Testing Intelligent Device Communications in a Distributed System

David Goughnour (Triangle MicroWorks), Joe Stevens (Triangle MicroWorks)
dgoughnour@trianglemicroworks.com
United States

Smart Grid systems have grown in complexity, utilizing many different communication protocols, network topologies, vendors, and an ever increasing number of intelligent devices. Testing these highly distributed systems throughout their life cycle is becoming increasingly important. For instance, during system design, it is desirable to pro-actively test communication interfaces to expose network bottlenecks, configuration issues, and interoperability problems before implementation. Likewise, testing potential upgrades (both hardware and software) before deployment is highly desirable. However, testing these complicated systems is not usually possible before they are deployed to the field. And once deployed, it can be very expensive and inconvenient to properly test upgrades and prepare for unexpected events. Tools exist to test single, or even multiple devices, but a system is required that can enable efficient testing of distributed systems.

This presentation introduces a software architecture for testing distributed systems that supports multiple protocols, vendors, and hundreds of simulated or real devices across a distributed network. This architecture allows multiple transmission, distribution, and substation devices to be simulated using standard communication protocols like IEC 61850, DNP3, IEC 61870-5, IEC 61870-6 (ICCP/Tase.2), and Modbus. In addition, real devices can be integrated into the test system with the use of these standard protocols.

The architecture described here can simulate from one to hundreds of devices on a single computer or spread across a network of computers. All of these devices can be managed and monitored from a single application located anywhere on the network. Tests can be monitored from multiple clients located across the network or possibly the web. Simulated devices can easily be moved from one computer to another to test various network topologies. A variety of techniques can be used to simulate data and functionality depending on the requirements of the test.

Substation/system upgrades are a critical part of any power system. Despite the use of industry standard protocols designed to improve interoperability, the issue of multi-vendor interoperability is still very real. In addition, software and hardware upgrades need to be validated before being deployed to the field. The architecture described here allows any part of the substation to be simulated and to interact with real components facilitating this type of testing.

Distributed System Simulation and Testing

Testing a distributed system presents significant challenges. Frequently users will attempt to simulate the entire system in a single process. This approach is simple to manage and coordinate, but introduces interdependencies, resource constraints, and cannot properly test communication channels and their impact on the overall system performance.

Alternatively, users will attempt to simulate a system using multiple, independent tools to simulate different components. This approach allows the simulation to be distributed and removes interdependencies but can be very difficult, if not impossible, to coordinate and manage.

In order to effectively test a distributed system, the architecture must enable components to be simulated on multiple devices and coordinated from a single point. Some of the requirements of an architecture designed to test distributed systems include:

- Managed from a single point (which is not necessarily always the same location).
- The ability to support many different data models and standard communication protocols in order to support mixed device and multi-vendor systems
- The ability to add, subtract, or move simulated devices across the network and have the system adjust to these changes (including the naming and addressing of devices)

- The ability to fully configure simulated devices, including their simulated database or model, to match the target system being simulated
- The ability to model the behavior of the simulated devices using a variety of techniques
- The ability to synchronize and manage tests across a distributed network
- The ability to create displays and layouts that can be used to easily monitor and control the system
- The ability to mix simulated and real components

Distributed Test Architecture

The architecture described in this paper satisfies the requirements for testing distributed systems listed above. The architecture has three main components, the Distributed Test Manager (DTManager), the Distributed Test Host (DTHost), and the Distributed Test Administration Tool (DTAdmin). These components are shown in the figure below and described in more detail in the following sections.

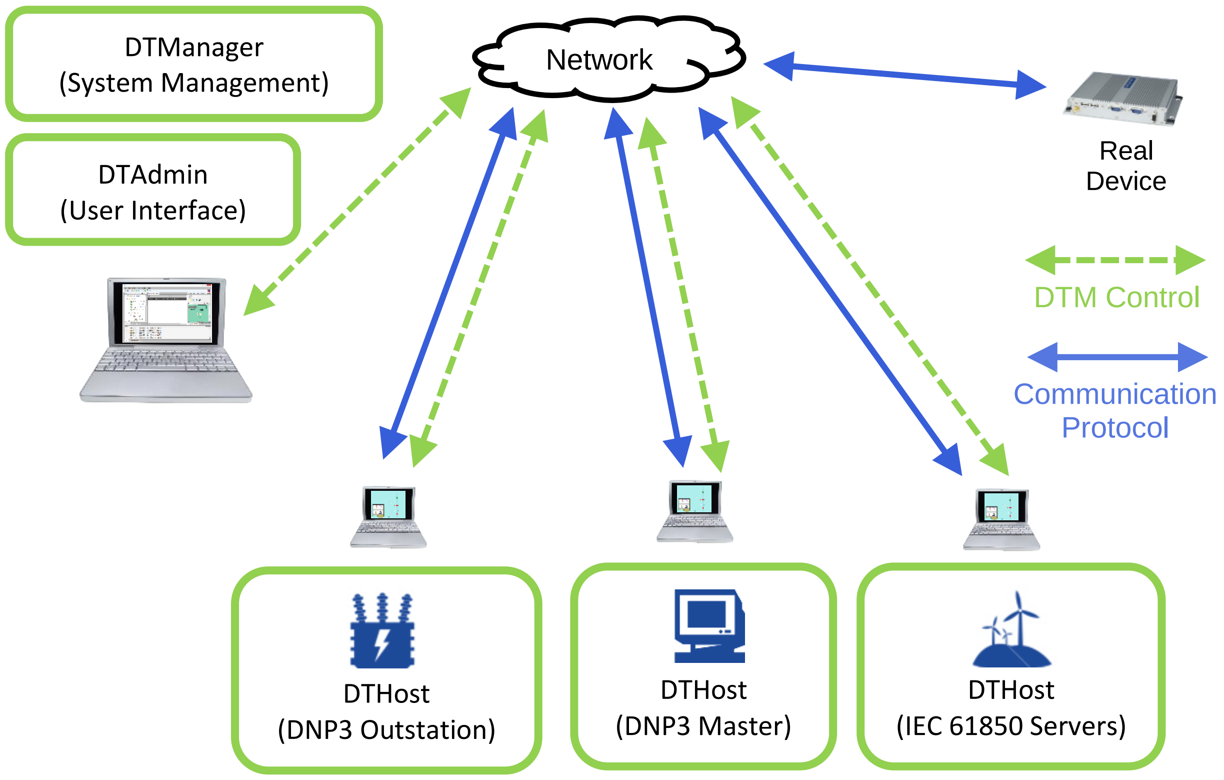

Figure 1: Distributed Test Architecture

Distributed Test Host (DTHost)

In order to support distributing simulated devices across multiple computers on a network, and support running larger numbers of simulated devices, simulated devices will be hosted in a separate process called DTHost. A single DTHost can simulate one or many devices. Multiple DTHost services can be run on the same computer (which typically contains multiple processors) or spread across several computers to match the network topology of the target application.

In addition, the DTHost process can be implemented to be portable and run on different platforms depending on the requirements of the test. If real time behavior is critical, DTHost could be run on a real time OS where critical time measurements are possible.

Distributed Test Manager (DTManager)

In order to coordinate multiple simulated devices across multiple DTHost services, potentially spread across multiple computers, a manager application called DTManager is used. This service coordinates the various DTHost services, manages workspaces, and manages devices, scripts and displays.

Distributed Test Administration Tool (DTAdmin)

It should be possible to administer the entire system from a single point. Attempting to configure and manage multiple components using different tools is error prone and time consuming. The system described here can be managed from the Distributed Test Administration Tool. This tool is an application that allows the user to create/update/delete devices, run scripts, monitor data, and coordinate tests. In addition, the Distributed Test Administration Tool can be started and stopped without affecting the current test and to monitor tests from different points. For example, the user could configure and start a test on Friday and monitor the test from their cell phone over the weekend.

The DTAdmin is a client application that could be implemented on multiple platforms, possibly performing different roles on the different platforms. For example, a traditional Windows application might provide full control and management of the system, while a web or phone based application would only provide monitoring of tests in progress.

Network Considerations

As can be seen in Figure 1, there are two classes of communications in the distributed test architecture. The first is the distributed test control protocol that is used to communicate between the various components of the distributed test architecture. The second is obviously the protocol specific communications of the simulated or real devices. If one of the goals of the test is to get an accurate picture of the network loading and performance, it is important to isolate these two classes.

The distributed test architecture should be designed to leverage multiple network adaptors if available and isolate the protocol traffic to the test network, while restricting distributed test control to an alternate network.

Distributed Architecture Features

In addition to providing an architecture that facilities distributed testing, a variety of features are also critical to many test cases.

Data Access

One of the most critical features for a system test tool is simple access to data from any component in the system. In order to simulate devices and monitor tests, the system will need to have access to the simulated device’s database and be able to change and monitor values from anywhere in the system.

The system described in this document includes a simple addressing scheme that allows access to data anywhere in the system using a path. The path has two components, the first component identifies the object (e.g. a simulated device) that owns the data point. The second part of the path identifies the specific point inside that object.

Data Simulation

In order to support load testing, a simple method for changing large amounts of data should be available. One method for doing this is to iterate through all the points in a device and apply a predefined algorithm (step, random, ramp, sine, etc.) to each point based on their data type.

A second method would be to playback a previously captured, or generated file that manipulates individual data points. This approach is sometimes referred to as key frame animation.

If a test requires more control over data changes than either of the methods described above, data can be controlled using the programming and extensibility options discussed below.

Display

In order to facilitate monitoring and coordinating tests, the ability to present the system using familiar visualizations is highly desirable. A mechanism to generate and support single line diagrams or system diagrams should be included. Ideally the ability to integrate live data into the display to monitor the system should be included.

Programmability and Extensibility

The ability to simulate the behavior of devices and control the system is obviously critical to testing the configuration and functionality. The distributed test system should include a scripting capability that allows the user to control the system and simulate device or system behavior. The script engine provides access to any of the data and functionality in the system.

If scripting does not provide enough flexibility or performance, the end user should be able to incorporate their own functionality by linking to external shared libraries. Using this technique, a user could implement the behavior of their device using a modern programming language (and/or existing libraries or simulators) and then link this functionality into the distributed test environment.

Integration with External Tools

Frequently, users will already be familiar with different test tools and applications and prefer to continue to use those to control their overall test environment but need to add the ability to emulate the protocol traffic of the actual system. The distributed test architecture's client API can be used to allow external tools to configure and manage the system. Many tools on the market (i.e. MATLAB, LabView) provide the ability to load shared libraries and call methods. Using this technique, the overall test could be managed by a different tool that then uses this architecture to simulate the SCADA communications.

Benefits

The Distributed Test Architecture has many benefits for system testing. First, the architecture allows testing coverage to be extended to the actual network and communication protocols used in the real system. This allows testing to uncover many network and communication issues earlier, in lab testing, that otherwise would be found during field testing.

The second major benefit of the architecture is that it allows the number of devices in the system test to be scaled up so that the capacity of the network, device, or application can be tested. This scaling capability enables network load testing and other stress testing for hardware or software.

The third major benefit is high flexibility and extensibility which allows integration with external design tools, simulation tools, or data models in order to extend the testing capabilities of the simulated system.

The architecture also has the flexibility to include real devices in the simulated system to facilitate "hardware in the loop testing". This form of testing is where a simulated device is replaced with a real device as a way to create more realistic test cases or to place the real device into a simulated test environment.

In summary, the main benefit of the architecture is the ability to test complex, distributed systems in early stages of the design process, when issues can be addressed with less time and effort. The alternative approach is to test devices in a piecemeal fashion and delay finding system-level interoperability and interdependency issues until the system is deployed when fixes can be far more expensive and time consuming.

Use Cases

System level simulation can be useful in a wide variety of situations. Examples include Load Testing, Substation Testing, Security, Training, and others. The distributed software architecture discussed in this document can be used to support simulation and testing in all of these areas.

Load Testing

Load testing is generally used to test the limits of a device or a system's capabilities. In some cases, the conditions being tested are designed to represent real life scenarios under extreme situations. In other load test scenarios, the conditions are simply to load the system with as much data, events, etc. as it can handle and observe the behavior.

Device Testing

There are many different scenarios for load testing a device. The device might be a single IED or RTU and the desire is to connect as many clients as possible and make sure the device functions correctly. Alternatively, the device might be a gateway or data concentrator, in which case the test environment could simulate multiple IEDs/RTUs communicating with the gateway along with one or more HMIs/Controllers communicating with the gateway. Finally, the device under test might be an entire HMI and the desire is to simulate the substation to validate the HMI functionality under load.

The example shown in Figure 2 shows a typical scenario for load testing a gateway. The gateway in this system is communicating with multiple IEC 61850 servers and mapping the data and controls to/from IEC 60870-5-104. Load testing in this scenario would be used to measure maximum throughput, latency, reliability, and other factors.

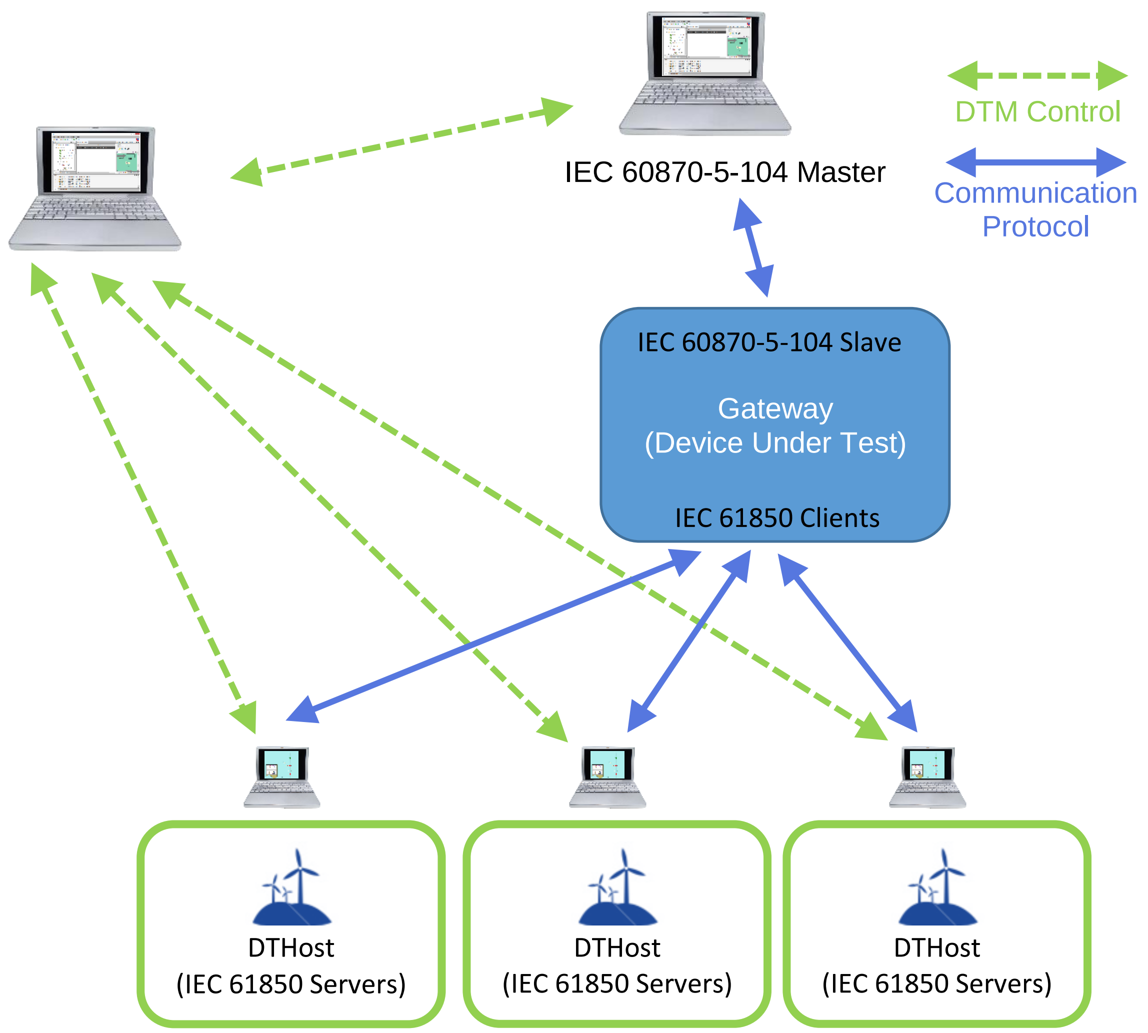

Figure 2: Typical Gateway Load Test Scenario

Using the distributed architecture, all of the IEC 61850 Servers, as well as the IEC 60870-5-104 Master, could be simulated on one or more computers. This would allow the distributed architecture to do closed loop testing, changing data values in one or more of the servers and timing how long it takes for the change to be reflected in the master. Alternatively, issuing a control in the master and measuring how long it takes until the control is performed in the server.

Network Testing

Network load testing is used to validate the physical network including switches, routers, wires, etc. Frequently, in this type of testing all of the devices in the system will be simulated. The objective of this test might be to simply verify that all messages are properly delivered. More often the objective would be to load the network with general traffic and then issue high priority messages and verify the high priority messages are delivered in a timely manner.

Figure 3 shows a simple example of network load testing where three IEC 61850 Servers are setup to send data to an IEC 61850 Client, as well as sending GOOSE messages between themselves, similar to a multi bay protection scenario. The simulated servers can be configured to generate a large amount of general network traffic and measurements can be taken to guarantee that the GOOSE messages arrive within the required limits.

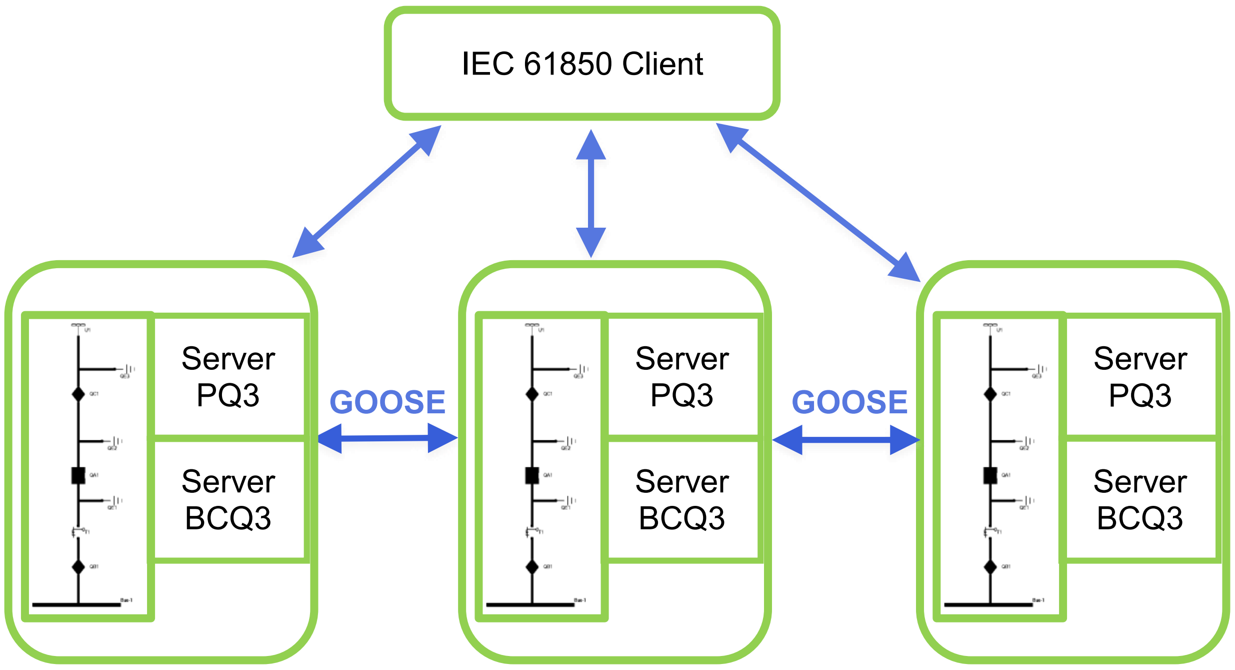

Figure 3: Network Load Test Example

Substation Testing

When implementing or upgrading a substation, it is critical to minimize the amount of time spent debugging the system so that the substation can be brought on-line sooner.

Greenfield Testing

Substation automation has increased the complexity of communications in substations. Part of this complexity arises from IEC 61850 which has introduced many new communication paths with GOOSE, Sampled Values, and MMS based messages that flow between different equipment in the substation. Implementing IEC 61850 on the process bus is moving hard-wired connections to network based messaging which requires a new form of verification versus traditional substations. This new virtual wiring can be verified by simulating the communications of different components in the system together and checking that the right messages are being transmitted and received throughout the system. Testing the communication configuration of the system early in the design process can help reduce risk during the deployment phase.

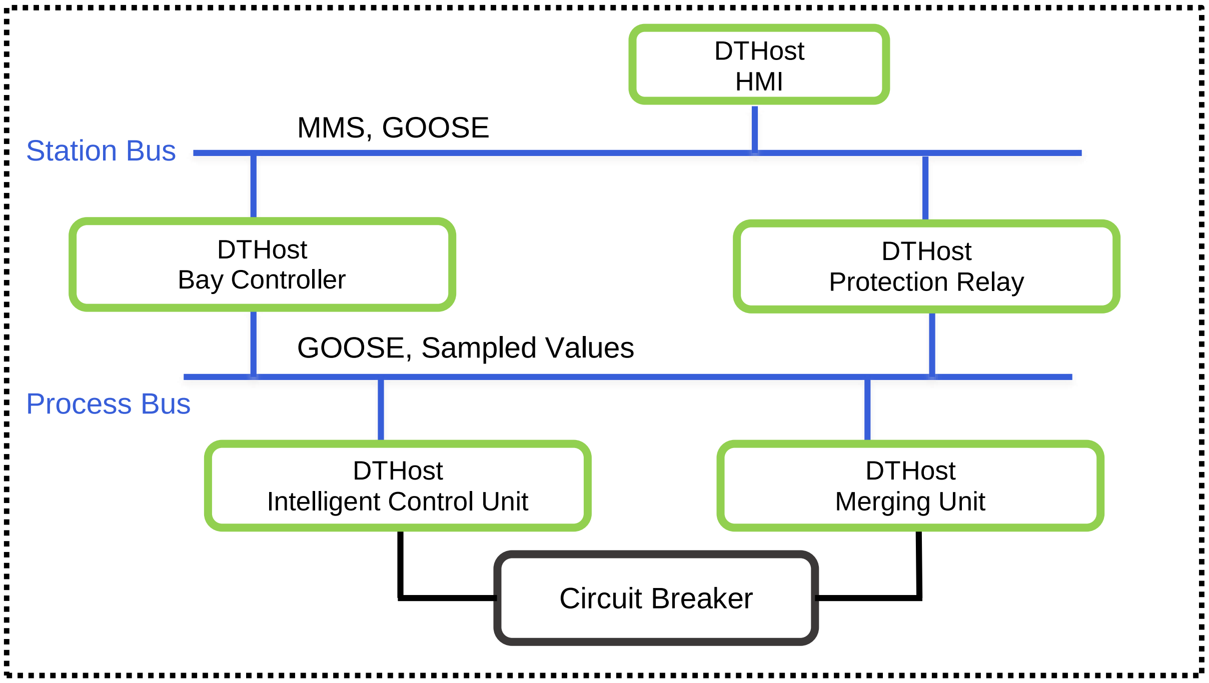

Figure 4: Substation Testing

Figure 4 shows an example of the different types of devices in an IEC 61850 based automated substation. Both the station bus and process bus have various messages transmitting data, commands, and status information between devices. The broadcast and subscription configurations have to be verified to ensure that the right information is available across the substation.

With the distributed architecture, it is possible to simulate the communications and Object Models of the devices shown in green, across a real station bus and process bus, to test the communications of the substation. Each device can be simulated on separate DTHosts, each running on different computers on the network. The ability to load device communication configurations from SCL files enables this system level simulation with IEC 61850 systems. Test cases can be built on top of these simulated communications to test specific substation automation processes. For example, GOOSE messages between protection relays could test a scenario where one breaker fails to open and other breakers should be opened or closed in response to the failure. Debugging these types of configurations in the design phase can save time and effort during implementation phases.

Brownfield Testing

Upgrading an existing substation can range from adding or replacing devices to performing simple firmware upgrades. These system upgrades must be performed with as few unexpected delays in order to limit down time for the system. Worse yet, issues that arise after an upgrade is completed, when the substation is back online, can be even more important to avoid. As with new designs, anything that can be done to minimize the risk is critical.

Using the distributed architecture, new or upgraded equipment can be tested with a simulated version of the existing system. Configuration updates can also be tested. This allows engineers to test the upgraded system before implementing the design changes. Issues discovered with these simulated tests avoid costly down time in the field.

Security

Security is critical in modern SCADA systems. A variety of techniques and tools are available to the SCADA system integrator. Frequently, the first time all of these are deployed together is when the system is constructed. This leaves very little time to address issues, and even less to analyze the system to verify that the security requirements are satisfied.

The scenario shown in Figure 5 is a DNP3 substation that leverages DNP3 Secure Authentication and Transport Layer Security (TLS) for data encryption. In this system, all communications between the DNP3 Master and Outstations run over TLS. In addition, DNP3 Secure Authentication is used to guarantee that only authorized users are able to access critical functions. The optional DNP3 Authority manages users, roles, and credentials for the entire system.

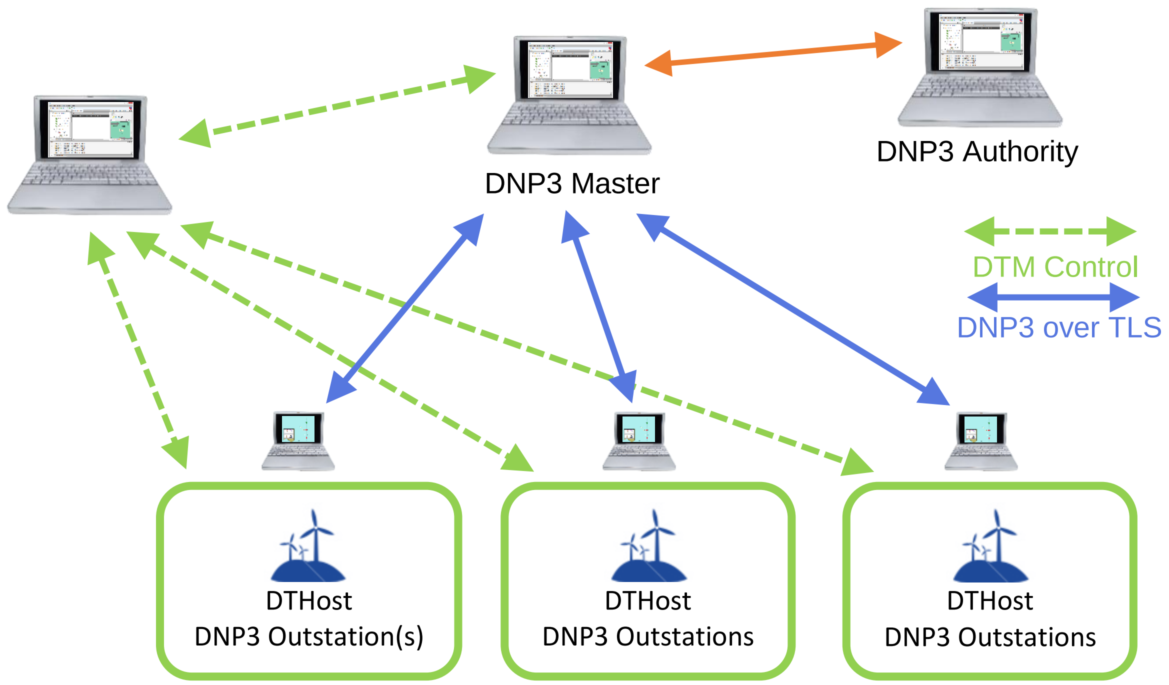

Figure 5: DNP3 Security Example using DNP3 Secure Authentication and TLS

Obviously, defining and configuring a system like this is a complicated process. The architecture described in this document allows all of this to be done in a lab before the system is deployed in the field.

Training

As SCADA systems are becoming more complex, training operators and technicians that use and maintain them is becoming more difficult. Replacing a failed device in the field involves more than simply disconnecting the wires from the old device and connecting them to a new one. The ability to simulate a system and walk through replacing and configuring different components before doing this in the field will assist in this process. In addition, training operators to verify the proper operation of replacement devices will expedite the process as well.

Conclusion

The architecture and most of the functionality described in this presentation have been implemented in the new Distributed Test Manager (DTM) software tool developed by Triangle MicroWorks. DTM is designed to test the use cases described above by simulating the communications and data models of DNP3, IEC 61850, IEC 60870-5, and Modbus devices. Simulated devices can be based on SCL files for IEC 61850 or Device Profiles for DNP3. Test scenarios can be automated with automatic data changes, built-in scripting, flow charting, and custom GUI capability. The distributed architecture allows DTM to coordinate tests across multiple computers so that test scenarios can scale to hundreds of devices and include the real network through a single interface. This approach has tremendous benefits over attempting to coordinate multiple test tools in order to simulate the different communication aspects of an entire system. The architecture also allows the capability and functionality of DTM to extend to other aspects of substation and distributed power systems, including interfacing with external simulation tools for modeling device behavior and monitoring other aspects of the system.