security software and perform extensive computation on its own. On one side, the gateway connects to the low-power network or networks used by the nodes, and on the other side it connects to the server. The gateways are communications providers and intermediaries, helping bring structure to the system and protecting the nodes from Internet-based attacks. Gateways can also be aggregators and perform data fusion and buffering locally, before passing data on up the chain to the server. A single gateway is typically responsible for a set of nodes in some limited geographical area like a single retail shop or single residential building. You might have several gateways for a single site in order to increase reliability and availability.

The *server* or backend, often hosted in the cloud, is where the data from the nodes meets the business logic of the enterprise. Data is collected from nodes via the gateways, and commands and configurations are pushed out to the gateways and nodes. The server interacts with the wider world—for example, for an agricultural system, it might use weather forecasts and market information to determine when, what, and how to plant and harvest crops. Each server is connected to many sites and many gateways, and a very large number of nodes.

Users of the system often interact with the server from their personal phones, tablets, or PCs, using a Web browser or custom applications.

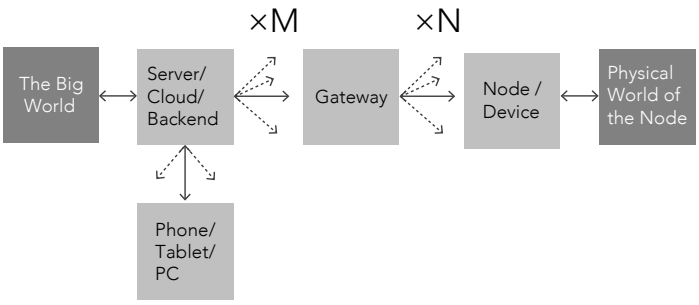

Figure 1. IoT hierarchy

Not every system follows this hierarchy strictly, of course. There are cases where individual leaf nodes talk directly to the servers, or where there are no servers involved and all computation is done in the nodes or gateways, or where users interact directly with gateways or devices. However, such variants do not fundamentally change the testing challenges of many machines deployed in a network interacting with the real world and application code.

Software Testing for IoT

Testing software and configuration variants for IoT systems is a real challenge when using physical hardware. You need many hardware nodes, preferably spread out across a large physical area. Testing wireless networks requires the ability to somehow isolate or control wireless connections between lab machines and between the lab and surrounding world. Inducing sensor data and network failures might require very expensive hardware setups even for very cheap nodes[1]. Changing the network topology or physical configuration of a system requires extensive manual work, and is difficult to automate or do in a repeatable manner. Simulation of the IoT hardware and network allows you to solve many testing problems in a very efficient way.

As shown in Figure 2, our primary concern is testing the software that powers an IoT system. Just like other modern systems, user value and competitive advantage in IoT is mostly derived from the software; basic sensor node hardware is a commodity that is fairly easy to obtain and hard to fundamentally differentiate a product with. In the end, the software and data aggregation are what make IoT systems smart and valuable to users. Software is continuously developed and improved, even after the hardware is developed and the system deployed into the field. Users expect software updates to their systems, to add features, fix bugs, and plug security holes. You cannot ship it and leave it; IoT software development has a lot more in common with traditional desktop or mobile software development, where maintenance and updates and upgrades happen long after the software is first shipped on the hardware.

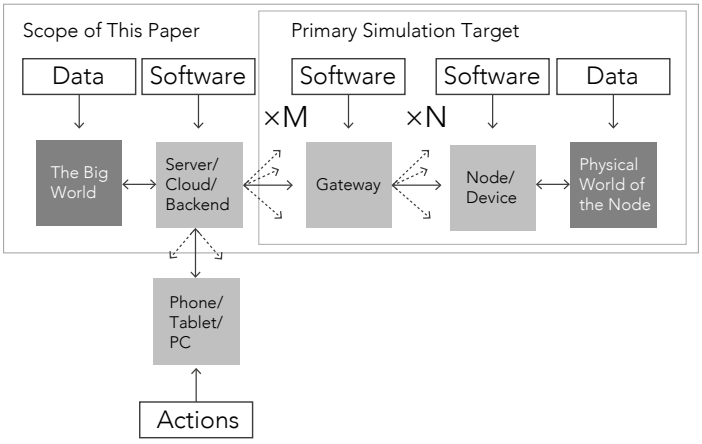

Figure 2. IoT system testing

In this paper, we discuss how fast, virtual platforms[2] can be used to create simulations that will help you test the software for your IoT systems better and more efficiently. It is worth noting that there are many simulation methods for IoT systems that do not run the real code or include the actual node software in their scope. When looking at an IoT simulator, it is important to make sure that the simulation abstraction level and content match what you want to achieve.

Staging for IoT

Before an IoT system is deployed into the real world, the developer should ideally test that their software and system design work in the context in which they will be deployed. This conundrum is a common problem in the world of Web, cloud, and server software development, where new software cannot just be deployed straight from the lab—you need to check that it works in the real context first.

The standard solution for this problem is to have staging setups in place. A staging setup or staging server is a system that is configured to be like the real production system, but is still under the control of development or operations and not being used by real users. It is a test setup modeled after the real world. The staging setup is used to check that software can be installed or upgraded, started, and run, without being tripped up by some quirk of the real system. Staging is about checking that software will work in the context of a particular complete system, not just on a single device or in the development lab.

With IoT, live software updates will become more common for embedded systems, and thus staging becomes a more important part of software development. You want to test a complete integrated software system with a particular topology of nodes, networks, and gateways, to make sure it works not just in the standard tests, but also in the real system. The staging setup would introduce the actual node IDs and addressing from the real world, and might reflect aspects like mixing an old version of the base OS with newer software applications. Correct use of staging is a crucial tool to avoid embarrassing discussions centered around the lines, "It works in our lab" vs. "It does not work in my shop."

SOFTWARE TESTING AND STAGING USING SIMULATION

IoT software testing is fundamentally complicated by the fact that IoT systems are large, distributed, networked systems that have to work in the rough and dirty environment of the real world. Working with a single node is not all that difficult, but working with hundreds or thousands of nodes makes the testing, development, and staging problem exponentially more difficult.

To test using physical hardware, you want to have the wireless nodes spread out over a large area so that not all are in contact with each other. In practice, this requires using entire buildings or campuses as the "lab." Setting up and maintaining such a setup is a significant amount of work, with labor costs quickly dwarfing the cost of the nodes themselves. Real-world labs are also typically limited in the number and variety of nodes and topologies that can be offered[3,4].

Drivers and Board Support

At the lowest level, IoT systems need to test that drivers and hardware work well together. This test is often done using a single node or a few nodes, since it is mostly about local issues. Measuring battery power and driving a complex wireless interface can be worked out on a single node. Virtual platforms can be used for such development, especially if the hardware does not yet exist. However, it is important to also work on the actual hardware, since low-power, cheap hardware tends to have quite a few quirks that only show up in the actual hardware.

Most of the time, IoT systems are based on existing, stable hardware with drivers and board support in place, and the job of the simulator is primarily to run these driver stacks to ensure that the same binaries can run on the simulator as on the real hardware. The main issue facing software developers is how to test the system-level functionality, assuming the basic platform is working.

Single Nodes

Some IoT systems do operate as individual nodes connected directly to the Internet. For such systems, virtual platforms and simulation are useful, just as they are for any other system. With virtual platforms, you can implement automated testing and continuous integration workflows, without depending on hard-to-automate hardware[5]. Testing can also be scaled up in volume by running the virtual platforms on large servers or in the cloud[1]. Network inputs can be controlled, recorded, and replayed to do fault injection as well as regression tests.

Wireless Protocols and Communications

Assuming that basic wireless functionality is there, the next step is to validate the behavior and robustness of the protocols and networking stacks being used.

You can vary the setup endlessly. There are many cases to explore, and many of them do not occur until there is a certain volume of nodes in the system. Wireless networks can have many different topologies, and a test for a certain size of network needs to take that into account. For example, Figure 3 shows two ways that six nodes can be connected to a gateway. In one case, distant nodes have to rely on mesh networking across peer nodes to get messages to the gateway, and in the other case, every node can talk directly to the gateway. These cases clearly subject the networking system to very different types of stress.

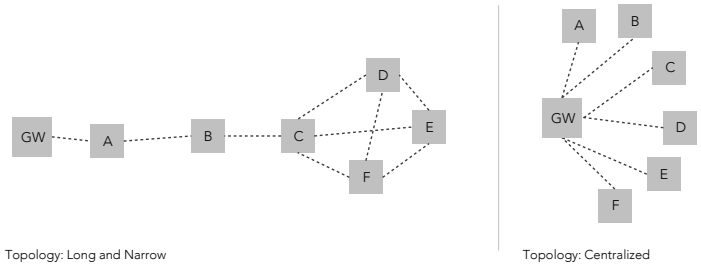

Figure 3. Examples of network topology variations for six nodes

In a simulator, setting up a large network is easy. As illustrated in Figure 4, you just write a program to deploy virtually and spread out the nodes over the virtual space you need, and then model the wireless reachability between the nodes. Instead of manually handling hundreds of physical items, you manage a single script or program.

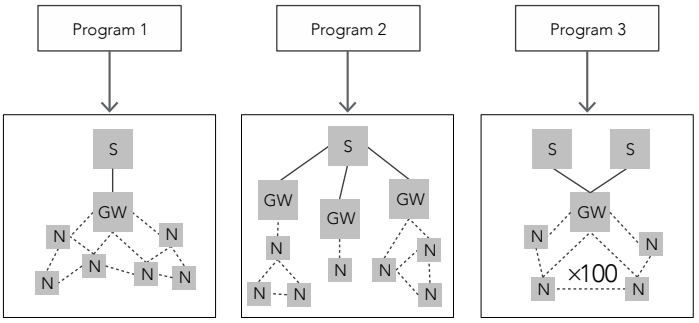

Figure 4. Programmable setups

Wireless networks are by nature unreliable, and faults have to be handled gracefully. What happens in the system when packets get lost or garbled, or a node never sends a reply that it was supposed to send? The connection between a pair of nodes might be interrupted due to changes in the physical world (such as a train passing between two nodes on either side of a railroad track), and what happens to connections and transmissions in such a case? What if a source of radio noise (such as a microwave oven) is close to a particular node, blocking its ability to send by filling the airwaves?

Such cases are easy to introduce in a simulator, and to replay for regression testing and to validate that fixes do indeed solve problems. Using simulation, you can subject software to situations that are very hard to reliably and repeatedly produce in the real world.

A simulator also has the useful property that it avoids using real radio networks, and thus does not interfere with other wireless systems or pick up noise from the outside. In the real world, building test systems for wireless usually involves special isolated rooms or boxes that contain and control radio signals[1].

For staging, simulated networks let you precisely model the network and communications topology of a real system, to test the system software in the context of the real topology. Fault injection can also be used to test the robustness of the proposed system to errors.

More Network Nodes

A key aspect of IoT system development and testing is dealing with the scaling up in size of the system. Experience tells us that fundamentally new phenomena emerge as a system grows. System behavior has to be tested not just with 10 nodes, but also with 100 and 1,000 nodes, and across a variety of ways to connect the nodes. The behavior of the system needs to be tested on a whole range of scales, from small unit tests or subsystem tests all the way up to the largest setups imaginable. Often, each system scale will reveal different issues in the system; this testing is not just about the very largest setups, but also about making sure things work efficiently at intermediate system sizes too.

As already mentioned and shown in Figure 4, setting up a large system is easy in simulation; just run the configuration program that produces the desired scale. It might be programmed to introduce some element of randomness to each test[6], or might be set up to produce a particular fixed setup for reliable regression testing.

Networked IoT systems are distributed systems, and distributed systems introduce timing as an important parameter. Events that happen too quickly or in a particular order can cause a system to crash—for example, many nodes restarting at once could overwhelm the master in a master/slave system. Reproducing such events in the real world is hard, but in a simulator, repeatability is a given[2].

IoT systems are also usually heterogeneous. Gateways and nodes are of different types, and many types of nodes can be expected to coexist in a network. Simulation offers a way to have an infinite supply of all types of hardware, making it possible to simulate arbitrarily mixed networks to ensure that software works on different generations of hardware as well in a system with mixed generations at the same time.

For staging, simulation gives you the ability to program a particular customer setup as a script—including nodes, gateways, how they are connected, and their IDs and addresses. Such a customer configuration can be saved, put under version control, and reused. Rather than physically reconfiguring a lab setup, you just run a program to get the needed setup.

End-to-End Features and Integration

IoT is all about networking and connectivity for systems that used to be isolated, and thus testing how nodes communicate with server applications via gateways and networks is necessary to ensure correct system functionality. Many important workflows require all parts of a system to be in place to be properly tested.

One essential example is data collection and aggregation. Sensor values from nodes have to reach the server and be handled correctly. With simulation, you can test this workflow with everything from a single node to thousands of nodes. Simulation makes it possible to test that the server can handle large volumes of data, and that the system can associate nodes and data even as the number of nodes increases. It provides a way to retest the server side with a known set of nodes and data from the nodes, after the server-side software has been updated. It also makes it possible to test the server management and user presentation of lots of nodes, to make sure that the user interface also scales as the system scales.

Communication in the other direction can also be tested. The server might push down configurations and commands to gateways and nodes, as well as over-the-air software updates. Software update mechanisms can be tested, including injecting network failures or node failures in the middle of the update process to check restart and robustness.

The interface and collaboration between nodes is another test subject. Many IoT networks are based on mesh networks that need to configure themselves spontaneously and discover new network members automatically. Testing that varies the number of nodes and the timing of when they are powered on and try to join the network is very hard to reliably test in hardware, but easy to set up in simulation. Simulation allows you to test things like bisecting a network to see how each separate half develops, and whether the nodes realize that they have been cut off from the rest of the system.

Another aspect of end-to-end testing is doing continuous integration (CI) with the IoT system. In a CI system, new code is first tested in a small context (like a single node or gateway), and once that passes, in successively larger contexts until it is tested in the entire system. To enable CI for IoT, simulation makes it easy to prepare setups that cover various levels of integration, such as a single node, a nodes plus a gateway, a small set of nodes with a gateway, and so on[5].

Integration and end-to-end testing is even more important when IoT systems are built by multiple different teams or companies. In such cases, building a shared simulated test rig and using CI enables gradual integration and reduces development risk.

Security and Authentication

Security is an essential component of any IoT system, and must be part of regular testing. Testing should include mechanisms for onboarding of new devices and retirement of old devices. Managing the membership of trusted devices is one of the key problems in IoT security. System robustness to network-level attacks must be explored, and the vulnerability of nodes and gateways assessed.

Simulation can also be used to quickly get nodes back to a clean state, in order to test onboarding and authentication mechanisms. With real hardware, you need to be very careful how you clean up after a test or initialize a node before a test so that previous tests do not affect the results. In simulation, you can start all tests from a known initial state, and avoid the risk that tests fail or succeed due to an accidentally inherited state.

Using simulation, it is possible to connect nodes to network testing tools and fuzzers, and to record and replay the results of such testing—without worrying about impacting other machines or having the outside world leak into the wireless network. As already mentioned, testing can be parallelized and performed automatically each time new software is released.

The World and Environment

IoT is fundamentally about interfacing computers with the physical world in order to sense it and control it. To test software that interacts with the real world, there has to be a way to provide data to sensors and somewhere to send the output from actuators.

For testing with physical hardware, people often replace the input and output systems with simple software stubs running on the same system, since it is very hard to provide physical inputs to real sensors in a controllable way. There are cases where real inputs and outputs are connected to digital simulations of the external world, but those require large and expensive custom setups.

In simulation, on the other hand, it is quite easy to provide a realistic environment to a simulated control system. From the perspective of the target software running on the simulation, it uses the same hardware interfaces to interact with the external world as it would on physical hardware, such as analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and general-purpose I/O (GPIO). These devices then connect to a simulation of the physical world (or sometimes pass through to the real external physical world of the host machine)[5].

The physics simulation can be a scripted list of values to provide as inputs, or a dynamic simulation of the world. A key benefit of using a simulation for the world is that it is possible to change what the world does and its parameters in order to test code paths for many different real-world situations. For example, a fire alarm could be given a set of stimuli that is supposed to trigger the alarm, as well as similar sets of stimuli that are not really a fire. An air-condition control unit could be forced to work on really hot days, as well as really cold days. There would be no need to wait for a certain type of weather to appear outside the lab, or to ship equipment around to chase interesting stimuli.

Thus, simulated physics provide a crucial source of stimulus for IoT testing, and the physics simulation is an important part of the testing solution, as shown in Figure 2.

At the server end of the system, there is also the big outside world (or markets) and large-scale environment. This world might be represented by the real world, but in order to do scripted closed-loop control experiments, it is often a good idea to simulate it as well. For example, rather than using current market information or weather data, information from a specific relevant point in history can be replayed. Alternatively, a simulation of the external world can be used to dynamically generate stimuli on the fly. The interface with the big world is normally via some form of Web-based service interface or database API, which needs to be simulated as an external network function. The server side normally does not interact directly with the physical world, and thus sensor-level simulation is rather unnecessary.

Tools

In addition to testing the IoT system itself, simulation can be interfaced with existing development tools to facilitate configuration and debugging of the system. For example, USB connections from nodes can be exposed to the host, and connected to the same tools that would connect to physical nodes. Real-world network connections can be used to connect various existing debug and analysis tools to the nodes and gateways. There are many cases where connecting the simulated IoT system to the real world adds value and allows the reuse of existing tools and workflows with the simulated systems.

Demonstrations

Simulation can be used to demonstrate IoT software and systems. We have seen teams use fairly small simulated IoT networks hosted on a cloud server in order to provide a way to demonstrate the server-side software to customers. Rather than carrying a network around or relying on a remote lab network hosted in an office, each demonstrating salesperson can spin up his or her own little IoT system and use it for demonstrations without having to worry about synchronizing access to a shared resource or other people accidentally breaking a demo.

DEBUGGING

It is well known that all testing eventually leads to debugging. Using a simulator to perform tests provides benefits to the debug side as well. First of all, you can use techniques like checkpointing and record–replay to capture and communicate issues. When an unexpected or wrong result is detected in a simulated test run, it is easy to store and reproduce the failing test case in engineering. This capability makes bug reproduction much easier, and makes it more likely for the issue to get fixed[7].

Once an issue has been replicated in development, using a simulator to debug the system provides many benefits. In particular, for a system with many small nodes, the simulation makes it easy to access any part of the state of any node, without having to physically connect a cable to a particular node somewhere out in the field[2]. In simulation, it is possible to stop the entire system instantaneously and inspect the state with everything standing still. The state that is accessible for debugging includes the hardware, software, and physics.

SIMULATING NODES AND GATEWAYS

So far, we have covered what we want to test, and how simulation can be used to facilitate testing. We will now discuss the mechanics of how to realize a useful simulation of a large IoT system.

The starting point for the IoT simulation that we propose is the use of fast, transaction-level virtual platforms[2] for the individual gateways and nodes, and sometimes also the servers. Fast virtual platforms simulate the hardware of an (embedded) target system, and run the same binaries that will run on the real system. In our work, we have used the virtual platform system called Wind River® Simics®[8]. However, most of the techniques we describe here are applicable to any virtual platform tool, and IoT simulation in the style described here has been implemented in the past in tools such as Cooja[6]. Google also uses some hardware simulation in their ChromeCast testing[1], although limited to single individual devices. Many higher-level approaches to IoT simulation have been proposed where you do not actually run the real code from the nodes and gateways, but rather abstract their behavior to host-based programs. However, such simulation does not let you test the real code, and thus does not do what we want to achieve in this paper.

As illustrated in Figure 5, the simulation would consist of a large number of individual systems, the hardware of each of which is simulated using virtual platforms. The virtual platform accurately models the aspects of the real system that are relevant for the target software, such as processor core instruction sets, device

registers, RAM, ROM, flash, memory maps, interrupts, timers, and the functionality of other peripheral devices and I/O devices. The architecture and hardware of the virtual platforms is entirely independent of the host system; for example, running code compiled for low-end microcontroller ARM® Cortex®-M or Intel® Quark™ core on a powerful Intel Xeon®–based server.

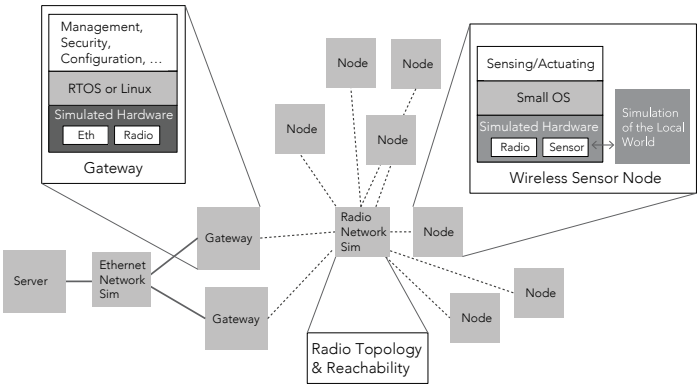

Figure 5. Node and gateway simulation

A fast virtual platform typically does not model the detailed implementation of the hardware and its microarchitecture, such as bus protocols, clocks, pipelines, and caches[2]. By avoiding these details, a simulation can run fast enough to run real workloads, and can typically cover between 80% and 95% of all software tests and possible issues. The underlying assumption, which has been proven true in very many projects, is that it is possible to create simulated devices that make the software run correctly, without modeling the hardware timing cycle by cycle. Virtual platforms suitable for timing-dependent device driver development do exist, and are often built as part of hardware design projects for system-on-chips (SoCs), but such timing-accurate virtual platforms run too slowly to allow the scaling-up simulation that we want for IoT system testing encompassing many nodes. For large-scale software and system testing, fast, transaction-level virtual platforms are the only reasonable technology choice.

The target software running on the virtual platform includes low-level firmware and boot loaders, operating systems, drivers, middleware, and applications. If a system uses hypervisors or containers, they are also part of the software stack on the platform. Drivers for I/O devices are part of the setup, and sensors and actuators are represented by simulations of their software-visible interfaces (memory-mapped registers, interrupts, and direct memory access [DMA]).

As shown in Figure 5, you can run multiple boards inside a single simulation, along with the networks connecting them. It is possible to connect the virtual platforms to the outside world via networks or integrations with other simulators.

Via real-world connections, you can reuse existing software testing infrastructure and systems that work via software on real nodes[4]. What simulation alone brings is the ability to do automated testing across very large numbers of nodes, which would be literally impossible to do using physical systems.

INCLUDING THE WORLD

Simulation of the world is typically done using technology distinct from the virtual platforms used for the computer systems in the target system. It is common to see simulators written in common computer languages such as C, C++, Python, or Java, as well as created using model-driven approaches such as MATLAB®/Simulink®, LabVIEW, and Esterel. Or a simple list of values might be provided to a sensor, with no feedback or handling of actuator output at all. In the most general case, a control system is best simulated by modeling its dynamic input-dependent behavior over time. Different levels of modeling might be used for different test cases, depending on the goal.

Regardless of how the simulation is performed, it needs to be connected to the IoT node simulators. This has to be done on each node (as shown in Figure 5) even though the simulation state might be global (as when modeling a single real system that many nodes attach to). It is also important to make sure that simulated time can be synchronized between the IoT node simulation and the physics simulation; usually, this is done by having the IoT node simulation drive the physics simulation to keep up with the simulated virtual time on the computer side.

SIMULATING NETWORKS

Just like virtual platform simulation of computer hardware, the simulation of wireless networks can be done at many levels of abstraction. For some applications, it is important to simulate the

actual physical behavior of radio, including the fact that multiple nodes transmitting at the same time might interfere and effectively destroy each other's messages. However, correctly modeling and simulating concurrent access is very expensive, since all simulated nodes would then have to synchronize often to check the current state of the shared medium. This requirement slows down simulation quite radically[9]. Another complex aspect of radio is just which nodes can reach which other nodes, and what the resulting signal strength is.

In Simics[8], we have modeled the IEEE 802.15.4 network in a scalable way, using a transaction-based network model that abstracts from the physical layer into a packet-level message system. The simulation moves entire packets as a unit, and is designed to expose the system software to relevant software-visible effects—without burdening the simulation with too much synchronization, which allows both parallel and distributed simulation for scalability[2, 9].

By contrast, the radio model used in Cooja[6] aims to model the radio aspects, including transmission range and interference due to simultaneous transmissions. A similar level of modeling is used in the parallel and distributed DiSenS simulator[10]. In both cases, the more detailed radio model is designed to support low-level protocol testing, but makes the simulation less suitable for truly large-scale network simulation. In our simulation, we assume that the low-level radio system is fairly stable, and focus on the issues for the software as the system scales up and as new software functions are added on top the network.

Pure network simulators like OMNeT++ are even more detailed, modeling the collision-sense multiple access (CSMA) media access, clear channel access (CCA) detection, and similar features. Such network simulators also typically include higher-level protocols, while our model leaves the protocols to the software running on the nodes (thus testing that software as part of the system test). Both ZigBee and 6LoWPAN has been run on top of the IEEE 802.15.4 network in Simics, showing the versatility inherent in modeling traffic transport rather than protocols[9]. In addition, by modeling at the packet-transport level, the traffic can be analyzed using tools such as Wireshark—all those tools see is a packet trace that looks like it came from a physical network.

In our model, illustrated in Figure 6, all nodes are connected to the same wireless network, allowing any node to send a message to any other node. All messages are sent as units (packets) over the network, and there are no attempts to detect overlapping or simultaneous sending of messages from multiple nodes. This is standard practice for transaction-level modeling of wired networks, which we now apply to wireless networks. In addition to the packet payload itself, in a wireless network we have to track the particular radio channel being used, and the signal strength of the message from the sender to the receiver.

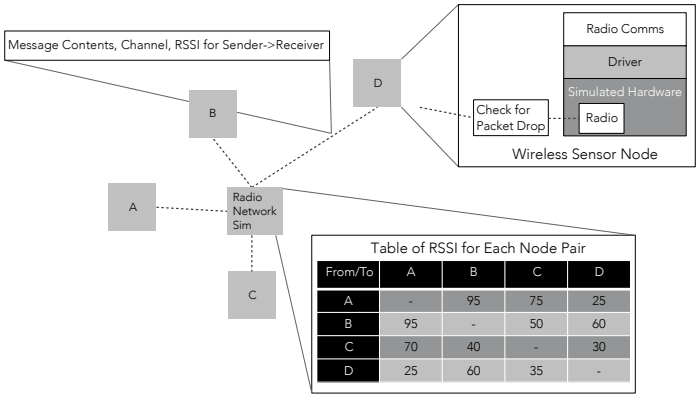

Figure 6. IEEE 802.15.4 network simulation in Simics

The message contents are all the message fields as specified in the IEEE 802.15.4 standard. Management of network addresses is left to the radio models (rejecting messages with the wrong recipient address, for example) or software. The network simulation simply delivers the message to all nodes that can receive it, just like they would in a physical radio network. Unlike wired Ethernet, there is no switch that makes sure that only the right nodes get a particular packet.

The channel information is needed to let only nodes that have their radios set to a certain channel hear the message.

The signal strength is used for three purposes. First of all, it is used to tell the simulated radio receiver in the receiving node what signal strength it should report to the software. Second, it is used to model reachability; by setting it to zero, we can model that a certain node cannot reach another node. Third, it is used to introduce a certain level of uncertainty into the simulation: by looking at the signal strength, a message is randomly dropped. The lower the signal strength, the higher the risk that a message is dropped (in a way that is defined by the user with a few parameters).

Thus, by setting the values in the signal strength matrix, users can model any radio scenario they want. A script or program can be used to scatter nodes in a virtual space, computing reachability based on physical location, or the values can be set according to a fixed setup to model a particular topology. During the run, the signal strength values can be changed to model changes in the wireless environment. It should be noted that in the implementation, the storage of the signal-strength matrix is actually done on each sending node in order to maximize the locality of the simulation.

Finally, in order to model temporary interference, a particular node can simply be blocked from sending or receiving messages. The node just will not see any incoming messages or be able to send anything out until it is unblocked. This simple mechanism allows us to model many types of contention and interference from the real world.

It should be noted that this wireless network model can be applied to other types of wireless networks—it captures the essential behaviors of the transport layer for packet-based wireless networks (assuming that software is responsible for all protocol-level processing).

INCLUDING THE SERVERS

Testing the server side of the IoT system is in some ways different from testing gateways and nodes. The server is a single powerful system, and often fairly generic or running on a standard cloud. Thus, simulating its hardware in the same way that we deal with nodes and gateways is overkill for many types of tests. Instead, as shown in Figure 7, the server can be left outside of the IoT simulation, with some form of real-world connection being used in the simulator.

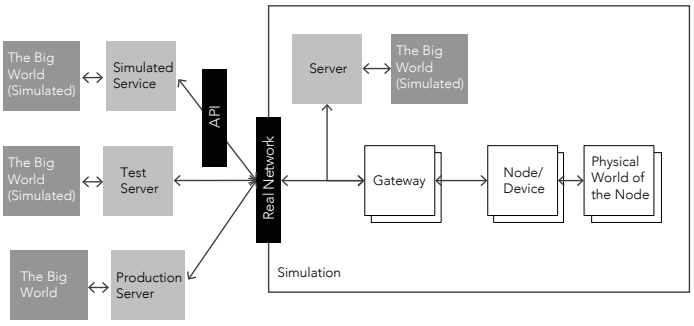

Figure 7. Server testing alternatives

The actual production server could be used for some testing, but only if it is very easy to separate test usage from real usage. The risks involved in accidentally leaking test data and test configurations into production have to be taken very seriously.

In most cases, it is better to use a test server setup that is dedicated to testing. This could be an internal server on the lab network, or a special setup on an external cloud-based server. The test server should have the ability to simulate the world, so that tests can be carried out with simulated physical I/O on the nodes alongside matching big world conditions on the server side.

Another popular technique for testing the connection to the server from the IoT nodes in the field is to simulate the service offered by the server, which is often known as service virtualization. The idea is to avoid having to set up a real server with operating system, databases, and real code, and instead just expose the API that the nodes and gateways expect using a simple simulation system. This approach is typically much more lightweight than running a full test server, and is useful for tests that involve inducing errors or particular replies into the communications with the server. Note that this technique is only useful when you do not need to test the actual implementation of the service; it just uses a simple implementation rather than the real code, and thus does not add much to your understanding of the server code that you write yourself.

SCALING UP

One particular issue that is mostly unique to IoT testing is how to scale up the simulation to hundreds or even thousands of nodes. Most virtual platforms and simulation technology are used with target systems that contain at most a handful of distinct machines, but for IoT, we need orders of magnitude more.

In general, Simics virtual platforms have proven to be fast enough to run even very large workloads including thousands of target processors[11]. For IoT in particular we have some target system properties that work to our advantage in speeding up the simulation. First of all, IoT nodes are generally based on low-power, low-speed processor cores that run at a fraction of the speed of a typical host PC. Second, most IoT nodes have very low duty cycles, often waking up once per second or minute or even hour to do their work—and spending the rest of the time doing nothing but waiting for the next network packet to come in or a timer to trigger

their periodic work. The gateways and servers can be busier, but since there are not that many of those nodes, they do not have that great an effect.

So we see that typically there is a large amount of idle time in the system, idle time that can be exploited to accelerate the simulation by using *hypersimulation*[2]. Rather than playing out idle time cycle by cycle, a simulator like Simics jumps straight to the next interesting event that would wake up a sleeping node. That means that a system of low-duty-cycle nodes can be simulated very efficiently (using very few host resources), which is a property that can be exploited to good effect in large IoT simulations.

Parallelism must also be used to obtain fast simulation. A multicore machine with dozens of cores is easy to acquire today, and when used efficiently, it can increase the speed of simulation commensurably and allow us to simulate very large networks at useful speeds. To achieve parallelism, in addition to having a parallelizable virtual platform simulator, the network simulation must allow for loose coupling of the simulation of individual nodes, and the physics simulation must not require global synchronization (other than very rarely).

When these requirements are met, the performance can be rather astonishing. We have done some experiments to quantify the possible scalability using a test setup that we built. This setup uses some fairly beefy IoT nodes based on virtual 500 MHz ARM Cortex-A9 processors and the VxWorks® 6.9 operating system, with a control algorithm that wakes up every second. Each node has a local physics model attached. With this setup in Simics, we could simulate 250 nodes with virtual time running twice as fast as real time, using just 10 host cores. Running a thousand nodes was linearly slower, ending up with a slowdown of around two. Had the nodes been set to a lower virtual clock speed, the slow-down would have been lower as well.

TUNING AND CALIBRATION

Even when testing is done mostly using simulation, it should be noted that the hardware and its behavior still matters. In the end, the goal is to build software that works on the hardware and in the real world, and thus the simulation has to be calibrated and tuned to properly reflect the real world. This tuning can take the form of setting appropriate range values for signal strength, and tuning the delay for radio message sending to reflect bandwidth. The speed at which virtual platforms process target instructions might need to be tuned to correspond to observed average instruction times on the real hardware. Data values used for sensors and the behavior of the physics simulation will have to be calibrated to correspond to the real-world scenarios. Note that there are cases when the simulation should deviate from the calibrated behavior. For example, if the goal is the examine behavior under faulty, extreme, or rare conditions, it is obviously powerful to make it behave differently from the middle-of-the-road, standard cases.

SUMMARY

In this paper, we have discussed how virtual-platform-based, transaction-level simulation can be used to build test beds for large IoT systems. Our goal is to test how the software on nodes, gateways, and servers behaves in a system context. By using a particular level of abstraction, the system simulation can run fast enough that system-level testing of real code is enabled. A key enabler for this is a high-level model of the 802.15.4 network, and fast virtual platforms that can still run the real target code.

In the end, simulation-powered software testing makes it possible to greatly expand the testing of an IoT system. Simulation can be used both for in-house testing as part of the development and continuous integration process, and as a way to set up staging environments to test how software will behave in a customer setup, before shipping the software and deploying it in the real world.

REFERENCES

1 Brian Gogan, "Chromecast Testing," Google Test Automation Conference (GTAC), Boston, MA: November 10–11, 2015.

2 Daniel Aarno and Jakob Engblom, "Software and System Development using Virtual Platforms—Full-System Simulation with Wind River Simics," Morgan Kaufmann Publishers, 2014.

3 Laura Belli, Simone Cirani, Luca Davoli, Andrea Gorrieri, Mirko Mancin, Marco Picone, and Gianluigi Ferrari, "Design and Deployment of an IoT Application-Oriented Testbed," IEEE Computer, September 2015.

4 Philipp Rosenkranz, Mattias Wählisch, Emmanuel Baccelli, and Ludwig Ortman, "A Distributed Test System Architecture for Open-source IoT Software," IoT-Sys 2015, Florence, Italy: May 18, 2015.

5 Jakob Engblom, "Continuous Integration for Embedded Systems using Simulation," Embedded World 2015 Congress, Nürnberg, Germany: February 24, 2015.

6 Fredrik Österlind, Adam Dunkels, Joakim Eriksson, Niclas Finne, and Thiemo Voigt, "Cross-level sensor network simulation with COOJA," Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006.

7 Jakob Engblom, "Transporting Bugs with Checkpoints," Proceedings of the System, Software, SoC and Silicon Debug Conference (S4D 2010), Southampton, UK: September 15–16, 2010.

8 Stuart Douglas (ed), "Simics Unleashed—Applications of Virtual Platforms," Intel Technology Journal, Volume 17, September 2013.

9 Jakob Engblom, David Kågedal, Andreas Moestedt, and Johan Runeson, "Developing Embedded Networked Products using the Simics Full-System Simulator," Proceedings of the IEEE 16th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), Berlin, Germany: September 11–14, 2005.

10 Ye Wen, Rich Wolski, and Gregory Moore, "DiSenS: Scalable Distributed Sensor Network Simulation," Proceedings of the ACM Sigplan Symposium on Principles and Practice of Parallel Programming (PPoPP), San Jose, CA: March 14–17, 2007.

11 Grigory Rechistov. "Simics on the shared computing clusters: the practical experience of integration and scalability," Intel Technology Journal, Volume 17, Issue 2, 2013.

WIND
AN INTEL COMPANY

Wind River is a global leader in delivering software for the Internet of Things. The company's technology is found in more than 2 billion devices, backed by world-class professional services and customer support. Wind River delivers the software and expertise that enable the innovation and deployment of safe, secure, and reliable intelligent systems.