NIST Technical Note 1755

Factory Equipment Network Testing Framework: Concept, Requirements, and Architecture

Jim Gilsinn
Kang Lee
John Michaloski
Fred Proctor
Eugene Song

http://dx.doi.org/10.6028/NIST.TN.1755

National Institute of Standards and Technology
U.S. Department of Commerce

NIST Technical Note 1755

Factory Equipment Network Testing Framework: Concept, Requirements, and Architecture

Jim Gilsinn
Kang Lee
Fred Proctor
John Michaloski
Yuyin Song
Engineering Laboratory (EL)

http://dx.doi.org/10.6028/NIST.TN.1755

September 2012

U.S. Department of Commerce
Rebecca M. Blank, Acting Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Under Secretary of Commerce for Standards and Technology and Director

National Institute of Standards and Technology Technical Note 1755
Natl. Inst. Stand. Technol. Tech. Note 1755, 18 Pages (September 2012)
CODEN: NTNOEF

Table of Contents

1 Introduction 7

2 Purpose & Concept 7

3 Testing Framework Requirements 8

3.1 Overview 8
3.2 Conformance, Interoperability, and Performance 8
3.3 Performance Measurement 9
3.3.1 Overview 9
3.3.2 Latency 9
3.3.3 Cyclic Jitter 10
3.3.4 Bandwidth 10
3.3.5 Physical Value 10
3.3.6 Pass/Fail Criteria 10
3.3.7 Test Conditions 10
3.4 Conformance Measurement 11
3.5 Code Modularity 11
3.5.1 Overview 11
3.5.2 Function Calls 11
3.5.3 Program Execution 11
3.5.4 Network Sockets 12
3.6 Code Extensibility 12
3.6.1 Overview 12
3.6.2 Protocol Modules 12
3.6.3 Testing Modules 12
3.6.4 Analysis Engines 12
3.7 Wireshark Interface 12

4 Testing Framework Architecture 13

4.1 Overview 13
4.2 Universal Client Application 13
4.2.1 General 13
4.2.2 Testing Module 14
4.2.3 Analysis Module 14
4.2.4 Reporting Engine 14
4.3 Personality Modules 14
4.3.1 General Description 14
4.3.2 PM Descriptor 14
4.3.3 Driver Application 15
4.4 UCA Application Programming Interface 15
4.5 Wireshark 15

5 Software Module Interactions 15

5.1 Overview 15
5.2 Scenario: Online Testing with No `StartCommunication` Message 15
5.3 Scenario: Offline Analysis of a Capture File 17

6 References 18

Acronyms

API	Application programming interface
CIP	Common industrial protocol
DUT	Device under test
EL	Engineering laboratory
FENT	Factory equipment network equipment
HMI	Human machine interface
HTML	Hyper-text markup language
ID	Identifier
IENetP	Industrial Ethernet Network Performance
IP	Internet protocol
MAC	Media access control
NIST	U.S. National Institute of Standards & Technology
PCI	Peripheral component interconnect
PM	Personality module
PNG	Portable network graphics
PSML	Packet Summery Markup Language
SDK	Software development kit
TCP	Transmission control protocol
UCA	Universal client application
UDP	User datagram protocol
USB	Universal serial bus
XML	Extensible markup language
XSD	XML schema definition

1 Introduction

This document describes the purpose, concept, requirements, and architecture for the Factory Equipment Network Testing (FENT) Framework and the software to test equipment on real-time factory networks. Other documents contain more detailed information about different aspects of the FENT Framework and software.

2 Purpose & Concept

The purpose of the FENT Framework is to test equipment such as sensors, actuators and controllers in a networked production environment, such as manufacturing or process control facilities, and evaluate their performance and conformance to standards or specifications. For performance testing, FENT provides data collection, analysis, and reporting so that problems that affect the quality and timeliness of information can be identified and corrected. For conformance testing, FENT provides reports that verify how correctly equipment implements the standards or specifications the vendor claims to support.

The FENT Framework is designed to be a modular, open-source testing framework capable of evaluating the performance and conformance of different types of equipment that exist in factory floor networks. The architecture has been broken into three main pieces: the Universal Client Application (UCA), the UCA Application Programming Interface (API), and a series of Personality Modules (PMs). Inside the UCA, there are additional blocks representing multiple Testing Modules, Analysis Engines, and a Reporting Engine. A graphical representation of the concept for the FENT Framework can be seen in Figure 1. A more detailed view of the architecture and discussion of the individual modules can be found in Section 4.

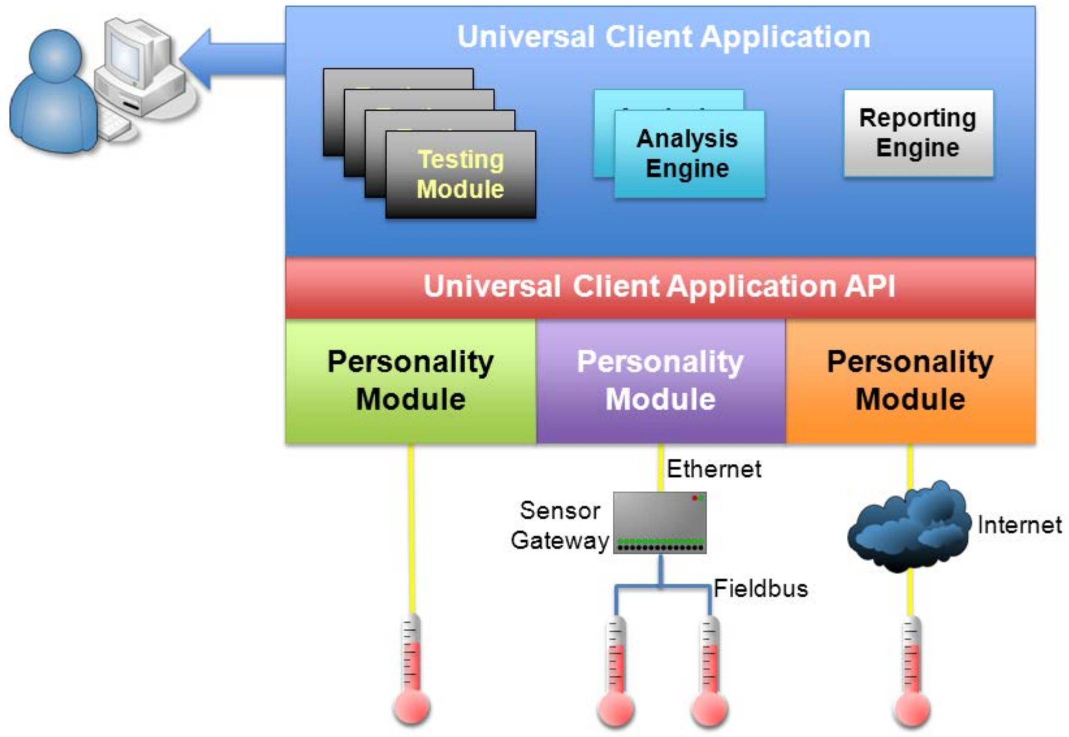

Figure 1 – Concept for the FENT Framework

3 Testing Framework Requirements

3.1 Overview

Previous projects at the National Institute of Standards & Technology (NIST) have shown a need in industry for a consistent tool capable of analyzing the performance and conformance for networked factory equipment. One project developed the Industrial Ethernet Network Performance (IENetP) test tool. [1][2] The IENetP test tool was originally developed by NIST to measure specific performance characteristics for a single industrial Ethernet network. The IENetP test tool proved useful for many organizations and showed a need in industry for this type of performance measurement for other networks. The current IENetP software code base has been shown to lack the extensibility necessary to be used by a larger audience.

The FENT framework will meet the needs of multiple groups that are responsible for measuring the network performance and conformance of factory equipment. By being designed from the outset to be portable, extensible, and open, the software code base should provide greater flexibility in application and use by a larger audience.

3.2 Conformance, Interoperability, and Performance

In order for end-users to know that a particular device will meet their needs for their particular application, the end-users need to have some level of confidence that the device:

1. Conforms to its protocols and standards;
2. Interoperates with other devices; and
3. Performs at a certain level.

Many network protocols and standards have tests and/or tools developed to prove that a device conforms to those protocols or standards. These tests and/or tools generally check to make sure that the device meets all of the required elements in the protocol definition or the standard, but may not test for optional elements. A device may meet the minimum set of requirements for a protocol or standard, but may not meet end-users’ needs if these optional elements are not implemented correctly or at all.

Some network protocols and standards have tests and/or tools developed to demonstrate that a device interoperates with other devices on a network. These interoperability tests go above and beyond the minimum conformance requirements for a standard to include some of the optional elements. The list of optional elements that are required to establish interoperability is generally not the entire set of optional elements, but some subset that different vendors or industry groups have decided upon. This subset makes up a minimum set used by a majority of the devices using that network protocol or standard. If a device demonstrates its ability to meet the interoperability requirements, end-users can be relatively comfortable that the device will operate correctly in their application and that it will work with a variety of other vendors’ products. Even though the device may operate correctly, it may still not be the right device for the end-user’s particular needs or application.

Conformance and interoperability are necessary, but not sufficient, for end-users to know whether a particular device will meet their needs. Performance metrics are also very important for that purpose. They establish that not only does the device operate correctly, but that it also operates at a particular level of performance. They are generally what differentiate one vendor’s product from another and one model from another. Having performance metrics for a device makes it possible for end-users to know to some level of certainty that a particular device will operate at this level under these conditions. It allows end-users to design their system to operate at a certain performance before obtaining equipment, thus reducing the integration time afterwards.

3.3 Performance Measurement

3.3.1 Overview

The performance of a device can be measured in many different ways, depending on what metric is being measured. For networked devices, it can mean that the network interface performs up to a certain level under a specific set of network conditions. For sensors, it may mean that the device measures and reports the correct value to within a certain percentage. For actuators, it may mean that the actuator responds within a given amount of time and to within a certain percentage of the desired position.

In many factory equipment networks, the network infrastructure is no longer the bottleneck for device communication. Generally, the network performance of a device is much more related to the device's internal architecture and software implementation. Different performance characteristics have different importance based on the type of communication used by the devices. The three main communication models that are used for industrial communications are:

1. Polled or Master/Slave;
2. Periodic or Publish/Subscribe; and
3. Triggered.

In polled or master/slave communications, a master device sends a poll or request to a slave device to perform some action and/or return some value. Slave devices cannot perform functions on their own without first receiving a request from the master device. Slave devices can have less computing power than master devices, however they must be architected to perform actions quickly. An example of a slave device may be a small sensor that reports its measurement to a controller when polled.

In periodic or publish/subscribe communications, one device publishes information at a particular frequency and another device subscribes to that information and uses it in some way. An example of this arrangement would be a controller that publishes information that is displayed on a human-machine interface (HMI) every 100 ms. In the case of a controller and HMI, there is most likely a bi-directional publish/subscribe relationship. The controller may publish information for the HMI to display and the HMI may publish information related to the operator's actions to change parameters in the controller.

In triggered communications, one device waits until a certain condition or event occurs before sending a message to another device. This type of communication is used in many systems to reduce the amount of traffic on a network. Devices using triggered communications typically maintain some sort of heartbeat messages to report their status on a regular interval, similar to publish/subscribe but at a much lower frequency.

3.3.2 Latency

Latency is the time delay between two events. For industrial devices, it may mean the time between a command being received and an action being performed and/or a response being sent back. It can also mean the time between a physical value being changed and the value reported by the device reflecting the physical change.

The network infrastructure latency between the sending device and the receiving device can be substantial for large chemical facilities or electrical power companies due to the signal distance travelled. In most discrete-part or batch manufacturing facilities, this network latency is relatively small when compared to the latency introduced by the end devices due to high-speed Ethernet-based networks currently being deployed. A large portion of the latency associated with factory networks is related to the device's internal architecture and protocol stack implementations.

The performance of devices using polled or triggered communications is greatly affected by any latency introduced by end devices. The control loop timing for the system can be affected by the latency of devices taking some action, responding to a request, or reporting some event.

Devices using publish/subscribe communications are less affected by latency. The same amount of latency introduced into every packet sent by a device is not important since that latency would only be seen during connection startup.

3.3.3 Cyclic Jitter

Cyclic jitter is the variability of the timing for periodic events. For devices utilizing publish/subscribe communications, the ability for a device to maintain its communications at the established cyclic frequency is very important. It is very common for industrial end-users to utilize every data point they receive in their control loops, disregarding Nyquist oversampling for digital communications. In that case, control loops may get out of sync with the data received if the packets are delayed enough.

Devices using triggered communications generally have a heartbeat message to report to another device their status and that they are still operating. These messages are typically sent at some low-speed cyclic frequency, and are thus subject to cyclic jitter measurement. Devices using polled communications are not subject to this type of performance measurement since they are not capable of sending messages without first being sent a request.

3.3.4 Bandwidth

Many industrial devices are capable of performing more than one operation at a time. Many networked industrial devices have web servers with diagnostic information or file systems for configuration data. If end-users attempt to access these services during normal operation, a device that is already sending and receiving the maximum number of packets it is capable of supporting, or maximum bandwidth, may start to show performance degradation on its industrial communications. End-users need to know the maximum bandwidth for a device to make sure that they limit their communications to within that bandwidth. Also, devices may need to prioritize their communications to favor the industrial communications over other services.

3.3.5 Physical Value

In addition to the network performance of a device, the ability for a device to relate network information to physical values is quite important. For a sensor, the ability to report its measured value correctly is just as important as the ability to report that value at a particular time. For an actuator, the ability to move to the desired location is just as important as how quickly it took the action. Physical values can be compared to the values included in the network packets in order to evaluate the physical value performance. The physical value would need to be measured using an independent, proven method in order to validate the physical to network value relationship.

3.3.6 Pass/Fail Criteria

The FENT Framework is not meant to establish pass/fail criteria for any particular performance measurement. It is strictly being designed as a measurement tool, similar to a thermometer. The FENT Framework will measure the different performance metrics and report their results to within a certain percentage. It is up to end-users or industry groups to decide what factors determine the pass/fail criteria for any particular device under any particular circumstances.

3.3.7 Test Conditions

The FENT Framework will not only specify the performance metrics to measure, but how to actually measure them and under what conditions. Certain tests may require a particular network architecture to limit the effect of network latency introduced into the measurement. Other tests may require a particular set of background network traffic load in order to measure the cyclic jitter or bandwidth for a device under “normal” industrial network conditions. When specified, these test conditions should be maintained as close as possible and should be documented to ensure that measurements taken at one time can be compared to measurements taken at another time or for another device.

3.4 Conformance Measurement

There are a large number of industrial communication protocols that have each been designed to meet a particular need. Some of the protocols have established conformance test tools and services, while others do not. Even for some protocols that do have conformance tools and services, the end devices may choose to use those services or not. Standards development organizations may not have the ability to force vendors to certify their devices. The FENT framework will help end-users to determine whether devices support the particular communication protocol, although it will not be able to fully replace these testing services.

PMs for each communication protocol will include a driver application that will communicate with the device under test. When developing a new device, or testing an existing device, the ability for the device to communicate properly with the PM can demonstrate some level of conformance. It will not be able to check all of the conformance cases with the device under test, however it will establish basic communication conformance.

3.5 Code Modularity

3.5.1 Overview

The devices tested by the FENT Framework are expected to span a broad range of platforms, from standalone systems with simple serial wired interfaces, through peripheral component interconnect (PCI) bus or universal serial bus (USB) devices with software drivers for Windows or Linux, to complex machine controllers with full Ethernet and web capabilities. It is impractical to develop the FENT software base so that it can be ported to all of the native software environments that host the factory equipment. Rather, the FENT architecture is designed so that custom interfaces to particular equipment can be written to localize the impact on the overall testing framework. The modular software approach shown in Figure 1 accomplishes this. In the figure, the UCA can communicate with the PMs by implementing any of three methods: direct function calls, program execution, or network socket communication across distributed computers. The choice is determined by the support for each type provided by the vendor’s software drivers or hardware protocol, and implemented in the UCA layer following the API. The modular approach followed by FENT to handle each alternative is via definitions of the UCA-API for each of the three communication methods. This concept is detailed in the following sections.

3.5.2 Function Calls

The driver software for some devices may reside on the same computer that runs FENT and include a software development kit (SDK) with a library of function calls to initialize and communicate with the devices. Typically these libraries are provided for one of the popular languages, such as C, C++, Java, or C#. Connecting the C#-based FENT performance testing modules to these SDKs is straightforward. Language bindings for each of these languages are defined in the UCA-API specification [3], and the FENT library contains functions that map the C, C++, or Java implementations of these bindings to the corresponding C# function that is reference by the FENT modules.

3.5.3 Program Execution

Some devices may reside on the same computer that runs FENT and include a set of programs to initialize and communicate with the devices. Connecting the C#-based FENT performance testing modules to these applications is straightforward. Program names that will be called by FENT to initialize and set up communication are defined in the UCA-API specification. Mapping the device-specific applications and their command-line options can be done using simple batch files or shortcuts. In cases requiring more sophisticated interaction, an extra application layer can be built in C, C++, Java, or C# instead.

3.5.4 Network Sockets

Some devices may not reside on the same computer that runs FENT and need to be connected across a network. In this case, FENT will connect to the device controller as a Transmission Control Protocol (TCP) / Internet Protocol (IP) client and send messages and receive replies following an ASCII protocol defined in the UCA-API specification. This method can also be used locally if desired, so that devices with this interface may easily be moved from the local host to remote computers with no impact on the software. This method requires that a TCP/IP socket server program be written that translates the FENT messages to the device's native interface, typically functions in an SDK.

3.6 Code Extensibility

3.6.1 Overview

In addition to the FENT Framework being modular, it is also extensible. The code base will include different modular sections that can be incorporated at compile time and at run time to extend the capabilities beyond that of the initial software. The following sections outline some of the different modules within the FENT Framework and their extensibility.

3.6.2 Protocol Modules

The FENT Framework is specifically designed to work with a variety of different network protocols. By utilizing the UCA-API specification and a simple descriptor file, PMs can be easily designed and implemented for different network protocols. The UCA is also capable of dynamically loading PMs utilizing program execution or network socket communication.

3.6.3 Testing Modules

Testing modules will be designed using a series of common base classes and function calls used by the UCA to conduct the performance and conformance testing. Similar to the UCA-API, this set of common classes and function calls will allow test designers to easily implement their own testing methodology. Due to the tighter integration necessary for the testing modules, these will not be designed to allow program execution or network sockets.

3.6.4 Analysis Engines

The initial analysis engine developed for the FENT Framework uses rather simple mathematical models. As more complex mathematical models are written, they will be implemented as additional analysis engines. This approach will allow end-users to compare new models to the existing models, improving them over time. As with the testing modules, these engines will only be implemented through common classes and function calls due to their tighter integration requirements.

3.7 Wireshark Interface

Wireshark [4] is the de facto standard network protocol analysis tool used in the industry. It is open-source and has a well-established process to develop new or revised dissectors for network protocols. Many of the industrial network protocols already have protocol dissectors written for Wireshark.

Wireshark provides multiple ways to interact with the software. One method is using network sockets to command Wireshark to perform some actions. Another is a command-line version called TShark, or text-based Wireshark, that provides even more features. The FENT Framework may use a combination of both of these methods.

4 Testing Framework Architecture

4.1 Overview

The concept FENT Framework architecture shown in Figure 1 provided a way to develop the requirements shown in Section 3. However, it does not provide enough detail to develop the functional architecture and module interactions. Figure 2 shows a modified version of the concept architecture showing a single protocol PM, the module interactions, and the device under test (DUT).

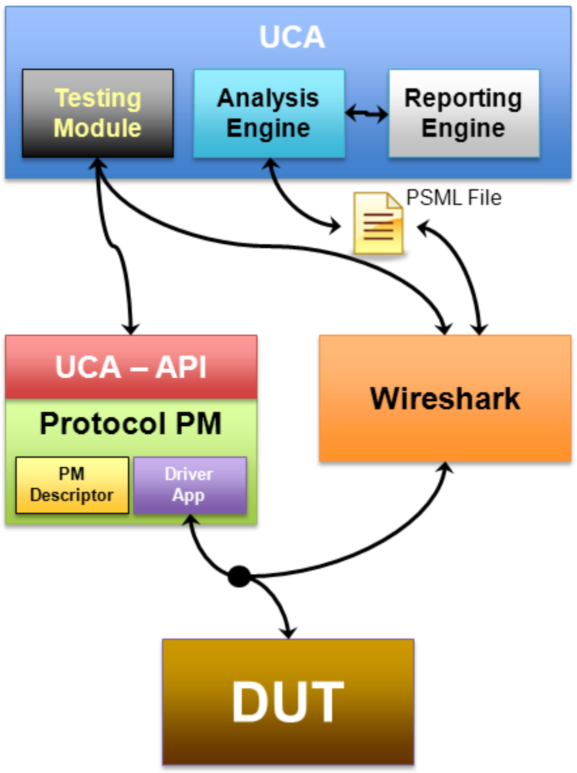

Figure 2 – FENT Framework Architecture with Module Interactions

The following sections provide greater detail about each one of the modules and describe the module interactions. Section 5 offers some example timelines to better demonstrate the sequence of module interactions.

4.2 Universal Client Application

4.2.1 General

The UCA serves as the central testing application for the FENT Framework. It contains the main operator interface as well as the testing modules, analysis engines, and reporting engine. The UCA coordinates the actions of the PMs and Wireshark to conduct the performance and conformance testing. Connections are made between the UCA, the PMs, and Wireshark. Each of the individual module connections will be described in greater detail along with those modules.

The UCA runs on a Microsoft Windows-based computer running Windows XP, Vista, or 7 with the .NET Framework installed. It is built using Microsoft Visual C# and the .NET 4.0 libraries.

The main UCA operator interface is responsible for storing and communicating runtime characteristics to each of the different modules within the FENT Framework, like the DUT's network address and filenames while conducting the tests.

4.2.2 Testing Module

The testing module is responsible for actually conducting the tests. It links with the PM through the UCA-API to perform any communications with the DUT and links with Wireshark to capture and filter the captured network for the appropriate network protocol. The way that the testing module links to the UCA-API and PM is dictated by the way the PM has been designed, as described in the PM Descriptor.

4.2.3 Analysis Module

The analysis module is responsible for analyzing the data collected by Wireshark and calculating the performance results based on some mathematical model. The analysis engine does not read data directly from Wireshark, it uses an intermediate file encoded in Packet Summary Markup Language (PSML) [5]. PSML is an extensible markup language (XML)-based file format that encodes a subset of the available protocol information Wireshark stores for each packet. The specific fields that are encoded in the PSML file are dictated by the Wireshark parameters stored in the PM Descriptor file. The analysis engine is capable of recognizing and sorting multiple communication streams in a single PSML file by using identifying information stored in each packet for each connection. Each of these communication streams is analyzed individually and performance results are calculated for each stream. The analysis engine then sends these performance results to the reporting engine.

4.2.4 Reporting Engine

The reporting engine is responsible for reporting the results of the tests in human-readable form. The results will be displayed in the UCA software as well as output to files. The on-screen display will allow the user to zoom and scroll through the data to observe specific events with the ability to export those views to a file. The automatically generated report files will contain images and reports about each communication stream. Portable network graphics (PNG) image files, hyper-text markup language (HTML), and XML files are the default methods for reporting results. Other file formats will also be available.

4.3 Personality Modules

4.3.1 General Description

PMs are designed for each protocol, and may also be designed for different implementations of each protocol. The PM handles any direct communications with the DUT through the driver application. It also contains a listing of the Wireshark parameters used by the UCA to calculate different aspects of the protocol's network performance.

4.3.2 PM Descriptor

The PM Descriptor is an XML-based file containing identification information about the PM, a list of Wireshark parameters, information about the driver application, and any runtime values that can be specified by the user. The XML schema definition (XSD) file describing the PM Descriptor will be described in a future document.

The Wireshark parameters contain a list of all the relevant Wireshark header fields necessary to analyze the performance of the protocol as well as an additional Wireshark filter that can help to limit the network packets in the capture file to the relevant set. These Wireshark header fields can be used to override some of the default Wireshark fields. Some protocols encode the packet timing value inside the packet, not relying on Wireshark's native packet timing value. This is useful for protocols that can timestamp each packet at the hardware layer. Some protocols do not use IP addresses or media access control (MAC) addresses to identify their devices, so it may be necessary to override the source and destination addresses with a non-default header field as well. Some of the modern industrial protocols have removed parts of the TCP/IP and Ethernet suites from their protocol stacks to improve performance, and some of the legacy industrial protocols were never designed to run over the TCP/IP or Ethernet suites. In these cases, another way to identify the source and destination devices may be encoded in the network packets. There may also be other header fields necessary to measure the performance of the device, including connection identifiers for identifying multiple connection streams, sequence counters to help identify missed packets, and data containing sensor values. All of these different header fields are identified and classified in the Wireshark parameters section of the PM Descriptor.

The PM Descriptor also contains information about how to access and utilize the driver application. This information allows the UCA to send the correct messages to the PM through the UCA-API to start and stop communications with the DUT. This message may include values like the path to the executable for a command line implementation or the correct network socket to use. It may also include any default parameters necessary to operate the driver application or DUT properly.

4.3.3 Driver Application

The driver application is the piece of software responsible for communicating directly with the DUT. The PM needs to be written to abstract the commands from the UCA-API and convert them to the appropriate commands and/or messages in order to communicate with the DUT. No specific requirements are necessary to build the driver application as long as the UCA-API commands are implemented correctly at the PM level.

4.4 UCA Application Programming Interface

The UCA-API is a series of messages that can be sent between the UCA and the PM to perform actions related to the test being conducted and to report results. Generally, these actions tell the PM to establish a connection to and communicate with the DUT. Other actions may be added in the future if a need arises. More information about the specifics of the UCA-API can be found in another document [3].

4.5 Wireshark

The Wireshark module allows the method used to connect to the Wireshark application and the actual commands used to be abstracted from the UCA. The UCA can use a single set of commands which the Wireshark module converts to the correct lower-level commands to interact with the Wireshark application.

In order for the Wireshark module to capture packets for the performance tests, it should also tap off of the communications between the driver application and the DUT. This tap connection should be done as close to the DUT as possible in order to reduce any latency or jitter associated with the network infrastructure.

5 Software Module Interactions

5.1 Overview

The following sections provide simple examples of how the different modules interact for a few of the more common scenarios using the FENT Framework. For each of these scenarios, it is assumed that the correct PM has already been loaded into the FENT Framework and UCA.

5.2 Scenario: Online Testing with No `StartCommunication` Message

One common way that the FENT Framework will be used is to conduct online performance tests on a device that does not require a `StartCommunication` message. A sequence diagram of the messages between and actions performed by different modules and in the FENT Framework is shown in Figure 3.

In the figure, the messages shown between the PM, Driver App, and DUT are not controlled by the FENT Framework. They are shown for informational purposes, but are managed and designed by the PM developer and the protocol specification.

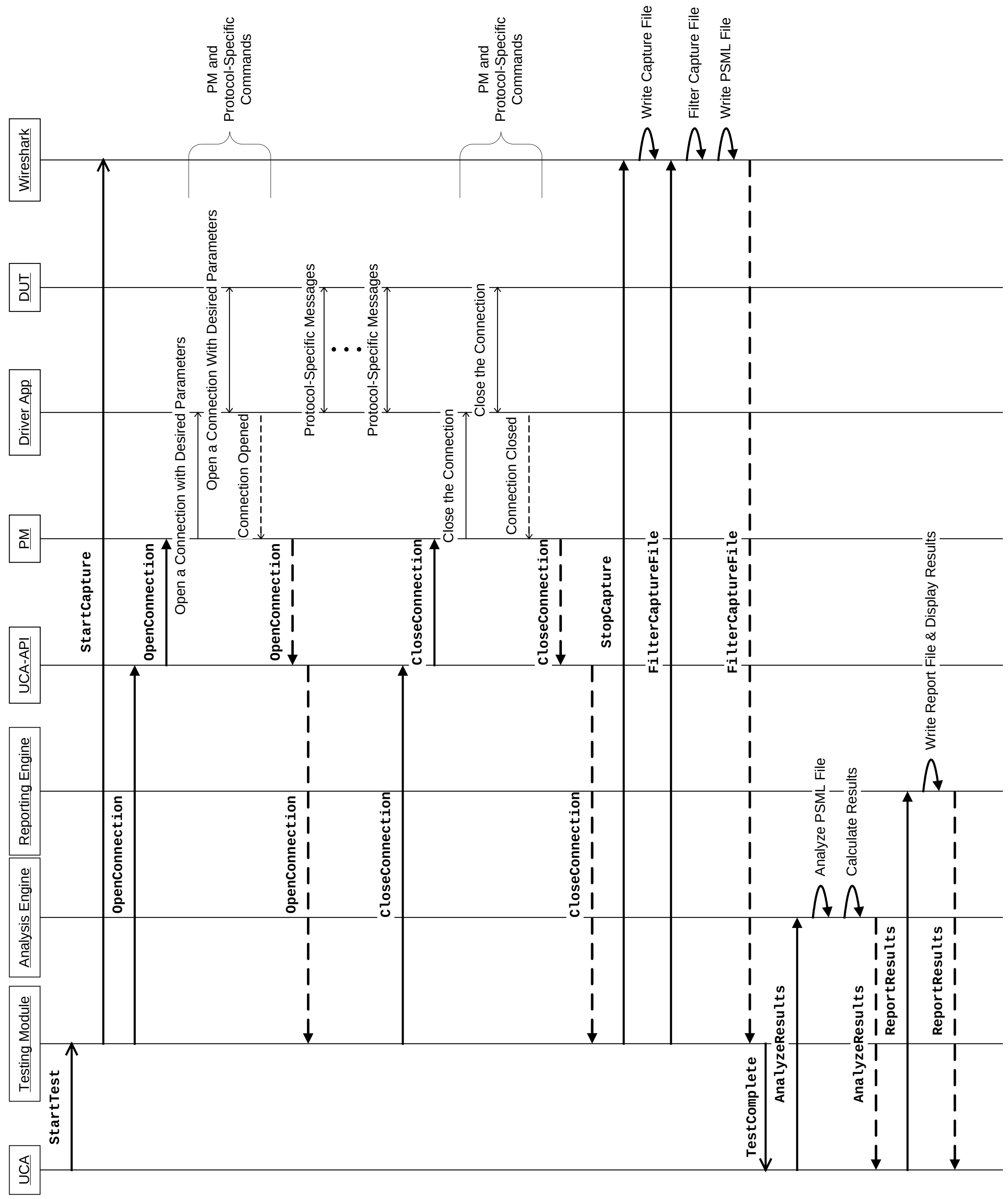

Figure 3 – Sequence Diagram for Online Testing with No `StartCommunication` Message

5.3 Scenario: Offline Analysis of a Capture File

Another common way in which the FENT Framework will be used is to conduct offline analysis of previously captured packets. It may not be possible to conduct online tests with all the equipment necessary at the end-user's facility. Offline analysis allows end-users to capture traffic in any way possible and send it elsewhere to be analyzed at a later time. This capability is also useful for doing long-term performance testing where a series of captures can be compared.

For this scenario, the architecture diagram can be revised and collapsed as shown in Figure 4. The driver application and DUT are removed and Wireshark only reads from a previously captured file.

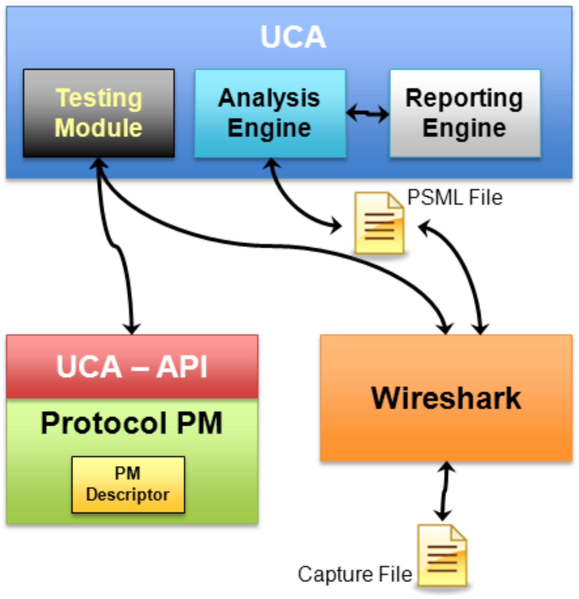

Figure 4 – Revised Architecture Diagram for Offline Analysis of a Capture File

A sequence diagram of the messages between and actions performed by different modules and in the FENT Framework is shown in Figure 5. This diagram is the same as that from the online sequence diagram shown in Figure 3 with the online communications removed.

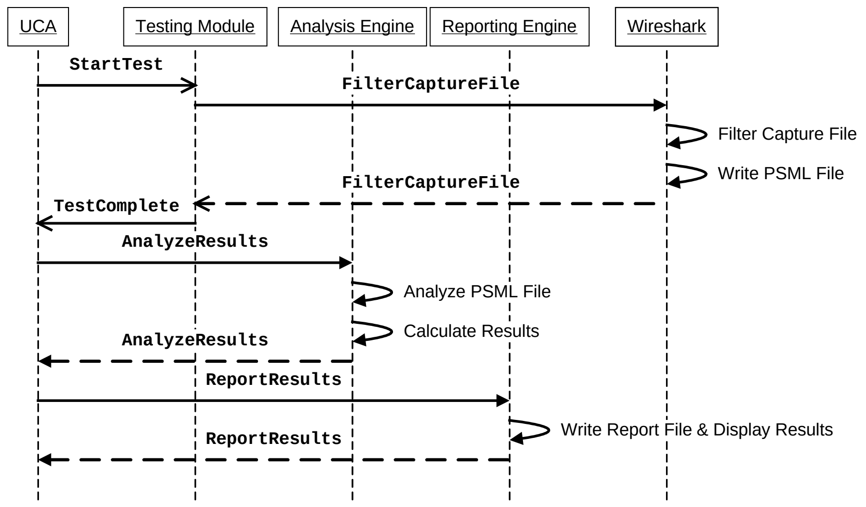

Figure 5 – Sequence Diagram for Offline Analysis of a Capture File

6 References

[1] Gilsinn, J., Johnson, F., “Test Tool for Industrial Ethernet Network Performance”, *55th International Instrumentation Symposium*, June 1-5, 2009, League City, TX.

[2] Gilsinn, J., Johnson, F., “Testing, comparing industrial Ethernets”, *InTech Magazine*, August 2009, pp. 12-15.

[3] *Factory Equipment Network Testing Framework: Universal Client Application*, *Application Programming Interface*, NIST Technical Note 1754, September 2012.

[4] Wireshark Network Protocol Analyzer, available at <http://www.wireshark.org>

[5] Packet Summary Markup Language (PSML) Specification, available at <http://www.nbee.org/doku.php?id=netpdl:psml_specification>