

Side Project Summary 2017–2019

William John Holden

September 11, 2019

Abstract

At my 2017–2019 job as a network management technician I built several side projects. I hope that these programs will be useful to others after I leave. This document catalogs the purpose, capability, and usage of each program. Only projects that were written directly in support of my job are listed here. None of these programs require privileged access or installation. Host-based firewalls may interfere with networking.

1 Connection Map

Location: <https://github.com/wjholden/Connection-Map>

Input: Syslog messages on UDP port 514 from a Cisco ASA.

Output: World map marked with the location of network connections (see figure 1).

Language: Java 8

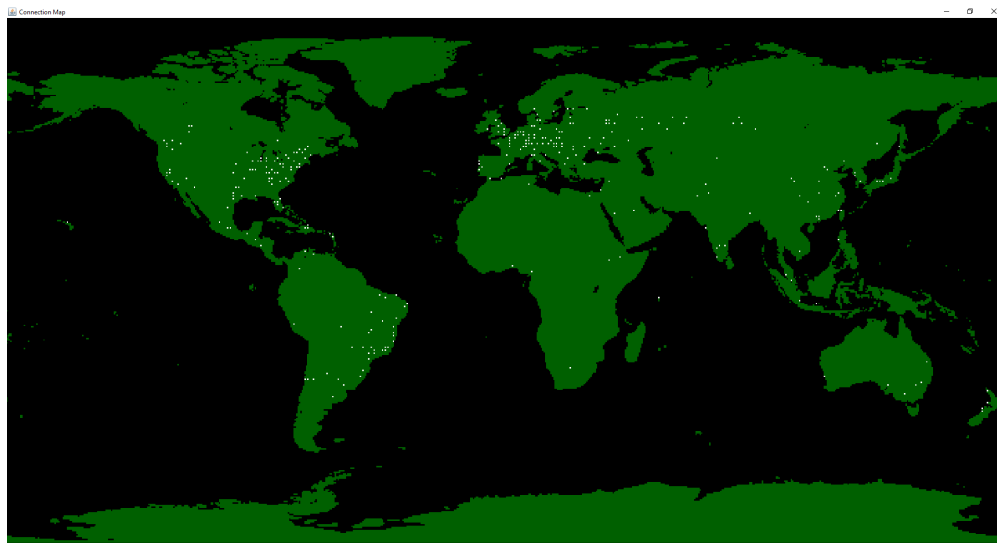


Figure 1: Connection Map

Connection Map is a pure-Java program to visualize network connections through a Cisco ASA firewall on a map. It uses the GeoLite2 “GeoIP” database available from <https://www.maxmind.com>.

Download the entire project from GitHub with the “Clone or download” button by clicking “Download ZIP.” Execute the runnable JAR at `Connection-Map/Connection_Map.jar`. The `Connection-Map/lib/` folder contains all necessary dependencies. Press `h` for “help” inside the program.

Connection Map passively listens on UDP port 514 for Syslog messages. The program actually opens UDP port 514 as a multicast on the non-standard group 239.5.1.4. Connection Map does not allow the operator to select which network interface it will bind to; binding decisions are left to the operating system. Microsoft Windows may bind the socket to an unexpected network interface. Use the command `netsh interface ipv4 show join` to observe which network interface the socket bound to. Npcap (included with Wireshark) may install a `Npcap Loopback Adapter` interface and VMware may install a `VMnet1` interface. These interfaces may have a faster “speed” and therefore lower route metric than the physical Ethernet and Wi-Fi interfaces (see <https://github.com/nmap/nmap/issues/1605>). If `netsh interface ipv4 show join` shows that 239.5.1.4 is joined on the incorrect interface, try disabling the other interface or change its metric (`Set-NetRoute`).

To configure a Cisco ASA to send Syslog messages to the Connection Map monitoring station, simply input the following commands in global configuration mode:

```
logging enable
logging trap debugging
logging host inside x.x.x.x
```

`x.x.x.x` is either the unicast IP address of the monitoring station or 239.5.1.4. The Connection Map program uses regular expressions to find Syslog messages 302013, 302014, 302015, and 302016 (see https://www.cisco.com/c/en/us/td/docs/security/asa/syslog/b_syslog/syslogs3.html). It assumes the external interface is named `outside` (case-sensitive). The name of the external interface is not configurable; if the name of the internal interface is not `outside` then the source code must be modified.

This program may be compatible with Cisco Firepower (see https://www.cisco.com/c/en/us/td/docs/security/firepower/Syslogs/b_fptd_syslog_guide/syslogs3.html).

Connection Map has limited utility in practice. This program is primarily “eye candy” for network operations centers, but it could be useful to give network operators a general sense of traffic patterns. For example, suppose a large organization has a data center on each continent. If a continental data center in, say, Japan began servicing an unusually large number of clients in North America, then this could indicate an availability or routing problem.

2 Route Monitor

Location: <https://github.com/wjholden/Route-Monitor>

Input: One or more network addresses, subnet masks, and network descriptions as space-separated command-line arguments.

Output: A graphical panel of squares showing route advertisements (see figure 2) and textual logs of routes learned or removed as standard output.

Language: Java 8

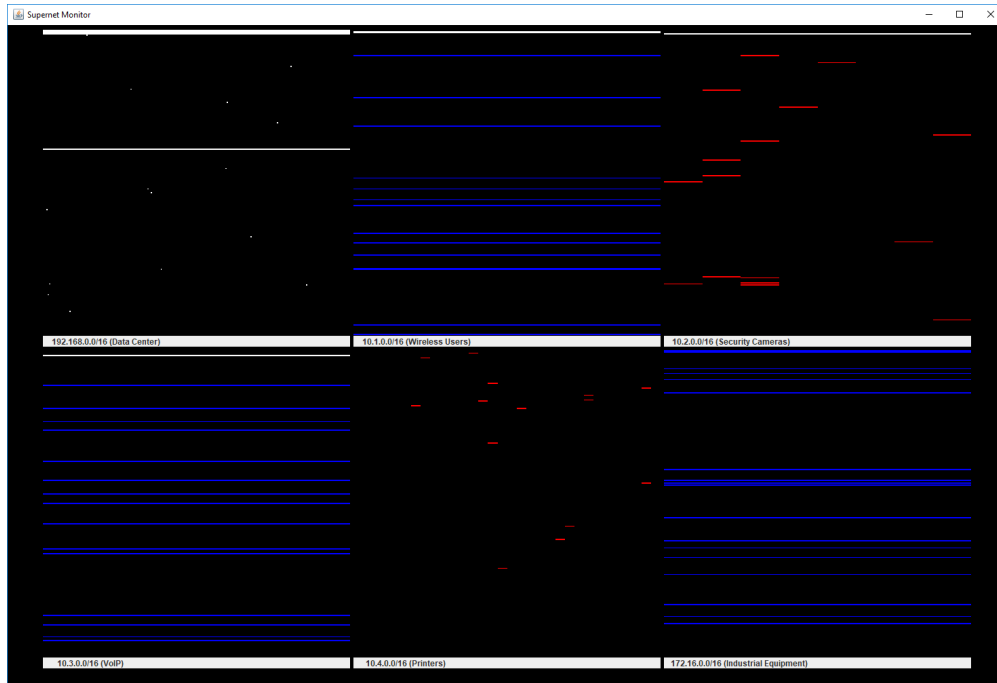


Figure 2: Route Monitor

Route Monitor builds upon a novel approach for visualizing route advertisements. Suppose a network operator wished to determine which subnets of `198.51.100.0/24` were known in the routing table of a network device. Traditionally, one might enter the command `show ip route 198.51.100.0 255.255.255.0 longer-prefixes`. In this program, we would represent each of the 256 addresses in this subnet as a pixel, arranged in a 16×16 grid in reading order. A pixel is colored black if it is not known in the routing table and white if it is. For a brief time, the pixel will be colored blue when it has been recently learned and colored red if it has been recently deleted.

Route Monitor uses the Routing Information Protocol (RIP) version 2. The program is written in pure Java and requires no external dependencies and no superuser privileges, although host-based firewall software may interfere with Route Monitor. The program binds to UDP port 520 and joins multicast group `224.0.0.9`. Multicast group membership in Windows is susceptible to the same interface metric problems in Route Monitor as it was in Connection Map. Troubleshoot group membership with `netsh interface ipv4 show join` and see section 1.

Few organizations use RIP in production. This is actually beneficial. A network operator may redistribute routes from their production network into RIP and refuse all routes learned on RIP. This allows Route Monitor to quickly and safely receive routing updates. RIP is a simple, mature, correct, and elegant dynamic routing protocol.

An example router configuration is shown below. This configuration uses an access list to filter any routes advertised on RIP. This prevents accidental or malicious changes to the router's forwarding table. OSPF routes from process number one are redistributed into RIP. Automatic summarization is turned off (which is essential) and version 2 is specified. The monitoring station running the Route Monitor software is assumed to be directly connected

on the network 198.51.100.0 from interface GigabitEthernet0/0. A key chain (not shown, but see section 5) is used as an additional control to prevent routes from being unintentionally learned on RIP.

```
access-list 1 deny any

router rip
  version 2
  no auto-summary
  network 198.51.100.0
  redistribute ospf 1
  distribute-list 1 in

interface GigabitEthernet0/0
  ip address 198.51.100.1 255.255.255.0
  ip rip authentication mode md5
  ip rip authentication key-chain GENERATED-KEY-CHAIN
```

Route Monitor can be configured to monitor networks of any size, but it was designed with the /16 in mind as a typical use case. Route Monitor is only compatible with IPv4.

A user story, which has repeatedly occurred in production, is that Route Monitor’s colors alert network operators of “flapping” networks. When pixels become red, then quickly turn blue, then red again, then blue — this indicates network instability. Flapping networks may have many causes, such as physical problems (such as a damaged fiber-optic cable), weather (such as strong winds moving a microwave antenna), and incorrect configuration (such as duplex or MTU mismatches).

In some cases, only a small subnet (such as a /30) will be repeatedly advertised and then “poisoned.” This may indicate the addresses of the individual link that is flapping. To assist with these problem investigations, Route Monitor outputs all route advertisements to standard output. An example of this output is shown below:

2019-09-09T20:58:14.171Z	192.168.128.0/30	16
2019-09-09T20:58:34.860Z	192.168.128.0/30	2
2019-09-09T21:00:11.460Z	192.168.128.0/30	16
2019-09-09T21:00:24.860Z	192.168.128.0/30	2
2019-09-09T21:00:27.299Z	192.168.128.0/30	16
2019-09-09T21:00:35.286Z	192.168.128.0/30	2

In this example, we see that the subnet 192.168.128.0/30 has repeatedly “bounced.” A problem investigation would reveal that the interface was repeatedly shutdown.

Textual output from the Route Monitor command might be saved to a file with the `tee` or `Tee-Object` commands. An example command to launch Route Monitor and redirect its output to a file is given below:

```
java -jar .\Route_Monitor.jar 192.168.0.0 255.255.0.0 "Data Center" 10.1.0.0
  255.255.0.0 "Wireless Users" 10.2.0.0 255.255.0.0 "Security Cameras"
  10.3.0.0 255.255.0.0 "VoIP" 10.4.0.0 255.255.0.0 "Printers" 172.16.0.0
  255.255.0.0 "Industrial Equipment" | Tee-Object history.txt
```

Press **f** to view this program in fullscreen and use the **+** and **-** keys to adjust the size of the panels. Fullscreen mode works correctly with Oracle's Java Runtime Environment (JRE), but there is a known bug with OpenJDK. Press **h** for additional help within the application.

3 lsdb.js

Location: <https://wjholden.com/lsdb> and <https://github.com/wjholden/lsdb.js>

Input: OSPF LSDB gathered by **snmpwalk**.

Output: OSPF LSDB in both DOT format and in graphical format (see figure 3).

Language: JavaScript

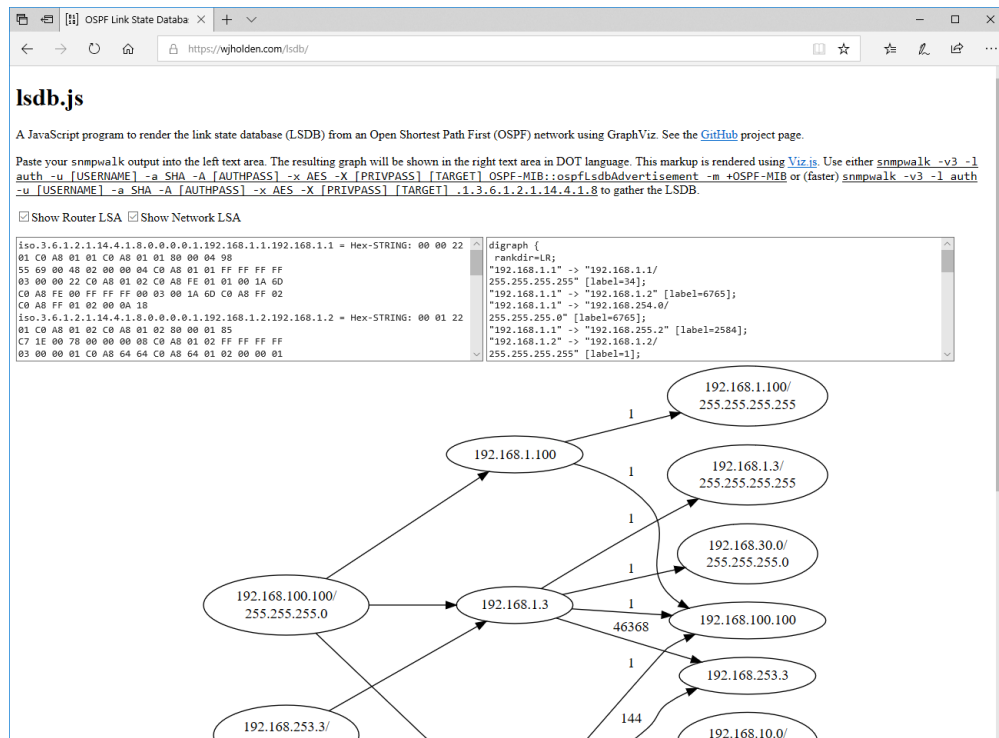


Figure 3: lsdb.js

lsdb.js is a derivative of an earlier Python program, Area Monitor (<https://github.com/wjholden/Area-Monitor>). Area Monitor began as an effort to peer with OSPF routers over IP protocol number 89. My first experiment with such low-level programming was several years ago with C (<https://github.com/wjholden/hello-ospf>). Java is only suitable for network programming in TCP and UDP. For protocols other than TCP and UDP, one needs access to raw sockets. Low-level network programming is very difficult. After significant research and experimentation, I realized that you can get the LSDB in its entirety directly from a router using SNMP.

The `snmpwalk` command is an easy and useful method to gather the LSDB. Either of the following commands are sufficient to gather the LSDB over SNMPv3. Use the latter if the OSPF-MIB is not available to `snmpwalk`. `lsdb.js` consists of only four total files (one `.html` and three `.js`) that can be hosted as static content anywhere or viewed locally.

```
snmpwalk -v3 -l auth -u [USERNAME] -a SHA -A [AUTHPASS] -x AES -X [PRIVPASS]
[TARGET] OSPF-MIB::ospfLsdbAdvertisement -m +OSPF-MIB
snmpwalk -v3 -l auth -u [USERNAME] -a SHA -A [AUTHPASS] -x AES -X [PRIVPASS]
[TARGET] .1.3.6.1.2.1.14.4.1.8
```

Paste the output from `snmpwalk` into the left text area.

`lsdb.js` translates the link state database to the DOT format used by GraphViz. The program can interpret router and network link state advertisements (LSAs). These LSAs may be switched on or off by toggling the checkboxes shown in the web page. The DOT output is shown in the second text area.

Finally, `lsdb.js` calls `Viz.js` (<http://viz-js.com/>) to render the DOT output as an image. `Viz.js` is an emscripten (<https://emscripten.org>) build of Graphviz (<https://www.graphviz.org>). The generated graph is dynamically added to the web page as shown in figure 3.

Operators may edit the text in both text areas to customize their graphic.

The layout of network diagrams is generally less than beautiful. The network diagrams generated might be useful to:

- Get started with a network diagram. Assign vertices to subgraphs and manually adjust their ordering of the vertices to find an appealing arrangement.
- Change detection. Run `snmpwalk` as a scheduled task on a regular basis. Store the output for later use in “spot-the-differences” troubleshooting.
- Discover mismatched interface costs. An interesting design feature (design flaw?) of OSPF is that interface costs may be mismatched. For example, routers R1, R2, and R3 may each be connected to the same broadcast network, but their the `ip ospf cost` configuration can be unequal. Unequal link costs may cause asymmetric routing. Link costs are shown as edge labels.

4 Node Monitor

Location: <https://github.com/wjholden/Node-Monitor>

Input: A list of hostnames or IP addresses to monitor as a file or as standard input.

Output: A table showing the reachability of each device (see figure 4).

Language: Java 8

Node Monitor is a simple pure-Java solution to “ping” network devices to test their availability. Technically, the method by which the IP addresses are polled is not specified. Reachability is determined by Java’s `java.net.InetAddress.isReachable` method (see <https://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html>). Node Monitor

192.0.2.12	192.0.2.22	The SNMP server 192.0.2.23	192.0.2.88	192.0.2.96
192.0.2.120	192.0.2.123	Some useful widget 192.0.2.188	My VPN gateway 198.51.100.3	198.51.100.60
198.51.100.89	198.51.100.100	A printer 198.51.100.132	198.51.100.143	198.51.100.173
198.51.100.190	Their call manager 203.0.113.2	203.0.113.8	203.0.113.34	Our DNS server 203.0.113.53
203.0.113.83	203.0.113.98	203.0.113.184	203.0.113.205	

Figure 4: Node Monitor

has no third-party dependencies, requires no installation, and does not require superuser privileges.

Node Monitor's configuration can be supplied as a file or as standard input. The format is a list of lines containing one to four tab-separated columns. Only the first column is mandatory. The first column is the IP address or hostname to be polled.

The second column is the timeout, in seconds, that the system waits until considering a ping failed. If not specified, the timeout defaults to two seconds.

The third column is the frequency, in seconds, with which we poll the node. If not specified, the frequency defaults to thrice the timeout.

The final column is a textual description of the node.

If supplying this configuration as a file, simply provide the file name as the only command-line argument for the program. An example configuration is shown below:

198.51.100.3	2	10	My VPN gateway
203.0.113.53	2	10	Our DNS server
192.0.2.23	5	30	The SNMP server
198.51.100.132	8	25	A printer
203.0.113.2	1	10	Their call manager
192.0.2.188	1	5	Some useful widget
198.51.100.60	2	10	
203.0.113.205	2		
192.0.2.88			

Node Monitor is not at all novel. There are dozens, perhaps hundreds, of commercial and open network monitoring solutions that provide superior functionality and a more appealing interface. The value Node Monitor brings is its simplicity. Given only a list of hostnames or IP addresses, a system administrator can begin monitoring the availability of their network in seconds. Node Monitor automatically sorts nodes in reading order by the numerical value of their IP addresses. Node Monitor organizes nodes in a left-complete square.

To create an input file for Node Monitor, simply enter the desired IP addresses or hostnames into a text file. Save the file to the same folder as `Node_Monitor.jar` and enter the command `java -jar Node_Monitor.jar FILENAME.TXT`.

Node Monitor's graphical user interface is not at all polished. Use the `+` and `-` keys to increase and decrease the font size. Use arrow keys (`↑↓←→`) to position text. Press `f` to enter fullscreen and press `q` to quit. Fullscreen is known to not work on OpenJDK.

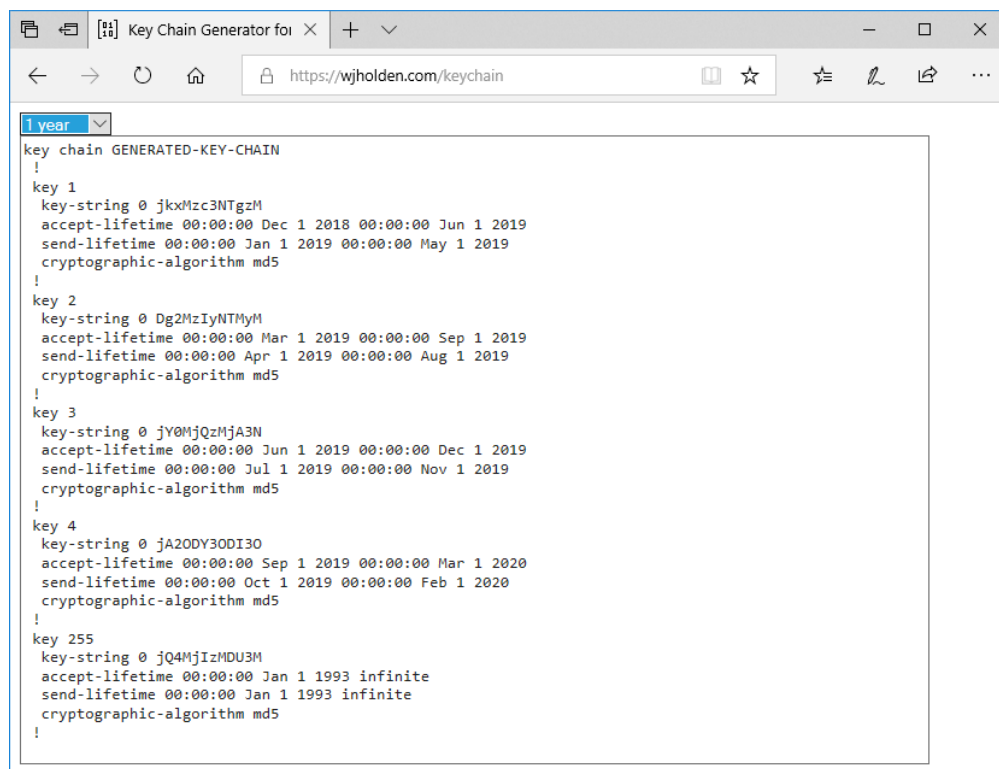
5 Key Chain Generator

Location: <https://wjholden.com/keychain>

Input: Number of years (1–10) of keys to generate.

Output: A key chain in the format Cisco IOS expects with three overlapping keys per year and one key (always key number 255) with an infinite lifetime (see figure 5).

Language: JavaScript



The screenshot shows a web browser window titled "Key Chain Generator for" with the address bar displaying "https://wjholden.com/keychain". A dropdown menu is set to "1 year". The main content area displays the following Cisco IOS key chain configuration:

```
key chain GENERATED-KEY-CHAIN
!
key 1
key-string 0 jkxMzc3NTgzM
accept-lifetime 00:00:00 Dec 1 2018 00:00:00 Jun 1 2019
send-lifetime 00:00:00 Jan 1 2019 00:00:00 May 1 2019
cryptographic-algorithm md5
!
key 2
key-string 0 Dg2MzIyNTMyM
accept-lifetime 00:00:00 Mar 1 2019 00:00:00 Sep 1 2019
send-lifetime 00:00:00 Apr 1 2019 00:00:00 Aug 1 2019
cryptographic-algorithm md5
!
key 3
key-string 0 jY0MjQzMjA3N
accept-lifetime 00:00:00 Jun 1 2019 00:00:00 Dec 1 2019
send-lifetime 00:00:00 Jul 1 2019 00:00:00 Nov 1 2019
cryptographic-algorithm md5
!
key 4
key-string 0 jA20DY30DI30
accept-lifetime 00:00:00 Sep 1 2019 00:00:00 Mar 1 2020
send-lifetime 00:00:00 Oct 1 2019 00:00:00 Feb 1 2020
cryptographic-algorithm md5
!
key 255
key-string 0 jQ4MjIzMjDU3M
accept-lifetime 00:00:00 Jan 1 1993 infinite
send-lifetime 00:00:00 Jan 1 1993 infinite
cryptographic-algorithm md5
!
```

Figure 5: Key Chain Generator

Network technicians should be familiar with using key chains to authenticate the En-

hanced Interior Gateway Routing Protocol (EIGRP) and Open Shortest Path First (OSPF) dynamic routing protocols. Key chain-based authentication has some security advantages, but it can be tedious and difficult to create a “perfect” key chain with overlapping keys. The Key Chain Generator creates random keys that are accepted for a six month period but sent for only four months. Key lifetime overlap is intended to allow tolerance for clock drift. The infinite lifetime key is required in by security guidelines in some organizations.

Interestingly, the behavior of router authentication in Cisco’s OSPF implementation is very different from its EIGRP implementation. According to https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_ospf/configuration/xr-3s/iro-xr-3s-book/iro-ospfv2-crypto-authen-xr.html, “OSPF selects the key that has the maximum life time” and a “key having an infinite lifetime is preferred.” By contrast, https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_eigrp/configuration/xr-3s/ire-xr-3s-book/ire-rte-auth.html states that in EIGRP “the software examines the key numbers in the order from lowest to highest, and uses the first valid key that it encounters.” The EIGRP behavior seems more closely aligned to one’s expectations.

Those who modify this software or its output should avoid using keys numbered 0 or anything greater than 255 if using OSPF. OSPF includes an eight-bit key ID in its authentication field (<https://tools.ietf.org/html/rfc2328#appendix-D>). Cisco’s IOS will send key 0 as default (empty) key if the configuration is incomplete and will log a OSPF-4-INVALIDKEY warning if received (see <https://packetlife.net/blog/2010/jun/1/ospfv2-authentication-confusion/>).

6 CommSync II Output Interpreter

Location: <https://wjholden.com/commsync>

Input: Output of commands `$SSTA*`, `$SIGP*`, `$AZL1*`, `$POS1*`, `$SVS1*`, and `$TPOP*`.

Output: Friendlier textual description of the data based on FEI-Zyfer’s specification of serial communication protocol in document 385-8002.

Language: JavaScript

The organization I worked for from 2017–2019 operates CommSync II devices (“CommSync”). The CommSync is a GPS and atomic clock which provides timing. Younger network technicians may be surprised to learn that timing was a major consideration in the planning and operation of traditional telecommunications (“telecom”) equipment. External timing was used to deliver both speed and reliability. Ethernet is a “best effort” protocol designed to tolerate collisions when two or more stations transmit at once. Furthermore, the physical layer of IEEE 802.3 uses Manchester coding to transmit timing in-band. By contrast, DS0 circuits aggregated into a DS1 did not provide in-band timing but needed to operate on precisely the same frequency. The solution used by telecoms was to synchronize the clocks of networking equipment to a very accurate (and very expensive) timing source using the Building Integrating Timing Supply (BITS) protocol. (I am not an expert on this subject. See this question on Quora for a few more details: <https://qr.ae/TWKtAn>).

The CommSync devices serve as Primary Reference Clock (PRS, see ITU G.811 <https://www.itu.int/rec/T-REC-G.811/>) for legacy Asynchronous Transfer Mode (ATM) and Synchronous Optical Networking (SONET) equipment. This organization operates Alcatel-

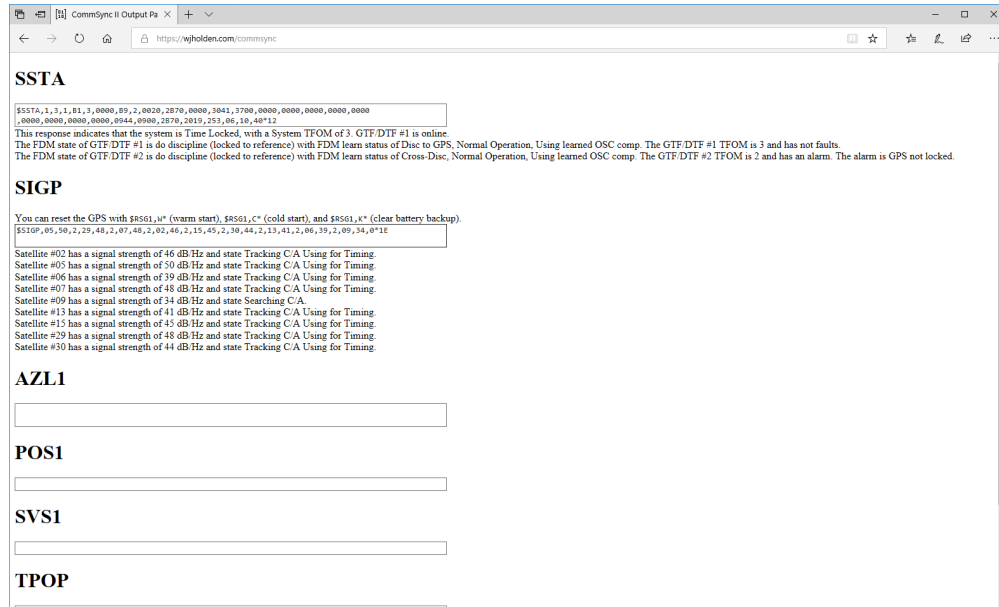


Figure 6: CommSync II Output Interpreter

Lucent 7270, 7470, 3600, 3630, and 1677 equipment. ATM control cards and SONET timing cards are “wire-wrapped” to T1/E1 ports of the CommSync using twisted-pair cables.

CommSync devices are managed using Telnet or SSH. The specification for the command-line interface is defined by the National Marine Electronics Association (NMEA). NMEA “sentences” begin of a “\$” symbol and end with “*”. All commands are four letters long.

The output of NMEA commands is not intended to be read by humans. The output of most commands is a list of comma-separated hexadecimal values. To assist with interpreting these outputs, I developed the CommSync II Output Interpreter program. This program is pure JavaScript. The supported commands are \$SSTA* (system status), \$SIGP* (satellite signal quality), \$AZL1* (azimuth and elevation), \$POS1* (position), \$SVS1* (satellite signal status), and \$TPOP* (8-port T1/E1 port configuration and status).

Contact the FEI-Zyfer, the manufacturer of the CommSync II, at techsupport@fei-zyfer.com or (714) 933-4003 to request detailed information. The document number for the user’s manual is 385-8000. The serial communication protocol is specified in document 385-8002. Optional modules for the CommSync II are described in document 385-8003.