Skype and Data Exfiltration

GIAC (GSEC) Gold Certification

Author: Kenneth G. Hartman, kgh@kennethghartman.com
Advisor: Antonios Atlasis

Accepted: April 18, 2014

Abstract
Few software packages have been as controversial, yet as ubiquitous as Skype. A common question on the Internet is whether Skype is safe for business. Skype makes extensive use of encryption. Encrypting traffic prevents intrusion detection systems and firewalls from inspecting the contents of the traffic. Therefore, an adversary can use Skype or traffic that simply resembles Skype traffic as the communication channel to exfiltrate a large amount of data off a network that permits Skype. Historically, miscreants have used and exploited Skype as a channel for a variety of nefarious purposes including data exfiltration. Microsoft has been active in addressing these abuses, but the overarching concern remains that Skype uses closed encryption in a highly distributed peer-to-peer network. Through the examination of prior research and utilization of tools and experimental observations, network operators can make the appropriate determination regarding the suitability of Skype for their own organizations.

1. Introduction

Few software packages have been as controversial, yet as ubiquitous as Skype. Skype is a P2P communication service that has grown to over 300 million users (Steele, 2013). A common question on the Internet is whether Skype is safe for business.[1] Many of the responses focus on the fact that Skype software is closed-source while others are concerned with the privacy of Skype communication (Masnick, 2012; Spirovski, 2008). However, there is another aspect of Skype that makes it an undesirable software package for a managed network that contains sensitive information. Skype uses standard cryptographic primitives to achieve strong end-to-end encryption (Berson, 2005). While this encryption provides the important benefit of protecting the information transferred, the Skype application sends its traffic to an ever-changing series of intermediate peer nodes around the globe (Skype Limited, 2010).

Encrypting traffic prevents intrusion detection systems and firewalls from inspecting the contents of the traffic (Fawcett, 2012). These technical controls are often a required component of secure network architecture. In order to support Skype traffic, the organization must configure their network to allow outgoing encrypted traffic to virtually any public IP address (Skype Limited, 2010).

Skype traffic consists of video, screen sharing, voice, file transfer, and instant messaging data and all of this traffic is encrypted (Skype Limited, 2010). Skype voice calls are typically about 30 kbps; however, video calls can consume up to 950 kbps (Zhang, et al., 2012). Therefore, an adversary can use Skype as the communication channel to exfiltrate a large amount of data off a network that permits Skype. Furthermore, if an adversary is able to create malicious traffic that simply has the same characteristics as Skype traffic, he would have a very high bandwidth covert communications channel that passes through the same network security devices that permit valid Skype traffic.

[1] A query of Google (http://www.google.com/#q=is+skype+secure+for+business) produced more than 46.1 million results on March 12, 2014.

Organizations that permit Skype traffic are accepting the risk, either knowingly or unknowingly, that a high volume of sensitive information could be leaving their network despite their investment in network security devices. Encryption nullifies the packet inspection capabilities of network security devices such as firewalls and intrusion detection systems (Baset & Schulzrinne, 2008). Although Microsoft is marketing Skype to the business community, and has taken measures to protect its $8.5 billion investment in the Skype franchise (Microsoft, 2011), there is a significant risk of data exfiltration via Skype traffic or more importantly, traffic that simply mimics the characteristics of Skype communication.

Virtually all organizations have some sort of sensitive information on their business network that they wish to protect from public disclosure. For example, typical businesses may be subject to HIPAA or PCI compliance requirements and need to determine if Skype is appropriate for their environment (Hayes, 2008). Security professionals and network administrators who have the challenge of articulating to management the risks of permitting Skype on their business network will find the following research summary to be an excellent resource to guide their decision.

2. Characteristics of Skype Traffic

2.1. Nodes

The Skype network consists of three types of nodes and a login server. The three types of nodes are ordinary nodes, super nodes, and relay nodes. The Skype client software is capable of functioning as any type of node depending on the type of network connection and on the configuration settings in the software (Skype Limited, 2010). The node types are as follows:

- **Ordinary Node –** A Skype client is an ordinary node when the Skype application communicates with another client, such as placing a call or sending a text message (Baset & Schulzrinne, 2008).
- **Super Node—**A super node is when the Skype client assumes additional duties in support of the P2P network, such as locating other nodes and users by functioning as a distributed directory service. A node can only become a

super node if it has a public IP address, adequate bandwidth, memory, and uptime. On Windows, it is also possible to prevent Skype from functioning as a super node by a registry setting (Skype Limited, 2010).

- **Relay Node—**Relay nodes pass communication traffic between nodes that cannot reach each other directly. This is typically employed because of firewalls and network address translation. (Skype Limited, 2010)

A small number of login servers perform user authentication (Skype Limited, 2010). The login server stores user names and passwords and is the only centralized component of the original Skype network. For a successful login, the ordinary node must register with the Skype login server as well as connect to at least one valid super node. (Baset & Schulzrinne, 2008). Currently, Skype also supports authentication via one's Microsoft Account (Skype, 2013) and integrates deeply with Facebook (Skype, 2010).

2.2. Login Process

The login process consists of two sub-processes. These are version verification and user authentication (Suh, Figueiredo, Kurose, & Towsley, 2005). To perform a version verification, the Skype clients sends a HTTP GET request containing the keyword "getlatestversion" to a server at ui.skype.com (Hayes, 2008). The user authentication process begins with the Skype client contacting known super nodes, to obtain the IP address information of the login server and then authenticates the user credentials via the login server (Baset & Schulzrinne, 2008).

2.3. Cryptography

In 2005, Skype engaged Tom Berson, an independent cryptography expert to review Skype's implementation of cryptography. Presumably, his analysis is still accurate because Skype still references it from their website. Unfortunately, because Skype is closed source, one cannot validate this assumption in a legal manner without Microsoft's cooperation. Berson reported that Skype uses the following cryptographic primitives:

- AES block Cipher
- RSA public key cryptosystem

- ISO 9796-2 signature padding scheme
- SHA-1 Hash function
- RC4 stream cipher

2.4. Firewall Circumvention

Baset and Schulzrinne (2008) observed that when Skype was installed on a MS Windows computer, a "host cache" was created in the registry at `HKEY_CURRENT_USER\ Software\Skype\Phone\Lib\Connection\HostCache.` They noticed that after logging in for the first time upon installation, the host cache was initialized with at least the same seven pairs of IP address and ports. These were designated as "bootstrap super nodes." They also observed that if the host cache was cleared subsequent to the initial login, the Skype client was no longer able to connect to the Skype network. Perhaps an even more interesting observation made by Baset and Schulzrinne was how Skype behaved when configured with a single invalid entry in the host cache. Their observation is depicted with the following psuedocode:

```
Send UDP Packet(s) to Host Cache IP Address at Host Cache Port
        If No Response within 5 Seconds, then...
Attempt TCP Connection with Host Cache IP Address at Host Cache Port
        If not connected then...
Attempt TCP Connection with Host Cache IP Address at Port 80 (HTTP)
        If not connected then...
Attempt TCP Connection with Host Cache IP Address at Port 443 (HTTPS)
        If not connected then...
Wait 6 Seconds and then repeat entire sequence up to 5 times
```

While this behavior enables Skype to communicate effectively behind most home and many business firewalls, this traffic pattern resembles port knocking. Port knocking is the technique of only opening a socket if a pattern of traffic is seen hitting specific ports in a specific sequence (Port Knocking, n.d.). Operators of secure networks should be watchful for signs of port knocking, as it may be a precursor to an attempt to exfiltrate data.

Michael Gough, a frequent contributer to computerworld.com regarding Skype security, summarized industry consternation regarding this behavior in an article titled, “How Dangerous is Skype?” as follows:

> “One of security professionals' primary concerns about Skype are it's so easy for a Skype client to find a way around a secure corporate firewall configuration. Skype does this by using ports 80 and 443, which are open in most firewalls to allow Web browsing. In addition, Skype may reroute traffic if the initial port assigned during the Skype installation isn't available. This makes blocking Skype at a firewall more difficult since the ports Skype uses can change as needed” (Gough, 2007).

To call another Skype user, the Skype client searches the distributed directory by querying known super nodes for the IP address of the computer that corresponds to the called user. It exchanges media traffic directly with the called computer if possible; otherwise, the application routes the traffic through a relay node to the called computer (Kho, Baset, & Schulzrinne, 2008).

When a Skype client comes online after a user signs in, it sends a notification to all of the Skype users that are in the buddy list of the user that just logged in (Kho, Baset, & Schulzrinne, 2008). An experiment in Section 4 will demonstrate that this can result in simultaneous traffic to a large number of peers when the buddy list has multiple entries.

Baset and Schulzrinne (2008) observed that depending on network configuration, the Skype client would contact as many as eight other nodes in addition to its super node while performing a seach for a Skype user. They also observed that the Skype client continued to exchange TCP or UDP packets with the peer that it is communicating with during periods of silence on a voice call. Likewise, the Skype client sends a TCP keep-alive message to its super node every 60 seconds.

2.5. Skype Protocol

Biondi and Desclaux (2006) performed a detailed analysis of the then-current version of Skype and presented their findings at BlackHat Europe 2006. This presentation detailed the reverse engineering counter-measures designed into the Skype application and disected the Skype network protocol. As part of the presentation, Biondi demonstrated an add-on for his Scapy[2] tool that could read and assemble Skype packets. Using the Scapy add-on, the researchers demonstrated how to query the Skype network to determine the IP addresses of the approximately 20,000 super nodes at that time. They even went on to postulate that it would be possible to create a rogue Skype network that could operate adjacent to the official Skype network. The following section discusses current efforts to achieve this and Microsoft's countermeasures.

In the Blackhat presentation, Biondi and Desclaux made the following observations about the network behavior of the Skype application stating that it is nearly impossible to distinguish normal Skype traffic from information exfiltration:

- Almost all of the traffic is encrypted.
- Because the Skype peer-to-peer network architecture includes relay nodes, the destination peer cannot be clearly identified.
- The Skype application generates traffic even when the software is not used.

Section 4 provides experimental evidence that demonstrate that these three characteristics of Skype highlighted at Blackhat EU 2006 are still present and therefore make Skype undesirable for a secure network.

3. Recent Developments

3.1. Covert Channel Research

As early as May 2007, developers using the Skype API started to contemplate whether Skype could be used as a covert VPN that would function like a SSH tunnel that takes advantage of Skype's encryption and firewall piercing technologies (Ppmotskula, 2007). In 2012, four researchers from the University of Waterloo implemented a similar idea in a proof of concept software that they called "SkypeMorph" that was designed to circumvent the censorship of Tor network users (Moghaddam , Li, Derakhshani, & Goldberg, 2012).

[2] http://www.secdev.org/projects/scapy/

The Tor network provides users with anonymous connections via the use of publicly known relay points (Dingledine, Mathewson , & Syverson , 2004). Because the IP addresses of the relay points are well known, it is also possible to connect to the Tor network by any one of multiple unlisted intermediate “bridge” points. SkypeMorph implements a “pluggable transport” that looks like a SOCKS proxy to the Tor client and disguises the traffic between the Tor Client and the bridge as a Skype Video call. The researchers acknowledged that every message appears to be random in an encrypted communication, such as a Skype call, because the encryption scheme will output a randomly distributed bit stream. To confirm the current validity of this claim, Section 4 includes visualizations of the entropy for each Skype connection observed.

Based on cited prior research, Moghaddam et al asserted that the SkypeMorph obfuscation layer would only need to mimic the packet sizes and the inter-arrival times of consecutive packets of a Skype video call. SkypeMorph accomplishes this by using traffic morphing to counter an adversary’s attempts to distinguish its traffic through statistical means. Although, SkypeMorph uses Skype to set up the covert channel between the Tor client and the bridge, the covert channel does not go through a Skype relay node. (Moghaddam, Li, Derakhshani, & Goldberg, 2012)

The developers of SkypeMorph decided not to use SkypeKit, which is a collection of API’s that exposes Skype functionality to devices and applications by allowing developers to deliver Skype functionality through the user interface of the developers’ products (Skype, 2013). As they stated it:

> “SkypeKit allows peers to exchange streaming data through the Skype network. However, the data sent to the Skype network might be relayed by other nodes in the network and this can impose an overhead on the Skype network, which is not desired. Therefore, we deliberately chose not to use this feature of SkypeKit. Skype-Morph (sic) data is sent directly from the client to the bridge; it is disguised as Skype data, but it is not sent over the Skype network.” (Moghaddam, Li, Derakhshani, & Goldberg, 2012)

Researchers at the Warsaw University of Technology achieved a covert channel of 1.8kbps by hiding data in the traffic generated by periods of silence in a Skype voice call, proving that legitimate Skype traffic is an effective carrier medium for steganography (Mazurczyk, Karas, & Szczypiorski, 2013).

3.2. Skype Botnet & Malware

A 2010 research paper titled "Take a deep breath: a stealthy, resilient and cost-effective botnet using Skype" demonstrated how to create a botnet using the Skype4Py python library for the Skype API. As part of their background material, the researchers discussed how Skype-related malware could leverage the API to propagate, transmit spam, and exploit all of the features of the API, including initiating or redirecting calls, chat, contact list queries, and even sending SMS messages (Nappa, Fattori, Balduzzi , Dell'Amico, & Cavallaro, 2010). Skype also includes file transfer capability (Skype Limited, 2010) so this feature can also be used by the API or a Skype plugin to automate data exfiltration.

Examples of the more notable Skype-related malware are:

- Peskyspy – a trojan that records Skype conversations as an MP3 and incorporates a backdoor so as to download them (Symantec, 2009).
- Pykspa – a worm that spreads by sending a chat message to all contacts in the contact list with the URL's of malicious websites (Symantec, 2007).
- Chatosky – another worm spread via Skype chat providing a malicious hyperlink (Symantec, 2006).

3.3. Microsoft Malware Countermeasures

On June 2, 2011, Microsoft filed a DMCA complaint of alleged copyright infringement on the blog site http://skype-open-source.blogspot.com claiming that it contains links to Skype's copyrighted source code and "may encourage spamming activities" (Chilling Effects, n.d.). This was in reaction to Efim Bushmanov's announcement earlier that same day that he was posting the results of his research with the aim of making Skype open source (Bushmanov, 2011).

As of July 2013, the Skype website states that, "SkypeKit is no longer accepting new developer registrations" (Skype, n.d.a) and instead of using the API, users are referred to Skype URI’s which require that all interaction be performed through the full Skype client. This recent change by Microsoft is likely an effort to control abuse of the Skype network along with their legal efforts to combat the reverse-engineered, “open-source” Skype.

3.4. User & Node Geolocation

A research paper titled “I Know Where You are and What You are Sharing: Exploiting P2P Communications to Invade Users’ Privacy” demonstrated that peer-to-peer communication, including Skype, can leak the called party’s IP even if the caller does not answer. The Skype network leaked the users last known IP address even when the user was offline for up to 72 hours. This paper’s authors note that an effective countermeasure is to force all Skype calls to go through a relay node. (Le Blond, Zhang, Legout, Ross, & Dabbous, 2011). Resolving the physical location from the IP address is relatively trivial using a program, such as the SimpleGeoIPcity.py script in the appendix. Section 5, Conclusion, discusses the data exfiltration ramifications of untrusted relay nodes.

On May 1, 2012, a researcher named Kostya Kortchinsky detected a major shift in the Skype peer-to-peer network. Kortchinsky used the Scapy add-on (presented at Blackhat EU 2006 and discussed above) to enumerate the current list of super nodes and discovered that there was only a little more than 10,000 super nodes down from over 48,000. He also determined that all of the super nodes were hosted on networks owned by Skype or Microsoft and that each super node could support in excess of 100,000 clients. (Kortchinsky, 2012a)

An Ars technica article that interviewed Kortchinsky regarding this discovery contained the following statement from a Skype executive:

> “As part of our ongoing commitment to continually improve the Skype user experience, we developed supernodes (sic) which can be located on dedicated servers within secure datacentres. This has not changed the underlying nature of Skype’s peer-to-peer (P2P) architecture, in which supernodes simply allow users to find one another (calls do not pass through supernodes). We believe this approach has immediate performance, scalability and availability benefits for the

hundreds of millions of users that make up the Skype community." (Goodin, 2012)

Kortchinsky posted the list of super nodes that he collected during his scan to PasteBin (http://pastebin.com/LgWsPUGe) but then five weeks later tweeted "Skype appears to have removed all (few exceptions) the supernode related code from 5.10.0.114" (Kortchinsky, 2012b). Based on his Twitter feed, he went to work at Microsoft shortly thereafter and still works there (Kortchinsky, n.d.).

3.5. Microsoft as Man in the Middle

Another tactic that Microsoft is using to control abuse is to use a man-in-the middle technique by providing the super nodes that facilitates Skype traffic. Prior to Microsoft's May 2011 acquisition of Skype, it had a patent pending on "legal intercept." A month after the acquisition, Computerworld (Vijayan, 2011) ran an online news story with the sub-title that read "'Legal Intercept' would allow it to silently record VoIP communications." This article contained the following statements from the Microsoft patent:

> "Data associated with a request to establish a communication is modified to cause the communication to be established via a path that includes a recording agent."
>
> "Modification may include, for example, adding, changing, and/or deleting data within the data. The data as modified is then passed to a protocol entity that uses the data to establish a communication session."

The Computerworld article quotes a law school professor who stated that the legal intercept patent is likely a move by Microsoft to comply with CALEA, the Communication Assistance for Law Enforcement Act (Vijayan, 2011), that requires the makers of communications equipment to enable their technology with real-time surveillance capabilities (Federal Communications Commission, n.d.).

Researchers associated with Heise Security discovered that a system at an IP address owned by Microsoft was accessing HTTPS URLs transmitted in Skype chat as evidenced by their web server logs (Heise Security, 2013). In response, Skype responded with an excerpt from its data protection policy:

"Skype may use automated scanning within Instant Messages and SMS to (a) identify suspected spam and/or (b) identify URLs that have been previously flagged as spam, fraud, or phishing links. (Skype, n.d.b)".

4. Observations

4.1. Version Verification

As stated in the *Characteristics of Skype Nodes* section, a Skype version verification occurs during the login process. Unlike most connections that Skype makes, this is plaintext HTTP and seems to serve as the means to ensure that the installed client is compatible with the versions required on the official Skype network. The Python program discussed in the next section extracted this content in the tcpflow output file named `192.168.043.074.49685-157.056.109.008.00080.txt`:

```
GET /ui/0/6.14.0.104./en/getlatestversion?ver=6.14.0.104
&uhash=10258c1a87e7e8753f8ad2640fa2c84f7 HTTP/1.1
User-Agent: Skype™ 6.14
Host: ui.skype.com
Cache-Control: no-cache
```

The presence of this HTTP Request on your network is a telltale indicator that Skype is present and blocking it is one means to prevent Skype from functioning (Hayes, 2008). However, once the login process has concluded, Skype will not repeat the request again until another login is required. For this reason, an attacker that mimics Skype traffic for data exfiltration is unlikely to bother to implement this message.

4.2. Entropy Analysis

In the Blackhat presentation discussed above, Biondi claimed that Skype encrypted almost all of its traffic. To demonstrate this, Wireshark captured packets from a Skype call between two lab systems. This packet capture contains traffic generated by exercising the following communication features of the Skype client:

- video,

- voice,
- file transfer, and
- instant messaging.

Appendix 4 contains a Python program named `Calculate-TCPflow-Entropy.py` that extracts the Skype application layer data for each TCP connection into a single file per connection and then generates a histogram to illustrate the entropy of each connection. It does this by automating `tcpflow`, a utility for creating stream files from a packet capture (Elson, 2001).

Entropy is a measurement of the randomness. In the field of cryptology, there are formal proofs that show that if an adversary can correctly distinguish an encrypted file from a file that is truly random with a greater than 50% probability, then the attacker has “the advantage.” The adversary can then exploit that advantage and possibly break the encryption. This concept of advantage applies to the mathematical analysis of encryption algorithms. (Cachin, 1977). However in the real world, files that contain random data have no utility in a file system, therefore it is highly probable that files with high entropy are actually encrypted or compressed.

Appendix 1 contains the full table that the `Calculate-TCPflow-Entropy.py` program generated. For convenient reference, Table 1 presents an excerpt of the first 10 records below. It lists the stream file name, the file size, and the entropy of the stream file. Note that the file name indicates the source and destination IP address and port. After consulting the full table of output in the appendix, one will notice the large number of connections that Skype made in this capture in less than eight minutes. Entropy of 8.0 indicates a perfectly random file with an equal distribution of all possible byte values from zero to 255. The program highlights any rows that have an entropy value of less than 7.0, indicating the presence of some plaintext or byte patterns.

The programmatically generated table contains hyperlinks. The hyperlink corresponding to the file name opens up the actual tcpflow output file as a text file in the browser. If one clicks a file name with high entropy, the text file will contain random characters because of the encryption. There are also several connections with lower entropy, typically to port 80. The tcpflow output file for most of these connections will show hypertext transfer protocol in plaintext. As with many web-enabled applications, this HTTP contains cookies and uniform resource indicators (URI's) that are random in appearance with undocumented purposes. A cursory examination shows that some of these connections are to support the embedded advertisements that the recent versions of Skype will display.

The preceding section discussed the login verification message that consists of a specially formatted GET request and the corresponding "HTTP/1.1 200 OK" message. Table 1 contains these connections as row 2 and row 1 respectively.

Table 1 - Entropy of TCPFlow Output Files

TCPFlow Output File	File Size	Entropy
157.056.109.008.00080-192.168.043.074.49685	279	5.18595641473
192.168.043.074.49685-157.056.109.008.00080	179	5.25797964902
157.056.106.210.00443-192.168.043.074.49686	4838	7.5192003656
192.168.043.074.49686-157.056.106.210.00443	601	7.5147519745
134.170.018.220.00443-192.168.043.074.49688	5175	7.57453985008
192.168.043.074.49688-134.170.018.220.00443	1551	7.85562976913
137.116.032.077.00443-192.168.043.074.49689	4803	7.57805992092
192.168.043.074.49689-137.116.032.077.00443	1566	7.81548553538
212.187.172.078.33033-192.168.043.074.49684	406	7.48101623018
192.168.043.074.49684-212.187.172.078.33033	567	7.68537525859

The full table in the appendix shows a large number of connections to or from port 443, which is the well-known port for HTTPS. There is a small amount of plaintext involved in setting up the TLS session, such as domain names. This accounts for the slightly lower entropy and is evident in the tcpflow output files.

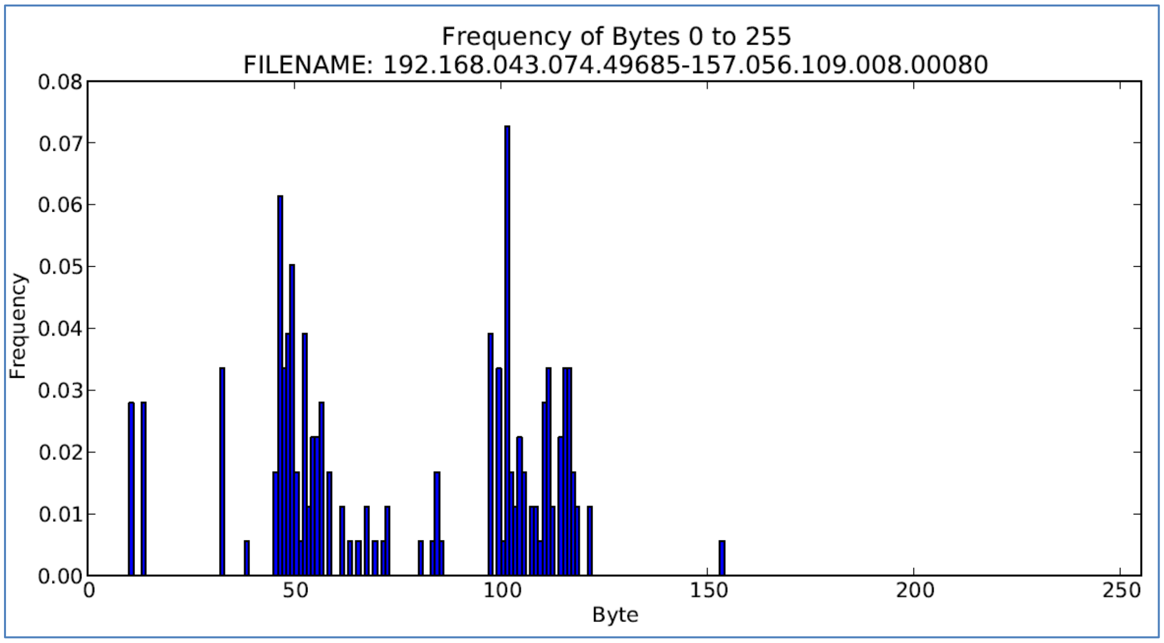


Figure 1 – Histogram of the Login Verification

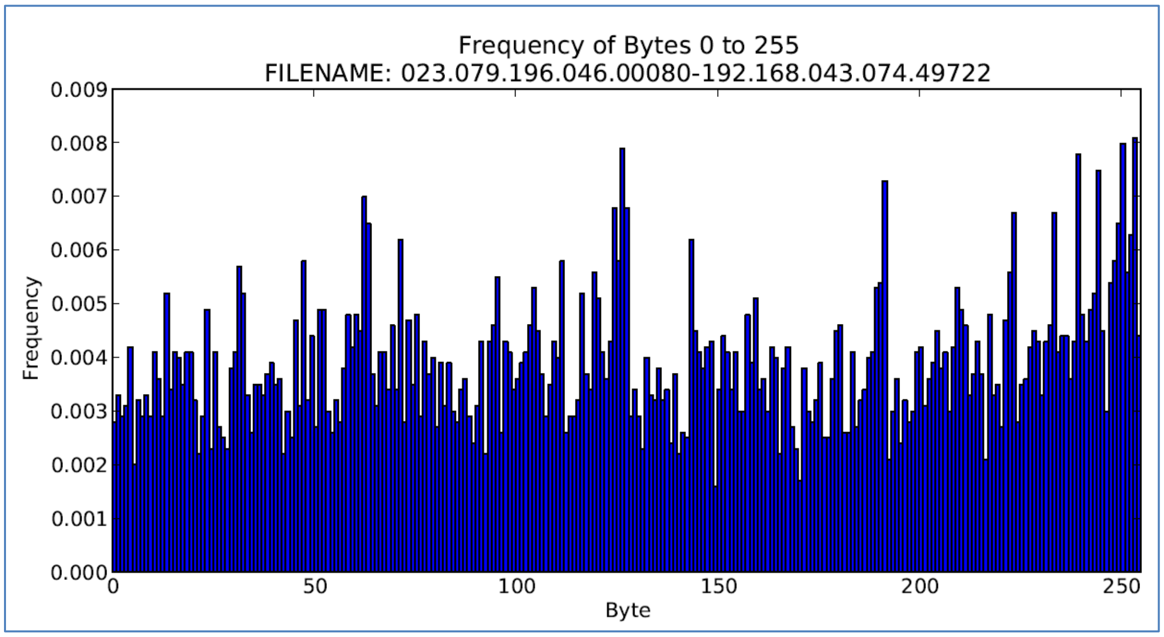


Figure 2 – Histogram of a typical Skype stream showing high entropy. (Note the frequency scale.)

The hyperlink associated with the Entropy value opens the histogram as a PDF document in the browser. Figure 1, above, contains the histogram for the login verification message and Figure 2 shows the histogram for a stream with a high entropy value.

Because the Login Verification is plaintext ASCII, the decimal byte values are less than 127 and the histogram in Figure 1 shows the relative frequency of each character. Contrast this with Figure 2, which shows a much more uniform distribution of decimal byte values in the range of 0 to 255. Examination of the associated tcpflow text file shows a random mixture of ASCII and Extended ASCII characters.

4.3. Traffic when Idle

The prior research discussed above also claims that Skype generates traffic when idle. To verify this is still the case, a lab system ran a PowerShell script (`Log-Connections.ps1`) to record all of the connections that Skype made during the experiment duration. After the login process completed, the lab system ran Skype unattended for four hours, with Wireshark capturing the packets. The remote connections logged by the PowerShell script became the filter for the Wireshark capture to ensure that only connections used by the Skype process were included in the analysis.

The Log-Connections.ps1 script generated a list of more than 2700 connections. These observations are available at http://www.kennethghartman.com/projects/Log-Connections/log-2014-03-12.txt. During the four-hour experiment, Skype made connections with 171 distinct IP addresses around the globe. To illustrate the geographic distribution of the remote IP addresses, `SimpleGeoIPcity.py` performed a lookup of the geolocation data for each IP address. Appendix 2 contains a complete listing of the location data for each remote IP address, while Table 2 contains an excerpt of some of the results.

Table 2 – Geolocation of Remote IP Addresses

IP Address	Country	Region	City	Latitude	Longitude
108.12.244.232	US	RI	Coventry	41.6933	-71.6395
108.129.55.203	US	GA	Albany	31.523	-84.3024
108.45.180.228	US	VA	Reston	38.9311	-77.3489
109.167.207.7	RU	66	Saint Petersburg	59.8944	30.2642
109.75.64.202	LB	2	Tyre	33.2711	35.1964
116.30.99.175	CN	30	Guangzhou	23.1167	113.25
119.160.118.20	PK			30	70
120.60.152.245	IN	16	Mumbai	18.975	72.8258
121.111.140.254	JP			35.69	139.69
124.6.181.42	PH	63	Tarlac	15.4802	120.5979

IP Address	Country	Region	City	Latitude	Longitude
128.211.249.130	US	IN	West Lafayette	40.4249	-86.9162
134.255.159.146	RU	66	Zarechny	52.1137	38.0953
135.23.226.25	CA	ON	Etobicoke	43.7	-79.5667
139.193.132.95	ID	4	Jakarta	-6.1744	106.8294
141.225.188.27	US	TN	Memphis	35.2017	-89.9715
151.213.191.20	US	TX	Sugar Land	29.5557	-95.6335
153.139.48.194	JP	40	Tokyo	35.685	139.7514
157.55.130.160	US	WA	Redmond	47.6801	-122.1206
157.55.130.166	US	WA	Redmond	47.6801	-122.1206

To examine the traffic pattern of all of the communication that Skype made to these remote IP addresses, a Python program named `Plot-IOStats.py` generated a traffic plot for each connection that had a payload. The results of this traffic analysis are in Appendix 3 and the program source code is in Appendix 6. The next sections discuss some of the interesting traffic patterns.

4.3.1. Spurious Spikes Traffic Pattern

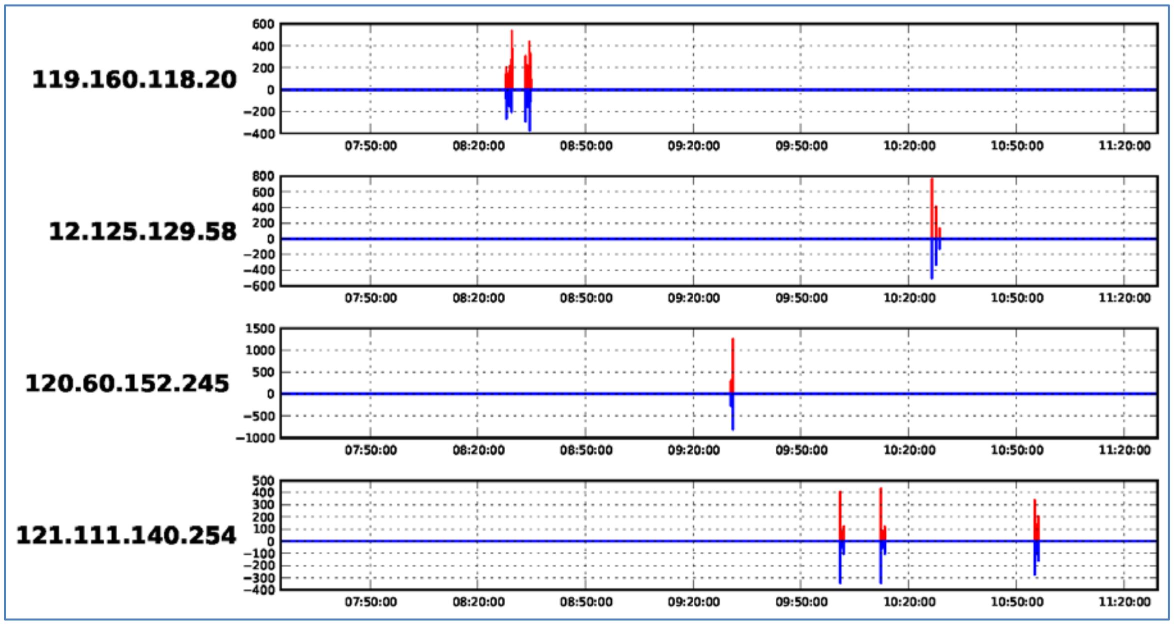


Figure 3 – Examples of the Spurious Spikes Traffic Pattern

Figure 3 depicts the most frequent type of encrypted traffic pattern, where there is just one or more spikes of a few hundred bytes exchanged. Note `Plot-IOStats.py` suppresses the labels for the X and Y-axes to achieve ten plots per page of output. The X-axis is the time of day and the Y-axis is the number of bytes transmitted. The red line indicates the bytes received while the blue line represents the bytes transmitted by the local Skype process.

4.3.2. Constant Flow Traffic Pattern

The observed results contain three examples of a continuous flow of data back and forth for a prolonged period. The plot shown in Figure 4 shows a stream that lasted over 90 minutes of the four hours when Skype was idle. This example is to IP address 223.218.6.137, which according to Table 4 in the appendix is in Tokyo, Japan.

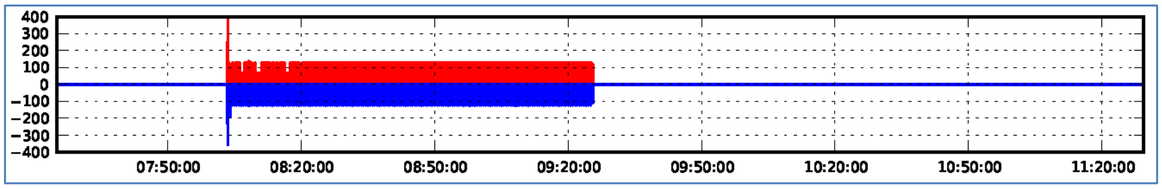


Figure 4 – The plot of traffic to 223.218.6.137 as an example of the Continuous Flow Traffic Pattern

4.3.3. Continuous Contact Traffic Pattern

Figure 5 contains an example of the last traffic pattern where there is continuous contact with the remote system but a relatively smaller amount of encrypted data passed.

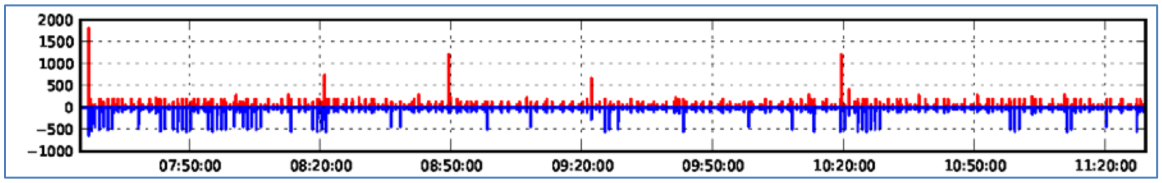


Figure 5 – Example of the Continuous Contact Traffic Pattern

The plot in Figure 5 is for IP Address 157.55.130.160 which is in a netblock owned by Microsoft according to the SANS ISC Whereis IP Country Lookup Tool (http://isc.sans.edu/tools/whereis.html). This is consistent with the TCP keepalive observed by Baset and Schulzrinne back in 2006.

5. Mitigation

The user has no control over which intermediate nodes that Skype will contact and no effective means of predicting those nodes, with the exception of the bootstrap super nodes in the host cache. Hence, whitelisting some specific intermediate nodes would not offer a way to operate Skype while preventing data exfiltration. Similarly, Data Loss Prevention (DLP) systems are not an effective control mainly due to the proprietary nature of the cryptographic methods used by Skype. However, there is at least one method to mitigate the risks of Skype traffic and traffic with similar characteristics. For organizations that have a legitimate business need to use Skype, one recommendation is to constrain Skype to networks and systems that are logically separated from the networks that process the sensitive information to be protected. The organization may then consider these networks out of scope for compliance requirements and decide that they do not need exfiltration controls. If this is not possible, these organizations should either accept the risk of data-exfiltration through Skype and Skype-like traffic, or use an alternative technology for VoIP communications.

6. Conclusion

The very same features that make Skype resilient and easy to use on diverse networks are the features that make Skype undesirable for networks that have higher security requirements. Users have exploited and misused Skype in ingenious ways. Microsoft has actively attempted to control that abuse. While Microsoft's efforts are undoubtedly noble, certain countermeasures, such as surreptitiously controlling the relay nodes raise additional concerns that decision makers should be cognizant of.

Some of the research discussed above included significant efforts to mimic the packet sizes and the inter-arrival times of consecutive packets of Skype traffic. However, for most networks that permit Skype traffic, this would not be necessary to exfiltrate data—simply encrypt the traffic and use seemingly random source and destination ports.

Because of the nature of peer to peer communications, the destination of Skype traffic and encrypted traffic that resembles Skype traffic is also a concern. If all Skype traffic only flowed through a relay nodes that are known to be controlled by Microsoft there would be a means of differentiating Skype traffic from traffic that attempts to mimic it because it could not flow through a legitimate Skype relay node.

The operators of managed business networks that contain sensitive information must know the nature and destination of the encrypted traffic leaving their network. Utilizing this synopsis of research and trends involving Skype, as well as the tools and analytical approach provided, network operators can make a well-informed risk-based decision regarding Skype and similar types of software for their own organization.

7. References

Baset, S. A., & Schulzrinne, H. (2008, September 15). *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol.* Retrieved January 19, 2013, from http://www1.cs.columbia.edu/~library/TR-repository/reports/reports-2004/cucs-039-04.pdf

Berson, T. (2005, October 18). *Skype Security Evaluation.* Retrieved January 5, 2013, from Skype.com: http://download.skype.com/share/security/2005-031%20security%20evaluation.pdf

Bionde, P., & Desclaux, F. (2006, March 2). *Silver Needle in the Skype.* Retrieved January 19, 2013, from http://www.secdev.org/conf/skype_BHEU06.handout.pdf

Bushmanov, E. (2011, June 2). *Skype protocol reverse engineered, source available for download*. Retrieved from skype-open-source: http://skype-open-source.blogspot.com/2011/06/skype-protocol-reverse-engineered.html

Cachin, C. (1977). *Entropy measures and unconditional security in cryptography*. Zürich: Diss. Techn. Wiss. ETH Zürich,. doi: http://dx.doi.org/10.3929/ethz-a-001806220

Caukin, J. (2012, March 5). *35 Million People Concurrently Online on Skype*. Retrieved January 13, 2013, from blogs.skype.com: http://blogs.skype.com/en/2012/03/35_million_people_concurrently.html

Chilling Effects. (n.d.). *Skype DMCA (Copyright) Complaint to Google*. Retrieved from Chilling Effects: http://www.chillingeffects.org/dmca512c/notice.cgi?NoticeID=89716

Dingledine, R., Mathewson , N., & Syverson , P. (2004). Tor: the second-generation onion router. *SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium - Volume 13* (pp. 21-21). Berkeley, CA, USA: USENIX Association.

Elson, J. (2001, March 1). *tcpflow*. Retrieved from www.circlemud.org: http://www.circlemud.org/jelson/software/tcpflow/tcpflow.1.html

Fawcett, T. W. (2012). *ExFILD: a tool for the detection of data exfiltration using entropy and encryption characteristics of network traffic*. University of Delaware, Department of Electrical & Computer Engineering. Delaware: University of Delaware. Retrieved January 5, 2013, from http://dspace.udel.edu:8080/dspace/handle/19716/5838

Federal Communications Commission. (n.d.). *Communications Assistance for Law Enforcement Act*. Retrieved from FCC Encyclopedia: http://www.fcc.gov/encyclopedia/communications-assistance-law-enforcement-act

Goodin, D. (2012, May 1). *Skype replaces P2P supernodes with Linux boxes hosted by Microsoft (updated)*. Retrieved from Ars Technica: http://arstechnica.com/business/news/2012/05/skype-replaces-p2p-supernodes-with-linux-boxes-hosted-by-microsoft.ars

Gough, M. (2007, March 6). *How dangerous is Skype?* Retrieved January 19, 2013, from Computerworld: http://www.computerworld.com/s/article/9012239/How_dangerous_is_Skype_?

Hayes, B. (2008, October 9). *Skype: A Practical Security Analysis*. Retrieved January 19, 2013, from SANS Institute: http://www.sans.org/reading_room/whitepapers/voip/skype-practical-security-analysis_32918

Heise Security. (2013, May 14). *Skype with care – Microsoft is reading everything you write*. Retrieved August 4, 2013, from The H Security: http://www.h-online.com/security/news/item/Skype-with-care-Microsoft-is-reading-everything-you-write-1862870.html

Kho, W., Baset, S. A., & Schulzrinne, H. (2008, April). Skype Relay Calls: Measurements and Experiments. *INFOCOM Workshops 2008* (pp. 1-6, 13-18). IEEE. doi:10.1109/INFOCOM.2008.4544646

Kortchinsky, K. (2012a, May 1). *Skype does away with random supernodes*. Retrieved from Expert: Miami: http://expertmiami.blogspot.com/2012/05/skype-does-away-with-random-supernodes.html

Kortchinsky, K. (2012b, June 14). *Tweet*. Retrieved from Twitter.com: https://twitter.com/crypt0ad/status/213327965214343169

Kortchinsky, K. (n.d.). *Google+*. Retrieved August 4, 2013, from Google+: https://plus.google.com/107438048326719758492/about

Le Blond , S., Zhang, C., Legout, A., Ross, K., & Dabbous, W. (2011). I know where you are and what you are sharing: exploiting P2P communications to invade users' privacy. *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement* (pp. 45-60). New York, NY, USA: ACM.

Masnick, M. (2012, July 27). *Clearing The Air On Skype: Most Of What You Read Was Not Accurate, But There Are Still Reasons To Worry*. Retrieved January 5, 2013, from www.techdirt.com: http://www.techdirt.com/articles/20120726/19283519848/clearing-air-skype-most-what-you-read-was-not-accurate-there-are-still-reasons-to-worry.shtml

Mazurczyk, W., Karas, M., & Szczypiorski, K. (2013). SkyDe: a Skype-based Steganographic Method. *International Journal of Computers, Communications & Control (IJCCC)*, 389-400.

Microsoft. (2011, May 10). *Microsoft to Acquire Skype*. Retrieved from www.microsoft.com: http://www.microsoft.com/en-us/news/press/2011/may11/05-10corpnewspr.aspx

Moghaddam , H. M., Li, B., Derakhshani, M., & Goldberg, I. (2012). SkypeMorph: Protocol Obfuscation for Tor Bridges. *Proceedings of the 2012 ACM Conference on Computer and Communications Security* (pp. 97-108). New York, NY, USA : ACM.

Nappa, A., Fattori, A., Balduzzi , M., Dell'Amico, M., & Cavallaro, L. (2010). Take a deep breath: a stealthy, resilient and cost-effective botnet using skype. *DIMVA'10 Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment* (pp. 81-100). Heidelberg, Germany: Springer-Verlag.

Port Knocking. (n.d.). Retrieved from portknocking.org: http://portknocking.org/

Ppmotskula. (2007, May 4). *Idea: skypetunnel*. Retrieved from Skype Blogs: http://blogs.skype.com/2007/05/04/idea-skypetunnel/

Skype. (2010, October 14). *Skype with Facebook integration and group video calling*. Retrieved from Blogs.Skype.com: http://blogs.skype.com/2010/10/14/new-skype/

Skype. (2013, February 13). *Skype and Messenger Coming Together: The Next Chapter*. Retrieved from Blogs.Skype.Com: http://blogs.skype.com/2013/02/15/skype-and-messenger-coming-together-the-next-chapter/

Skype Limited. (2010). *IT Administrators Guide.* Retrieved January 5, 2013, from Skype.com: http://download.skype.com/share/business/guides/skype-it-administrators-guide.pdf

Skype. (n.d.a). *What is SkypeKit?* Retrieved July 27, 2013, from Skype: https://support.skype.com/en/faq/FA12322/what-is-skypekit

Skype. (n.d.b). *Skype Privacy Policy*. Retrieved August 04, 2013, from Skype.com: http://www.skype.com/en/legal/privacy/

Spirovski, B. (2008, February). *Is Skype a Good Corporate Tool?* Retrieved from www.shortinfosec.net: http://www.shortinfosec.net/2008/02/is-skype-good-corporate-tool.html

Steele, E. (2013, August 28). *Skype Celebrates a Decade of Meaningful Conversations!* Retrieved from blogs.Skype.com: http://blogs.skype.com/2013/08/28/skype-celebrates-a-decade-of-meaningful-conversations/

Suh, K., Figueiredo, D. R., Kurose, J., & Towsley, D. (2005, July 11). *Characterizing and detecting relayed traffic: A case study using Skype.* Retrieved January 19, 2013, from ftp://gaia.cs.umass.edu/pub/Suh05_relay.pdf

Symantec. (2006, December 18). *W32.Chatosky - Technical Details*. Retrieved from Symantec Security Response: http://www.symantec.com/security_response/writeup.jsp?docid=2006-121910-5339-99&tabid=2

Symantec. (2007, September 10). *W32.Pykspa.D - Technical Details*. Retrieved from Symantec Security Response: http://www.symantec.com/security_response/writeup.jsp?docid=2007-091011-2911-99&tabid=2

Symantec. (2009, August 27). *Trojan.Peskyspy - Technical Details*. Retrieved from Symantec Security Response: http://www.symantec.com/security_response/writeup.jsp?docid=2009-082710-4600-99&tabid=2

Vijayan, J. (2011, June 28). *Microsoft seeks patent for spy tech for Skype*. Retrieved from ComputerWorld: http://www.computerworld.com/s/article/9218002/Microsoft_seeks_patent_for_spy_tech_for_Skype

Zhang, X., Xu, Y., Hu, H., Liu, Y., Guo, Z., & Wang, Y. (2012). Profiling Skype video calls: Rate control and video quality. *In Proceedings of INFOCOM*, (pp. 621-629). Retrieved from http://eeweb.poly.edu/faculty/yongliu/docs/skype_congestion_infocom.pdf

8. Appendices

8.1. Appendix 1 - Entropy of TCPFlow Output Files

The following table contains a complete list of all of the files generated by the `Calculate-TCPflow-Entropy.py` program for the experiment discussed in Section 4.2.

Table 3 – Full List of Entropy of TCPFlow Output Files

TCPFlow Output File	File Size	Entropy
157.056.109.008.00080-192.168.043.074.49685	279	5.18595641473
192.168.043.074.49685-157.056.109.008.00080	179	5.25797964902
157.056.106.210.00443-192.168.043.074.49686	4838	7.5192003656
192.168.043.074.49686-157.056.106.210.00443	601	7.5147519745
134.170.018.220.00443-192.168.043.074.49688	5175	7.57453985008
192.168.043.074.49688-134.170.018.220.00443	1551	7.85562976913
137.116.032.077.00443-192.168.043.074.49689	4803	7.57805992092
192.168.043.074.49689-137.116.032.077.00443	1566	7.81548553538
212.187.172.078.33033-192.168.043.074.49684	406	7.48101623018
192.168.043.074.49684-212.187.172.078.33033	567	7.68537525859
192.168.043.074.49696-207.046.011.151.00443	484	7.14407088779
078.141.177.069.12350-192.168.043.074.49701	194	6.93213513589
192.168.043.074.49701-078.141.177.069.12350	556	7.65740957145
192.168.043.074.49705-072.074.054.110.32766	126	6.53073498297
072.074.054.110.32766-192.168.043.074.49705	170	6.87686549488
192.168.043.074.49706-068.225.032.189.48665	130	6.60494733221
068.225.032.189.48665-192.168.043.074.49706	143	6.75770507801

192.168.043.074.49703-212.008.166.036.12350	1067	7.83469672057
212.008.166.036.12350-192.168.043.074.49703	157	6.67016788254
091.190.218.017.00080-192.168.043.074.49702	1183	7.07364542005
031.013.074.128.00443-192.168.043.074.49698	797	7.6830092787
192.168.043.074.49708-193.095.154.038.12350	714	7.70857829576
193.095.154.038.12350-192.168.043.074.49708	1186	7.82536696334
192.168.043.074.49690-023.063.158.161.00443	2287	7.88475139306
023.063.158.161.00443-192.168.043.074.49690	4886	7.67859979924
204.154.110.079.00443-192.168.043.074.49717	4684	7.34796091701
192.168.043.074.49717-204.154.110.079.00443	1018	7.64756488992
012.129.210.071.00080-192.168.043.074.49719	3460	7.77935320631
192.168.043.074.49719-012.129.210.071.00080	304	5.56470023004
012.129.210.071.00080-192.168.043.074.49721	571	5.48337975552
192.168.043.074.49721-012.129.210.071.00080	534	5.57642934354
204.154.111.011.00080-192.168.043.074.49725	1922	5.52009224933
192.168.043.074.49725-204.154.111.011.00080	696	5.61047904829
131.253.040.047.00443-192.168.043.074.49727	5514	7.61229204654
192.168.043.074.49727-131.253.040.047.00443	936	7.62322403578
178.255.083.001.00080-192.168.043.074.49733	859	7.01174295998
192.168.043.074.49733-178.255.083.001.00080	234	5.83495640147
178.255.083.001.00080-192.168.043.074.49734	859	6.9866968062
192.168.043.074.49734-178.255.083.001.00080	235	5.73685864818

178.255.083.001.00080-192.168.043.074.49736	859	6.93994368314
192.168.043.074.49736-178.255.083.001.00080	231	5.71795422516
165.193.093.108.00443-192.168.043.074.49735	15046	7.94221302789
012.129.210.071.00080-192.168.043.074.49739	637	5.57260607849
192.168.043.074.49739-012.129.210.071.00080	620	5.51683230027
138.108.007.020.00443-192.168.043.074.49723	4019	7.54761850819
192.168.043.074.49715-194.165.188.080.12350	452	7.46241629772
194.165.188.080.12350-192.168.043.074.49715	328	7.28359997553
192.168.043.074.49740-012.129.210.071.00080	652	5.56775556668
012.129.210.071.00080-192.168.043.074.49740	525	5.53890311372
023.079.196.046.00080-192.168.043.074.49716	3649	7.84142764759
023.079.196.046.00080-192.168.043.074.49722	10016	7.93707713071
023.050.075.027.00080-192.168.043.074.49718	2190	7.16190751757
192.168.043.074.49718-023.050.075.027.00080	237	5.69442717804
063.088.100.144.00080-192.168.043.074.49726	2983	7.8693141844
063.085.036.024.00080-192.168.043.074.49720	85086	7.99166734723
192.168.043.074.49741-137.116.032.214.00443	1385	7.75653132651
209.247.230.174.00443-192.168.043.074.49732	4334	7.48519286601
192.168.043.074.49746-212.187.172.059.00080	5	1.92192809489
192.168.043.074.49751-212.008.166.019.00080	5	1.92192809489
192.168.043.074.49749-189.193.249.071.51738	200	6.98366068969
189.193.249.071.51738-192.168.043.074.49749	90	6.12790990186

063.140.035.166.00080-192.168.043.074.49759	2320	5.72843005871
195.046.253.253.12350-192.168.043.074.49758	15164	7.98834474969
192.168.043.074.49758-195.046.253.253.12350	1505	7.88505555992
091.190.218.019.12351-192.168.043.074.49752	10093	7.98265526398
192.168.043.074.49752-091.190.218.019.12351	1018	7.83391155109
192.168.043.074.49712-023.063.158.161.00443	2415	7.873472685
023.063.158.161.00443-192.168.043.074.49712	3894	7.53635649528
131.253.040.047.00443-192.168.043.074.49763	5514	7.60158869282
192.168.043.074.49763-131.253.040.047.00443	936	7.65201213656
192.168.043.074.49765-193.095.154.038.12350	1676	7.87023994127
192.168.043.074.49687-091.190.216.063.12350	1940	7.89410075384
192.168.043.074.49692-134.170.024.094.00443	25182	7.99162967549
134.170.024.094.00443-192.168.043.074.49692	30477	7.97477537505
091.190.216.063.12350-192.168.043.074.49687	1765	7.88356703803
192.168.043.074.49683-111.221.077.150.40008	4909	7.95703073888
192.168.043.074.49704-098.116.058.052.18796	5600	7.97112650145
192.168.043.074.49707-173.053.063.167.02370	1932	7.9071578287
192.168.043.074.49709-074.103.067.168.30595	1053	7.78110282629
192.168.043.074.49710-098.113.144.055.23006	993	7.8184240553
192.168.043.074.49711-071.199.041.095.23957	883	7.82241524032
173.053.063.167.02370-192.168.043.074.49707	1097	7.80637911042
098.116.058.052.18796-192.168.043.074.49704	8232	7.97850633576

074.103.067.168.30595-192.168.043.074.49709	948	7.82973682616
098.113.144.055.23006-192.168.043.074.49710	1028	7.83345844551
071.199.041.095.23957-192.168.043.074.49711	915	7.78322034217
168.061.160.096.00443-192.168.043.074.49766	4803	7.57470267239
192.168.043.074.49766-168.061.160.096.00443	1118	7.68744974823
111.221.077.150.40008-192.168.043.074.49683	1797	7.88737802023
023.063.203.032.00443-192.168.043.074.49737	4236	7.61261226745
192.168.043.074.49732-209.247.230.174.00443	1268	7.75381023588
192.168.043.074.49764-131.253.034.154.00443	1013	7.67706447286
192.168.043.074.49759-063.140.035.166.00080	2637	5.5022074076
192.168.043.074.49697-207.046.011.151.00443	873	7.59189549873
023.063.157.195.00443-192.168.043.074.49694	4021	7.55289042623
192.168.043.074.49753-023.063.159.240.00443	2313	7.87248879208
192.168.043.074.49713-023.063.159.240.00443	1511	7.81850422773
192.168.043.074.49699-131.253.040.050.00443	885	7.61486461554
192.168.043.074.49742-137.116.032.214.00443	1832	7.83339886201
065.055.005.231.00443-192.168.043.074.49762	7325	7.61675632688
192.168.043.074.49730-023.063.203.120.00443	916	7.63892906837
131.253.034.154.00443-192.168.043.074.49764	5119	7.50678472752
192.168.043.074.49695-137.116.032.077.00443	1374	7.76691117575
137.116.032.077.00443-192.168.043.074.49695	5059	7.60751076864
192.168.043.074.49735-165.193.093.108.00443	988	7.66568820025

193.095.154.038.12350-192.168.043.074.49765	759	7.73630217565
192.168.043.074.49738-023.063.203.032.00443	2081	7.87052923859
192.168.043.074.49714-065.055.005.231.00443	944	7.6658743385
192.168.043.074.49698-031.013.074.128.00443	988	7.69837377239
023.063.203.120.00443-192.168.043.074.49731	422	7.36888083162
023.063.203.120.00443-192.168.043.074.49730	20665	7.97239161704
023.063.203.120.00443-192.168.043.074.49729	4681	7.67102672799
192.168.043.074.49724-023.079.196.046.00443	913	7.6413847484
023.063.203.120.00443-192.168.043.074.49728	4009	7.57046038831
192.168.043.074.49726-063.088.100.144.00080	235	5.35754528132
192.168.043.074.49693-168.061.145.190.00443	1759	7.8394321477
192.168.043.074.49716-023.079.196.046.00080	284	5.40720223581
023.063.159.240.00443-192.168.043.074.49713	3145	7.51554166438
192.168.043.074.49754-023.063.159.240.00443	517	7.25415740109
192.168.043.074.49760-023.063.158.161.00443	2527	7.88966734671
192.168.043.074.49722-023.079.196.046.00080	436	5.47373046825
192.168.043.074.49756-023.063.159.240.00443	549	7.31932931869
192.168.043.074.49731-023.063.203.120.00443	825	7.59865761358
192.168.043.074.49728-023.063.203.120.00443	2025	7.86740891446
192.168.043.074.49702-091.190.218.017.00080	691	5.5888673831
131.253.040.044.00443-192.168.043.074.49700	5096	7.52793968899
137.116.032.214.00443-192.168.043.074.49741	9002	7.84351232903

137.116.032.214.00443-192.168.043.074.49742	9614	7.86283602625
192.168.043.074.49691-023.079.207.139.00443	880	7.60075850528
131.253.040.050.00443-192.168.043.074.49699	5253	7.53567910325
023.073.031.240.00443-192.168.043.074.49761	2852	7.43497926447
192.168.043.074.49761-023.073.031.240.00443	978	7.65148441802
023.079.196.046.00443-192.168.043.074.49724	6674	7.80951689048
168.061.145.190.00443-192.168.043.074.49693	4889	7.58702677541
192.168.043.074.49720-063.085.036.024.00080	1332	5.63311195841
207.046.011.151.00443-192.168.043.074.49697	2753	7.38310948914
207.046.011.151.00443-192.168.043.074.49696	2636	7.34333865268
023.079.207.139.00443-192.168.043.074.49691	3612	7.4767396122
192.168.043.074.49694-023.063.157.195.00443	898	7.63115965814
192.168.043.074.49755-023.063.159.240.00443	549	7.3329197216
192.168.043.074.49757-023.063.159.240.00443	549	7.37445490292
192.168.043.074.49700-131.253.040.044.00443	987	7.66234521776
192.168.043.074.49729-023.063.203.120.00443	2329	7.88618876858
023.063.158.161.00443-192.168.043.074.49760	3857	7.51992001873
065.055.005.231.00443-192.168.043.074.49714	8560	7.69719044777
023.063.159.240.00443-192.168.043.074.49757	2502	7.9240559236
023.063.159.240.00443-192.168.043.074.49756	2182	7.91965000462
023.063.159.240.00443-192.168.043.074.49755	9606	7.9812634929
023.063.159.240.00443-192.168.043.074.49754	19227	7.9895687701

023.063.159.240.00443-192.168.043.074.49753	106139	7.99765685862
192.168.043.074.49737-023.063.203.032.00443	1324	7.78358818913
192.168.043.074.49723-138.108.007.020.00443	918	7.59001566866
192.168.043.074.49762-065.055.005.231.00443	944	7.62400045232
023.063.203.032.00443-192.168.043.074.49738	3873	7.55406715175

8.2. Appendix 2 - Remote IP Address Geolocation Data

The following table contains a complete list of the location data generated by the `SimpleGeoIPcity.py` program for the experiment discussed in Section 4.3.

Table 4 – Geolocation of Remote IP Addresses

IP Address	Country	Region	City	Latitude	Longitude
107.107.191.18	US			38	-97
108.12.244.232	US	RI	Coventry	41.6933	-71.6395
108.129.55.203	US	GA	Albany	31.523	-84.3024
108.45.180.228	US	VA	Reston	38.9311	-77.3489
109.167.207.7	RU	66	Saint Petersburg	59.8944	30.2642
109.75.64.202	LB	2	Tyre	33.2711	35.1964
116.30.99.175	CN	30	Guangzhou	23.1167	113.25
119.160.118.20	PK			30	70
12.125.129.58	US			38	-97
120.60.152.245	IN	16	Mumbai	18.975	72.8258
121.111.140.254	JP			35.69	139.69
124.6.181.42	PH	63	Tarlac	15.4802	120.5979
125.60.156.217	PH			13	122
128.211.249.130	US	IN	West Lafayette	40.4249	-86.9162
131.253.40.47	US			38	-97
131.253.40.50	US			38	-97
134.170.19.133	US			38	-97
134.170.24.237	US			38	-97
134.255.159.146	RU	66	Zarechny	52.1137	38.0953
135.23.120.199	CA			60	-95
135.23.226.25	CA	ON	Etobicoke	43.7	-79.5667
137.116.32.77	US			38	-97
139.193.132.95	ID	4	Jakarta	-6.1744	106.8294

IP Address	Country	Region	City	Latitude	Longitude
141.105.136.1	UA			49	32
141.225.188.27	US	TN	Memphis	35.2017	-89.9715
151.213.191.20	US	TX	Sugar Land	29.5557	-95.6335
153.139.48.194	JP	40	Tokyo	35.685	139.7514
153.167.115.57	JP	40	Tokyo	35.685	139.7514
154.0.75.100	AO			-12.5	18.5
154.52.1.183	US			38	-97
157.55.130.160	US	WA	Redmond	47.6801	-122.1206
157.55.130.166	US	WA	Redmond	47.6801	-122.1206
157.56.108.82	US	WA	Redmond	47.6801	-122.1206
157.56.109.8	US	WA	Redmond	47.6801	-122.1206
166.147.108.114	US			38	-97
168.61.145.190	US			38	-97
168.61.160.96	US			38	-97
172.56.19.215	US	NY	Bronx	40.8472	-73.8983
173.72.29.123	US	NJ	Glassboro	39.6969	-75.125
173.77.151.67	US	NY	Massapequa	40.6826	-73.467
174.236.102.18	US	NY	Brooklyn	40.6501	-73.9496
176.51.224.91	RU			60	100
177.194.76.185	BR			-10	-55
178.126.236.189	BY	7	Vitebsk	55.1904	30.2049
178.135.143.12	LB	4	Beirut	33.8719	35.5097
178.42.23.202	PL	73	Polska	52.5931	19.0894
179.129.27.15	BR			-10	-55
181.37.137.127	DO	5	Santo Domingo	18.4667	-69.9
182.185.104.228	PK	3	Peshawar	34.008	71.5785
182.56.161.164	IN			20	77
186.151.16.164	GT	16	Antigua	14.5611	-90.7344
186.222.145.250	BR			-10	-55
186.48.124.28	UY	10	Montevideo	-34.8581	-56.1708
186.77.204.66	NI	10	Managua	12.1508	-86.2683
187.13.219.110	BR			-10	-55
187.34.181.224	BR			-10	-55
188.247.77.142	JO	2	Amman	31.95	35.9333
189.203.102.106	MX	9	Mexico	19.4342	-99.1386
190.167.78.130	DO	25	Santiago	19.45	-70.7
190.249.15.182	CO	2	Medellín	6.2518	-75.5636
191.77.253.228	CO			4	-72
198.228.233.69	US			38	-97
198.84.206.216	CA			60	-95

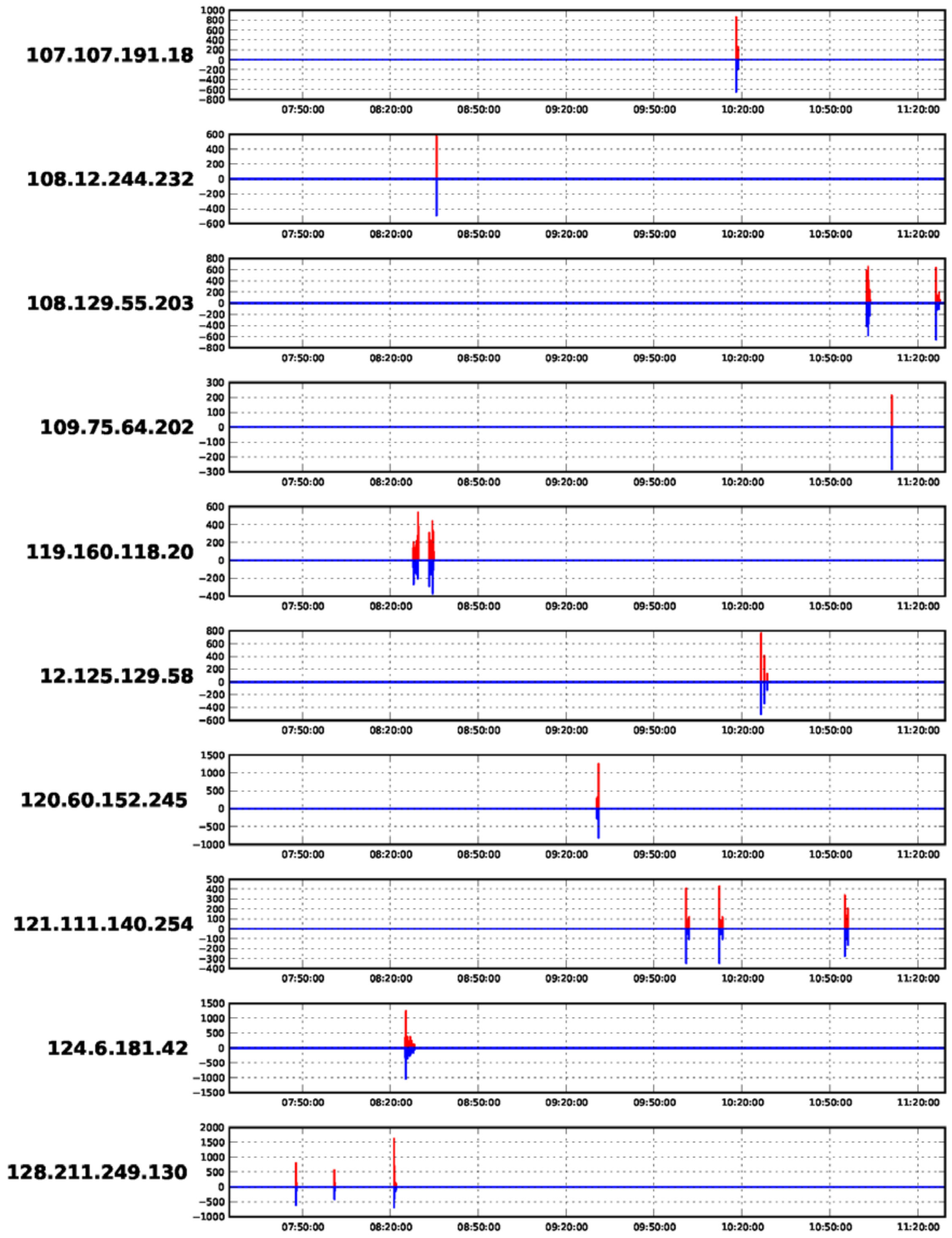
IP Address	Country	Region	City	Latitude	Longitude
199.255.211.24	A1			0	0
2.84.188.66	GR	35	Athens	37.9833	23.7333
204.16.68.124	US	CA	Sunnyvale	37.3887	-122.0188
204.79.197.200	US			38	-97
206.191.134.233	US	GA	Atlanta	33.7516	-84.3915
206.214.31.159	VG	0	Road Town	18.4167	-64.6167
207.244.82.129	US	VA	Manassas	38.7932	-77.5366
207.46.11.151	US	WA	Redmond	47.6801	-122.1206
207.46.206.13	US	WA	Redmond	47.6801	-122.1206
207.46.206.81	US	WA	Redmond	47.6801	-122.1206
208.88.255.210	US	IN	Mccordsville	39.8938	-85.8881
213.166.51.4	LU			49.75	6.1667
213.87.133.99	RU			60	100
216.165.156.27	US	WI	Madison	43.0737	-89.5274
223.190.199.222	IN	19	Bangalore	12.9833	77.5833
223.205.226.254	TH	2	Chiang Mai	18.7904	98.9847
223.217.55.139	JP	40	Tokyo	35.685	139.7514
223.218.6.137	JP	40	Tokyo	35.685	139.7514
23.28.10.246	US	MI	Haslett	42.773	-84.3745
23.63.157.195	US	MA	Cambridge	42.3626	-71.0843
23.63.158.161	US	MA	Cambridge	42.3626	-71.0843
23.63.159.240	US	MA	Cambridge	42.3626	-71.0843
23.63.162.110	US	MA	Cambridge	42.3626	-71.0843
23.63.175.139	US	MA	Cambridge	42.3626	-71.0843
24.102.144.202	US	NJ	Phillipsburg	40.6911	-75.1756
24.155.243.221	US	TX	Austin	30.2672	-97.7431
24.163.35.128	US	NC	Raleigh	35.8874	-78.751
24.181.81.118	US	AL	Decatur	34.5389	-86.8733
24.228.80.100	US	NY	Brooklyn	40.6714	-73.9361
24.233.189.137	US	FL	Pompano Beach	26.2436	-80.2656
27.147.173.189	BD	81	Dhaka	23.7231	90.4086
27.53.194.9	TW	3	Taipei	25.0392	121.525
27.99.27.51	AU	2	Blacktown	-33.7667	150.9167
31.13.74.33	IE			53	-8
31.167.110.78	SA			25	45
31.193.216.71	IE	27	Waterford	52.2583	-7.1119
37.124.186.201	SA			25	45
39.7.24.209	KR	11	Seoul	37.5985	126.9783
41.105.3.38	DZ			28	3
41.82.95.75	SN	1	Dakar	14.6708	-17.4381

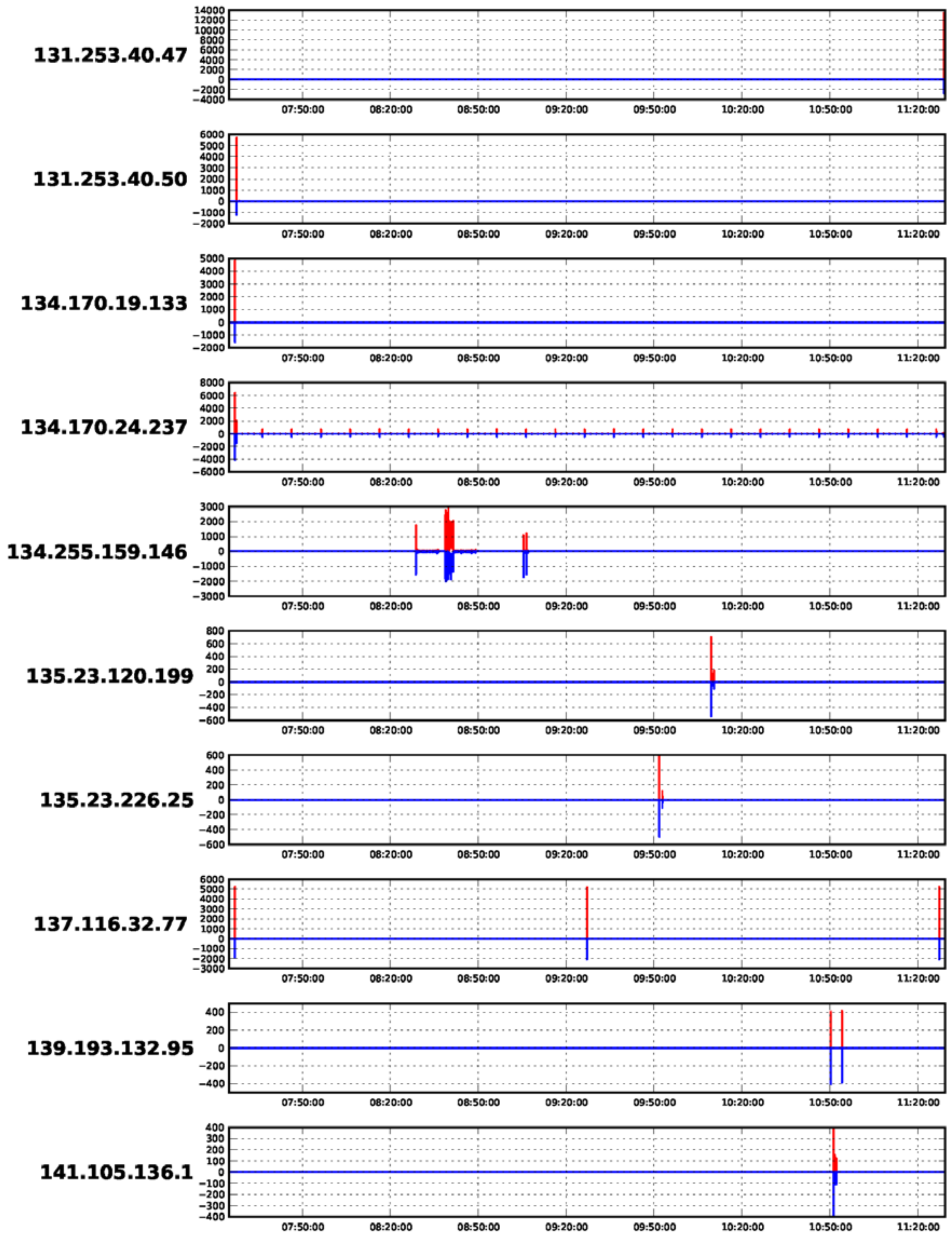
IP Address	Country	Region	City	Latitude	Longitude
46.162.213.94	AM			40	45
46.211.122.243	UA			49	32
46.71.19.199	AM			40	45
49.98.51.49	JP	32	Osaka	34.6864	135.52
64.223.214.39	US	ME	Ellsworth	44.6513	-68.4507
64.30.84.178	US	MA	Shrewsbury	42.2865	-71.7147
65.52.108.3	US	WA	Redmond	47.6801	-122.1206
65.55.5.233	US	WA	Redmond	47.6801	-122.1206
65.78.104.44	US	PA	Allentown	40.5678	-75.4828
65.8.135.19	US	FL	Miami	25.7743	-80.1937
66.98.36.69	DO	5	Santo Domingo	18.4667	-69.9
67.187.103.149	US	TN	Knoxville	35.9217	-84.1829
67.41.242.193	US	IA	West Des Moines	41.572	-93.7453
68.15.189.254	US	AZ	Scottsdale	33.2765	-112.1872
68.186.125.231	US	OR	Grants Pass	42.5386	-123.3478
68.9.189.126	US	RI	Bristol	41.6783	-71.2708
69.119.189.76	US	NJ	Howell	40.1524	-74.1978
69.124.19.13	US	NJ	Piscataway	40.5516	-74.4637
69.165.192.100	CA	QC	Montréal	45.5	-73.5833
69.171.166.128	US	TX	Houston	29.7633	-95.3633
69.244.27.141	US	AL	Tuscaloosa	33.3323	-87.4741
70.102.165.150	US	ID	Nampa	43.6008	-116.6103
70.192.141.37	US	PA	Tamaqua	40.7818	-75.987
70.52.24.204	CA			60	-95
71.1.203.209	US	FL	Ocoee	28.5704	-81.5298
71.239.157.24	US	IN	Schererville	41.4881	-87.4424
71.46.57.69	US	FL	Orlando	28.5383	-81.3792
71.87.50.214	US	WI	Baraboo	43.4807	-89.7399
72.177.111.214	US	TX	Bastrop	30.1406	-97.3275
72.184.244.92	US	FL	Largo	27.9166	-82.8032
74.215.72.18	US	OH	Cleves	39.2039	-84.7225
75.114.173.50	US	MI	Farmington	42.4629	-83.3499
75.115.36.115	US	FL	Tampa	27.9987	-82.5156
75.162.62.249	US	IA	Urbandale	41.6267	-93.7122
75.182.111.214	US	NC	Raleigh	35.7463	-78.7239
75.69.136.181	US	MA	Gloucester	42.6179	-70.7154
75.92.71.116	US	CA	Los Angeles	34.0478	-118.2923
76.253.161.192	US	TX	Mansfield	32.5702	-97.1313
76.7.178.60	US	TN	Johnson City	36.3134	-82.3535
77.246.55.139	ZW	4	Harare	-17.8178	31.0447

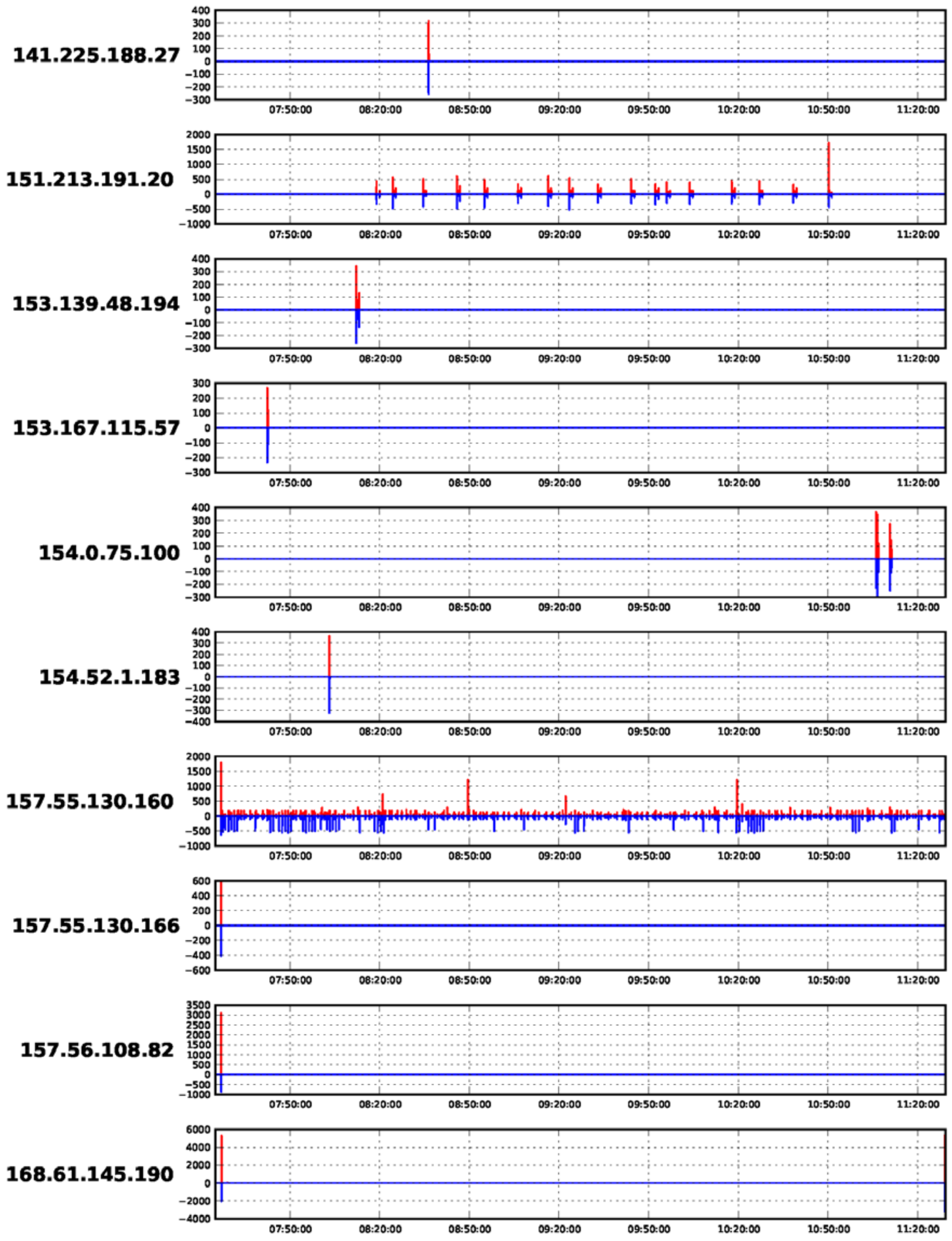
IP Address	Country	Region	City	Latitude	Longitude
80.31.56.165	ES	60	Valencia	39.4667	-0.3667
83.34.171.90	ES	51	Coria Del Río	37.2877	-6.0541
85.73.100.91	GR			39	22
85.76.145.26	FI	13	Vantaa	60.3	24.85
86.73.49.220	FR	A8	Paris	48.8574	2.3795
88.185.135.243	FR	B6	Le Plessis	48.7833	2.2667
91.124.15.151	UA			49	32
91.146.63.155	RU	80	Izhevsk	56.8498	53.2045
91.190.218.17	LU			49.75	6.1667
91.190.218.64	LU			49.75	6.1667
91.75.71.190	AE			24	54
92.53.12.119	MK	41	Skopje	42	21.4333
93.184.215.200	US	DC	Washington	38.8951	-77.0364
93.184.215.201	US	DC	Washington	38.8951	-77.0364
93.219.172.197	DE			51	9
93.73.55.58	UA	12	Kiev	50.4333	30.5167
95.107.228.83	AL			41	20
96.20.243.34	CA	QC	Montréal	45.5	-73.5833
96.58.221.194	US	FL	Seminole	27.8512	-82.7545
98.112.222.104	US	CA	Panorama City	34.2287	-118.4441
99.175.77.82	US			38	-97
99.183.209.167	US	MO	Saint Louis	38.6446	-90.2533
99.199.224.163	CA			60	-95
99.39.114.88	US	TX	Sandy	30.2198	-98.3586
99.88.249.179	US	MO	Saint Louis	38.5491	-90.3199

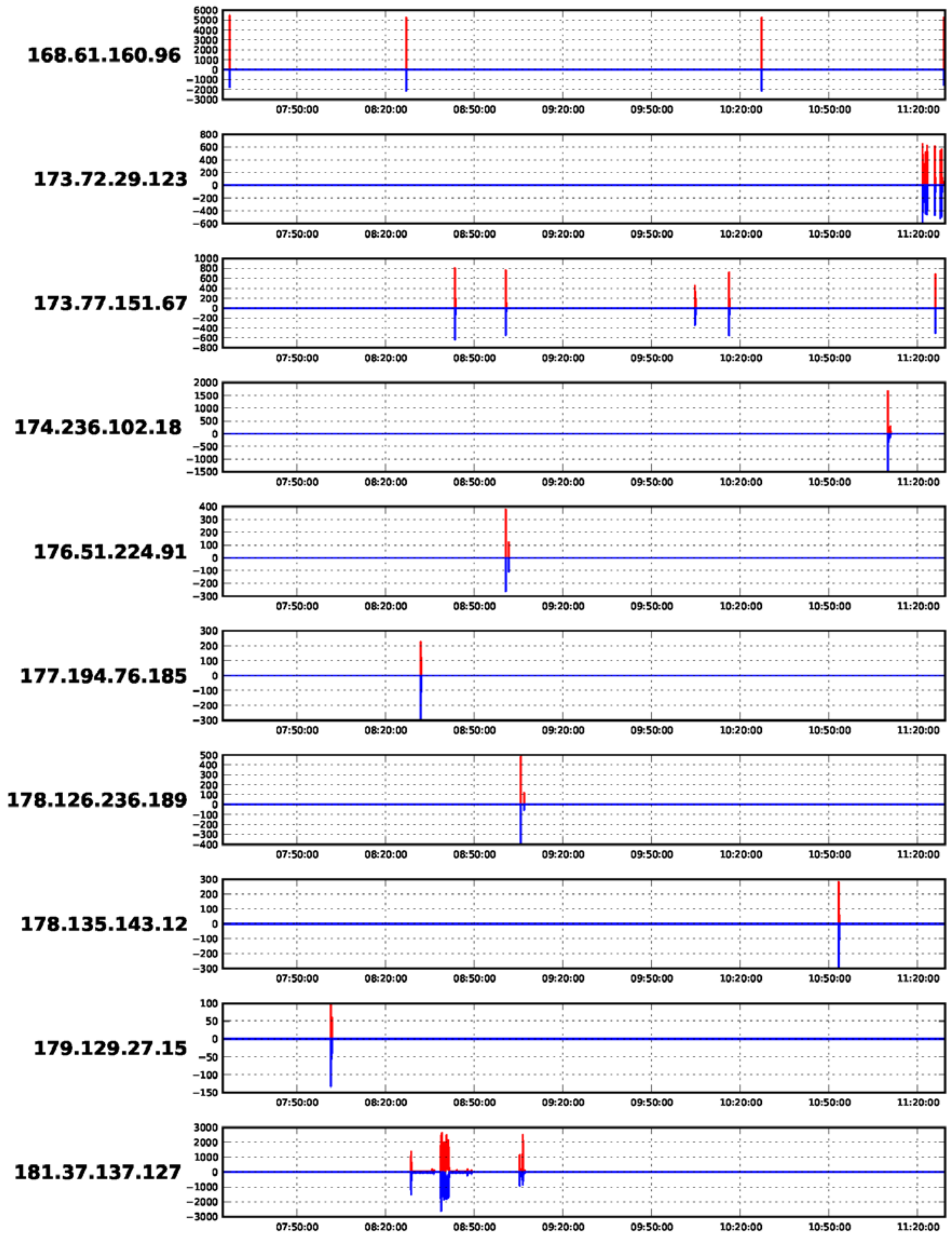
8.3. Appendix 3 - Traffic Analysis Using PlotIOStats.py

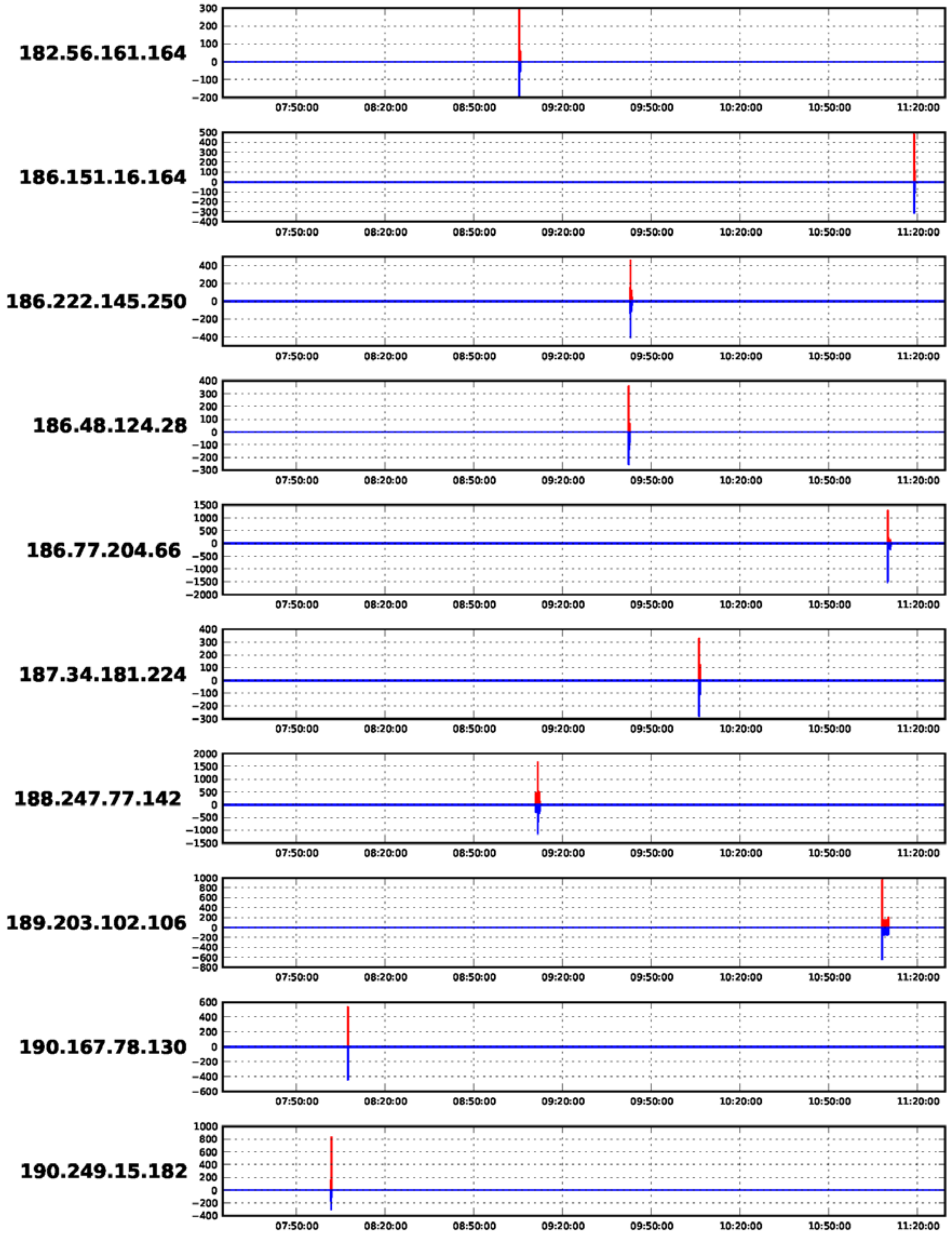
The `PlotIOStats.py` program generated one plot for each remote IP address that Skype communicated with during the four-hour experiment discussed in Section 4.3. The following pages contain the complete output.

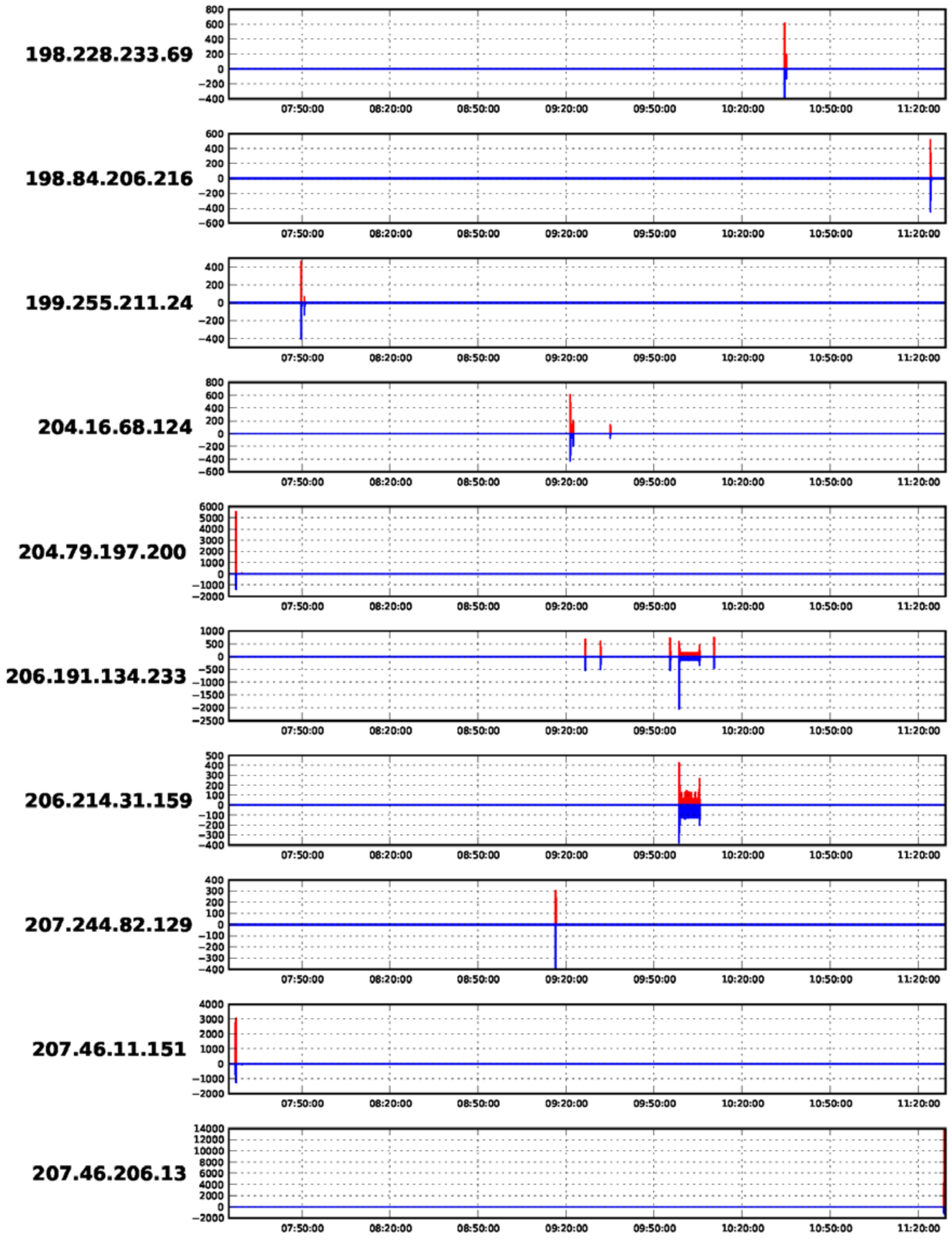

107.107.191.18
108.12.244.232
108.129.55.203
109.75.64.202
119.160.118.20
12.125.129.58
120.60.152.245
121.111.140.254
124.6.181.42
128.211.249.130
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

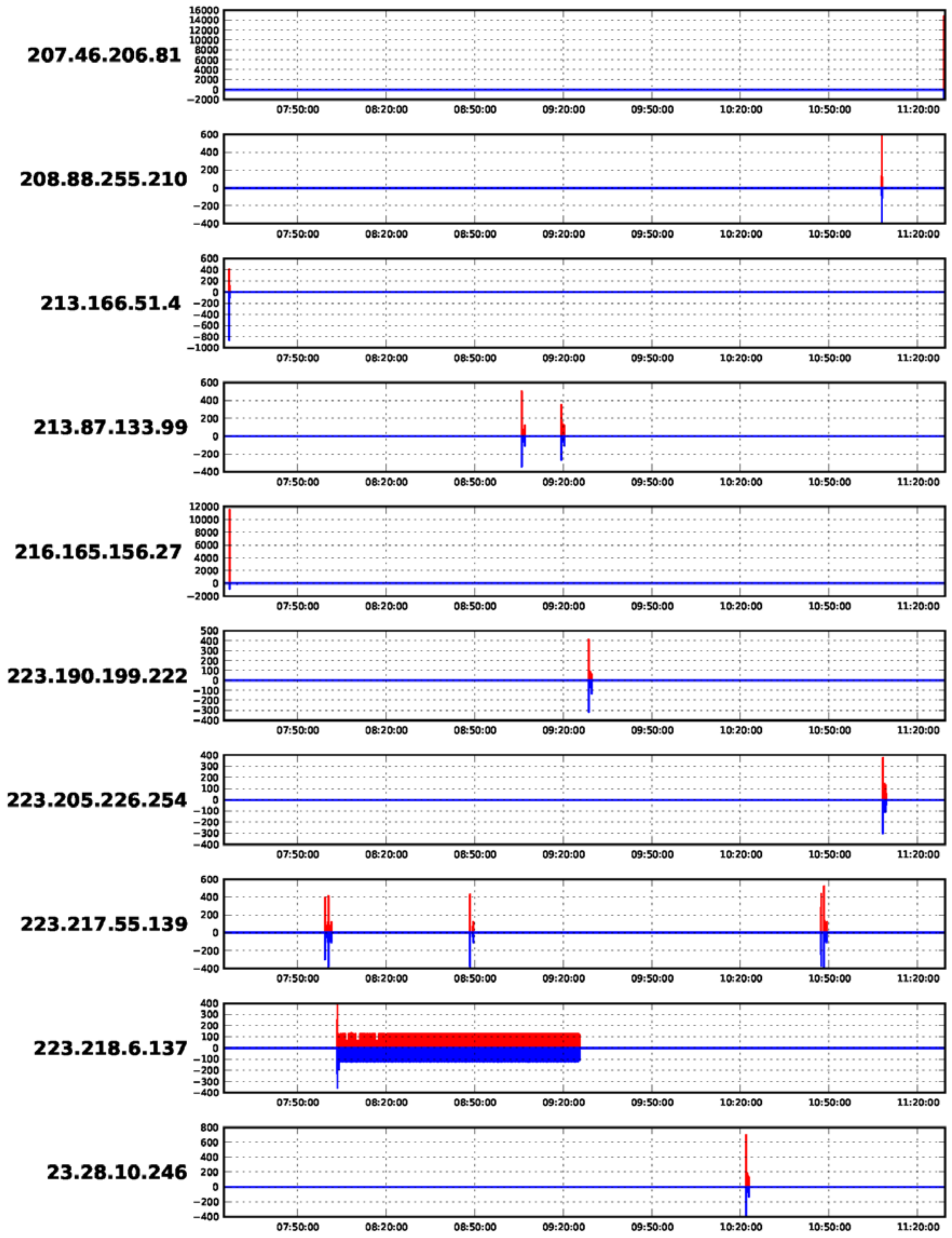

131.253.40.47
131.253.40.50
134.170.19.133
134.170.24.237
134.255.159.146
135.23.120.199
135.23.226.25
137.116.32.77
139.193.132.95
141.105.136.1
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

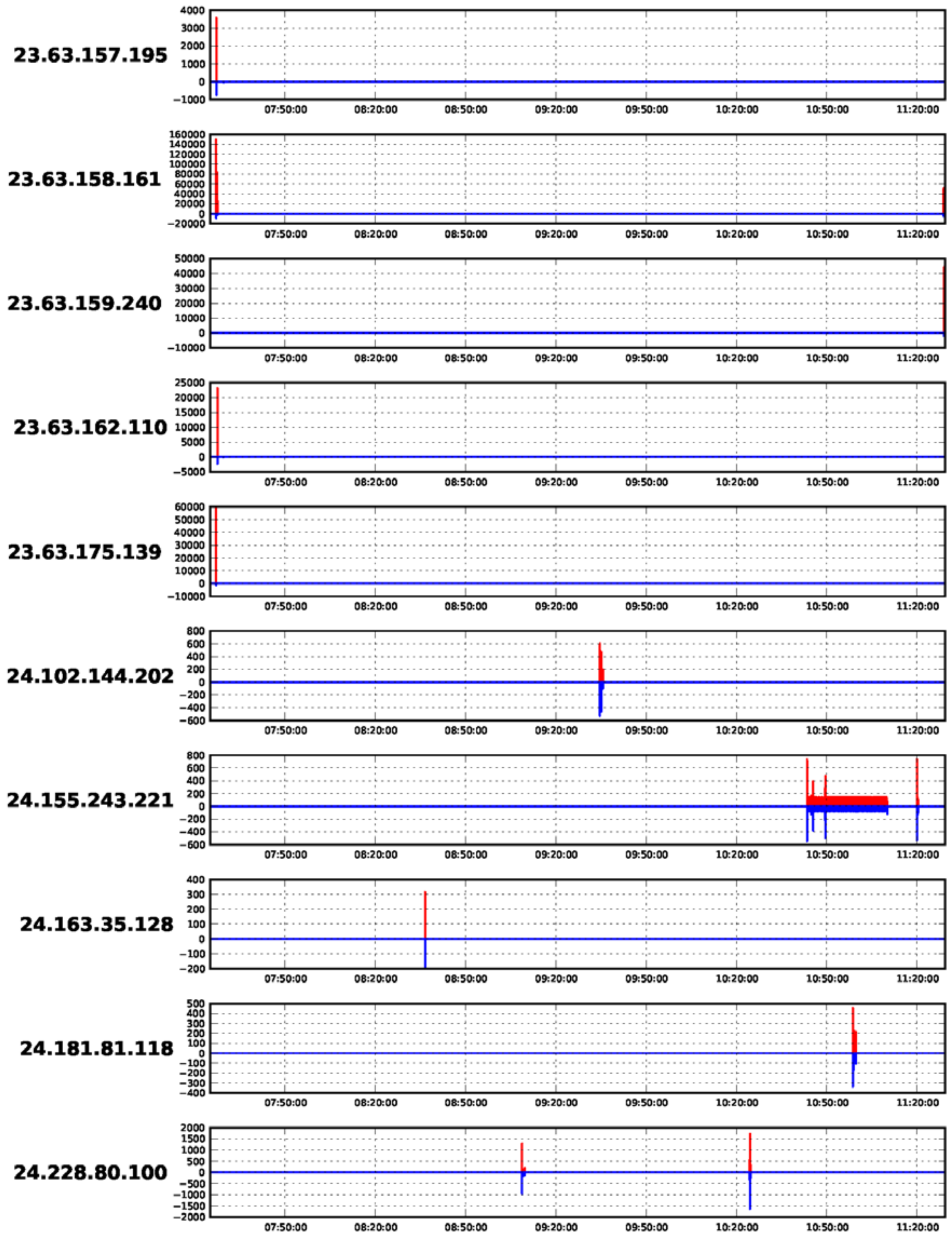

141.225.188.27
151.213.191.20
153.139.48.194
153.167.115.57
154.0.75.100
154.52.1.183
157.55.130.160
157.55.130.166
157.56.108.82
168.61.145.190
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

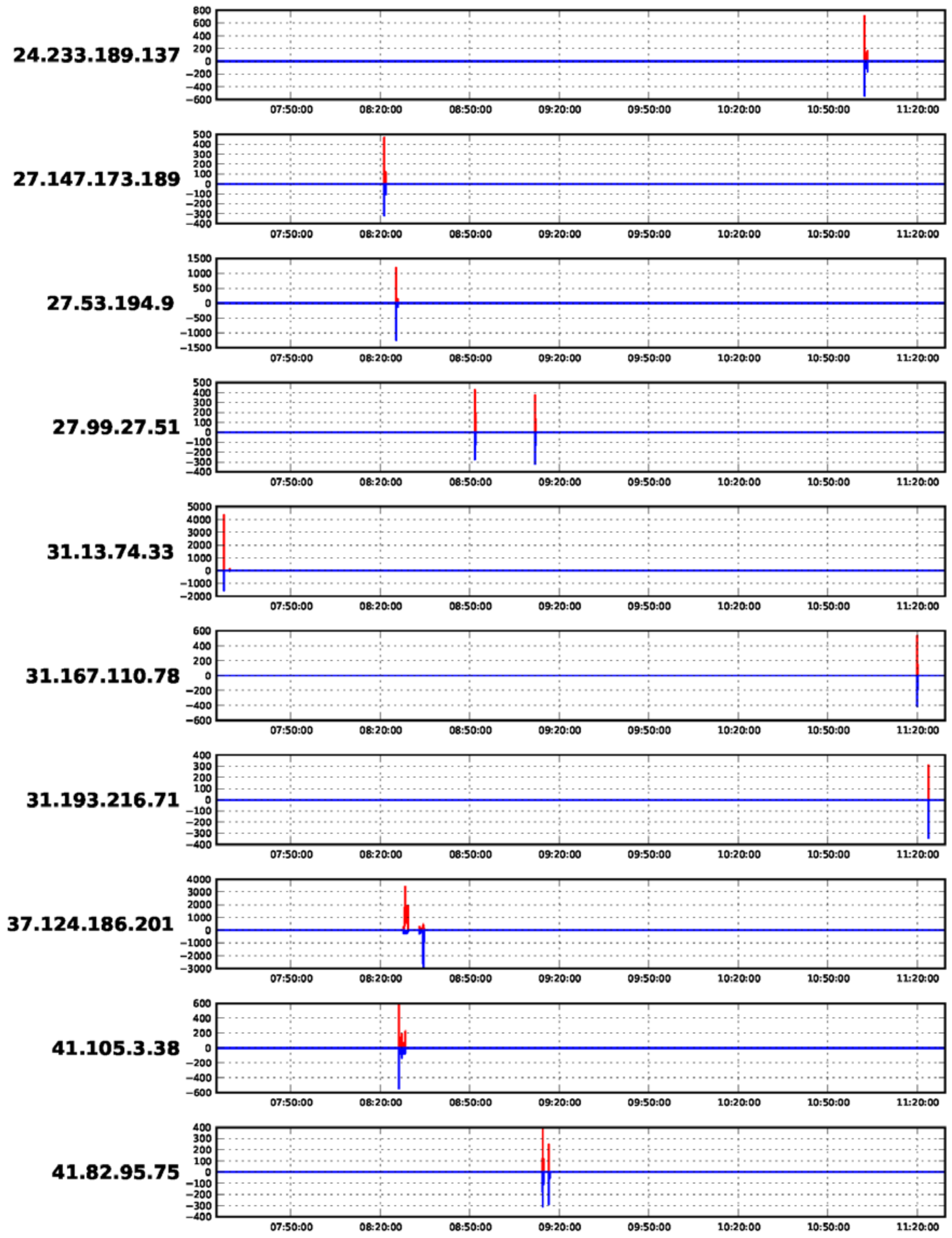

168.61.160.96
173.72.29.123
173.77.151.67
174.236.102.18
176.51.224.91
177.194.76.185
178.126.236.189
178.135.143.12
179.129.27.15
181.37.137.127
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

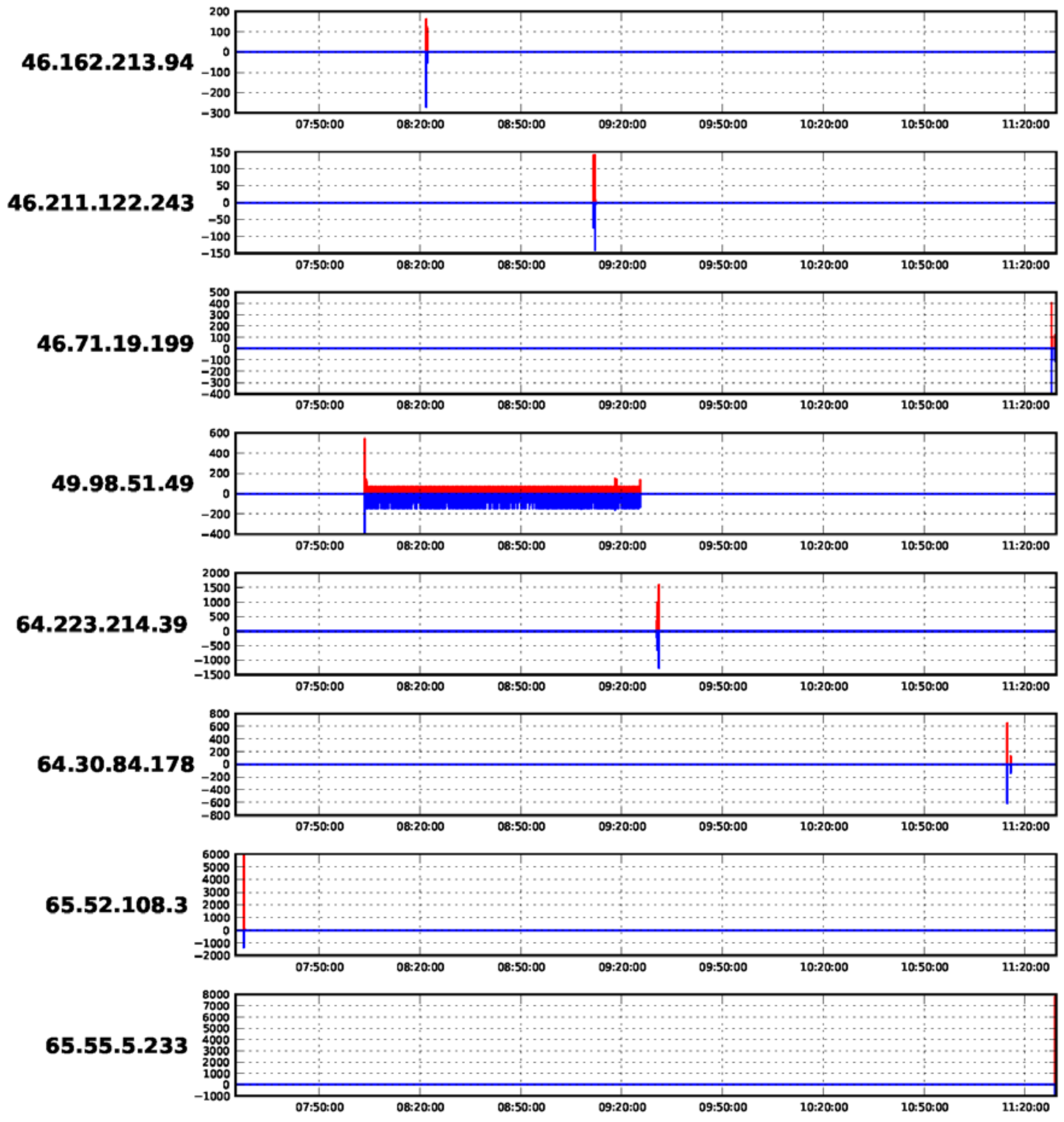

182.56.161.164
186.151.16.164
186.222.145.250
186.48.124.28
186.77.204.66
187.34.181.224
188.247.77.142
189.203.102.106
190.167.78.130
190.249.15.182
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

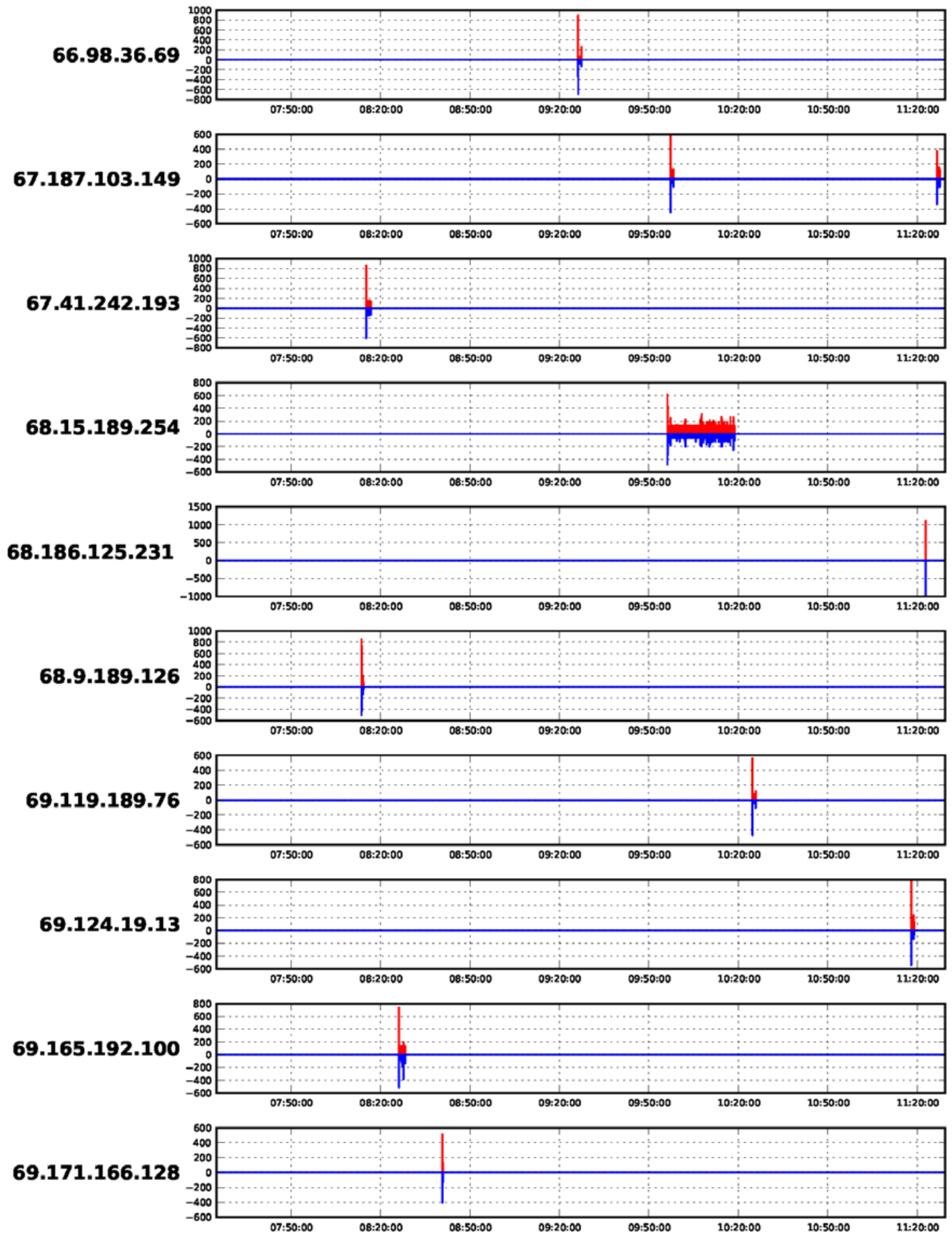

198.228.233.69
198.84.206.216
199.255.211.24
204.16.68.124
204.79.197.200
206.191.134.233
206.214.31.159
207.244.82.129
207.46.11.151
207.46.206.13
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

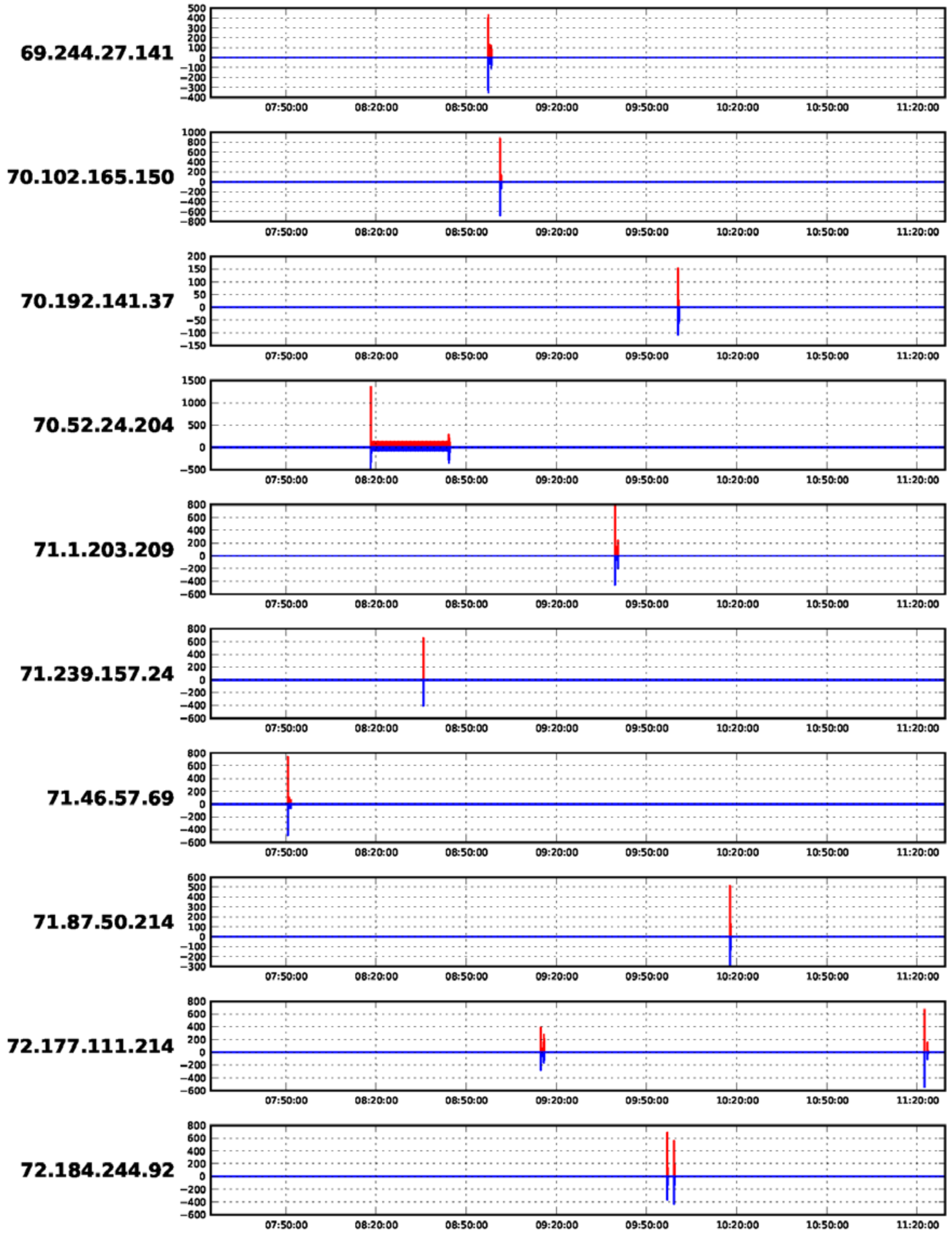

207.46.206.81
208.88.255.210
213.166.51.4
213.87.133.99
216.165.156.27
223.190.199.222
223.205.226.254
223.217.55.139
223.218.6.137
23.28.10.246
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

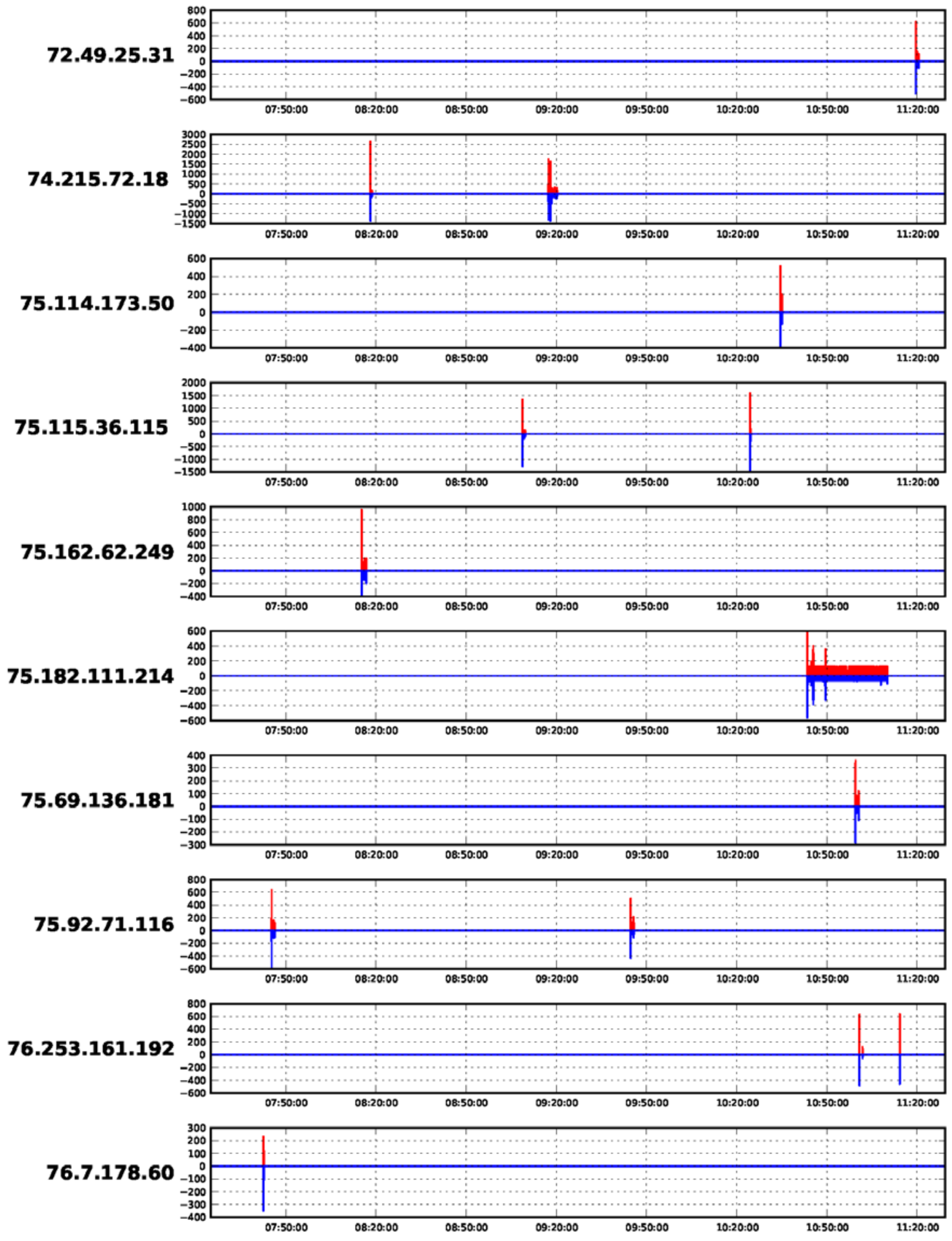

23.63.157.195
23.63.158.161
23.63.159.240
23.63.162.110
23.63.175.139
24.102.144.202
24.155.243.221
24.163.35.128
24.181.81.118
24.228.80.100
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

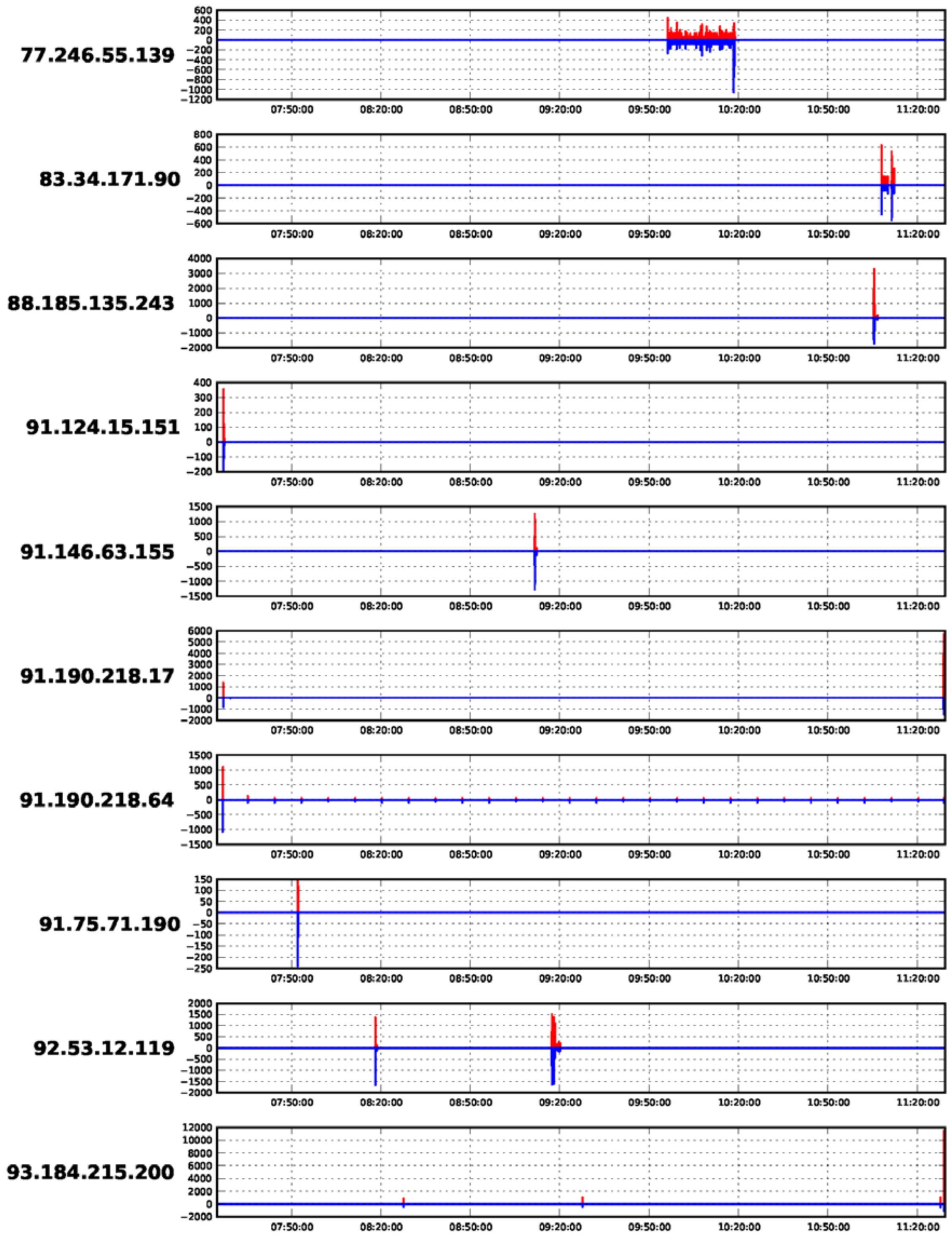

24.233.189.137
27.147.173.189
27.53.194.9
27.99.27.51
31.13.74.33
31.167.110.78
31.193.216.71
37.124.186.201
41.105.3.38
41.82.95.75
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00

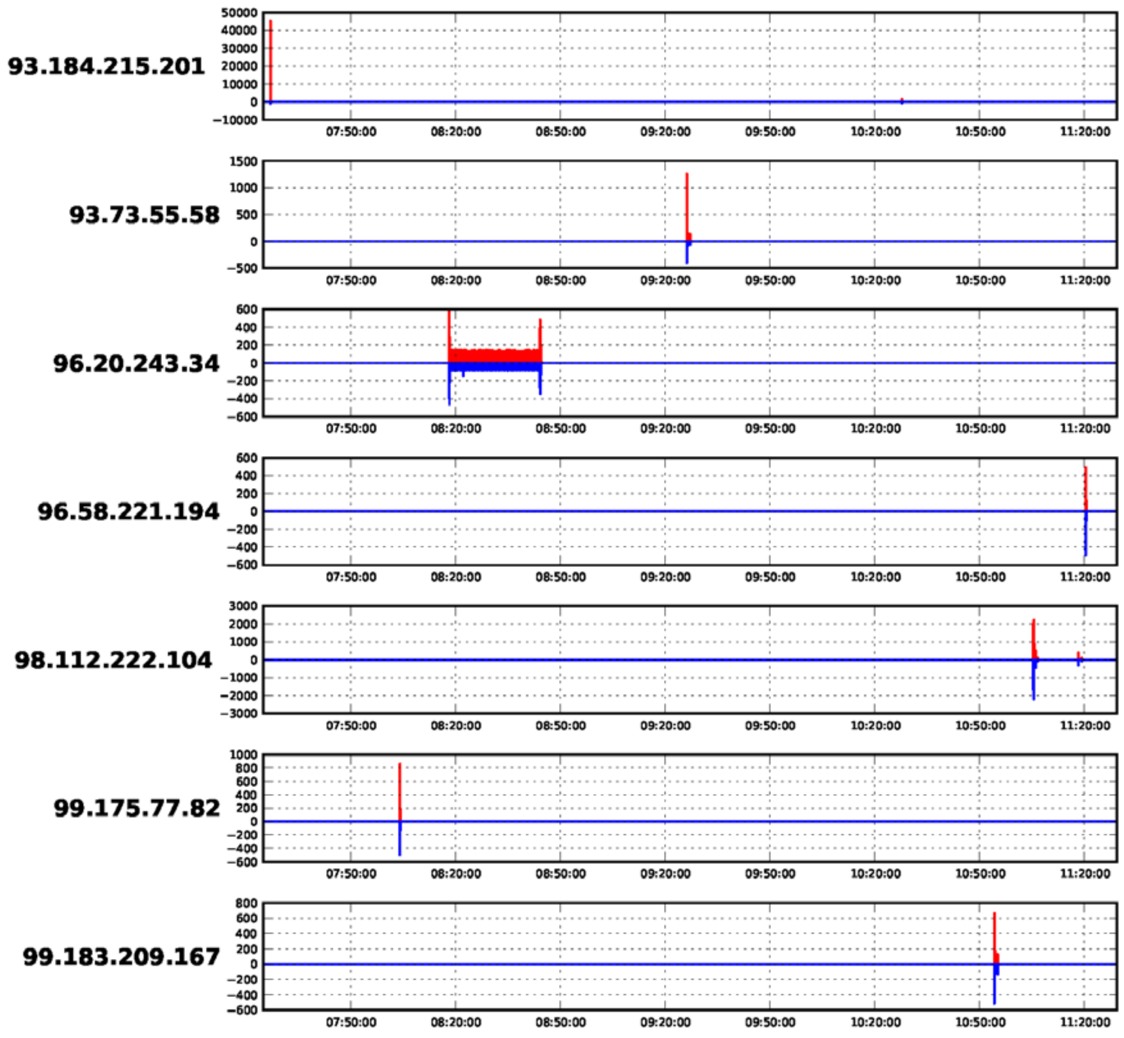

46.162.213.94
46.211.122.243
46.71.19.199
49.98.51.49
64.223.214.39
64.30.84.178
65.52.108.3
65.55.5.233
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00


66.98.36.69
67.187.103.149
67.41.242.193
68.15.189.254
68.186.125.231
68.9.189.126
69.119.189.76
69.124.19.13
69.165.192.100
69.171.166.128
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00


69.244.27.141
70.102.165.150
70.192.141.37
70.52.24.204
71.1.203.209
71.239.157.24
71.46.57.69
71.87.50.214
72.177.111.214
72.184.244.92
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00


72.49.25.31
74.215.72.18
75.114.173.50
75.115.36.115
75.162.62.249
75.182.111.214
75.69.136.181
75.92.71.116
76.253.161.192
76.7.178.60
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00


77.246.55.139
83.34.171.90
88.185.135.243
91.124.15.151
91.146.63.155
91.190.218.17
91.190.218.64
91.75.71.190
92.53.12.119
93.184.215.200
07:50:00
08:20:00
08:50:00
09:20:00
09:50:00
10:20:00
10:50:00
11:20:00



8.4. Appendix 4 - Calculate-TCPflow-Entropy.py

The Calculate-TCPflow-Entropy.py program will compute the Shannon Entropy of every file generated by tcpflow and will create a histogram to visualize the entropy. The tcpflow program is a well-known tool that outputs a file for the stream of data between each socket pair in a packet capture file.

The source code, supporting files and complete documentation is available at http://www.kennethghartman.com/projects/Calculate-TCPflow-Entropy.html

8.4.1. Calculate-TCPflow-Entropy.py Source Code

```
# graph_TCPFlow-file_entropy.py
#
# Shannon Entropy of a file
# = minimum average number of bits per character
# required for encoding (compressing) the file
#
# So the theoretical limit (in bytes) for data compression:
# Shannon Entropy of the file * file size (in bytes) / 8
# -------------
# Part of this function is based on
# http://code.activestate.com/recipes
#    /577476-shannon-entropy-calculation/#c3
# For more information and documentation,
# visit http://www.kennethghartman.com/calculate-file-entropy/


import xml.dom.minidom
import os
import subprocess
import sys
import math

def graph_TCPFlow_file_entropy(TCPflow_filename, outputfile):
    import os

    ## read the whole file into a byte array
    f = open(TCPflow_filename, "rb")
    byteArr = map(ord, f.read())
    f.close()
    fileSize = len(byteArr)

    ## rename the file with a TXT extension
    os.rename(TCPflow_filename, TCPflow_filename + '.txt')

    ## calculate the frequency of each byte value in the file
    freqList = []
    for b in range(256):
        ctr = 0
        for byte in byteArr:
            if byte == b:
                ctr += 1
        freqList.append(float(ctr) / fileSize)

    ent = 0.0
    for freq in freqList:
        if freq > 0:
            ent = ent + freq * math.log(freq, 2)
    ent = -ent

    ###  Modifications to file_entropy.py to create the Histogram start here ###
    ###  by Ken Hartman  www.KennethGHartman.com                             ###

    import os
    import numpy as np
    import matplotlib.pyplot as plt

    if ent < 7.0:
        highlight = 'bgcolor = "yellow"'
    else:
        highlight = 'bgcolor = "#C8C8C8"'

    f = open(outputfile, 'a')
    output = '<tr ' + highlight + '><td><a href="' + TCPflow_filename[7:]
    output = output + '.txt">' + TCPflow_filename[7:] + '</a></td>'
    output = output + '<td  align="right">' + str(fileSize) + '</td><td>'
    output = output + '<a href="' + TCPflow_filename[7:] + '.pdf">' + str(ent)
    putput = output + '</a></td></tr>\r\n'
    f.write(output)
    f.close()

    N = len(freqList)

    ind = np.arange(N)  # the x locations for the groups
    width = 1.00        # the width of the bars

    fig = plt.figure(figsize=(11,5),dpi=100)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, freqList, width)
    ax.set_autoscalex_on(False)
    ax.set_xlim([0,255])

    ax.set_ylabel('Frequency')
    ax.set_xlabel('Byte')
    ax.set_title('Frequency of Bytes 0 to 255\nFILENAME: ' + TCPflow_filename[7:])

    # plt.show()
    plt.savefig(TCPflow_filename + '.pdf',bbox_inches='tight',pad_inches=0.5)

#######################
##    MAIN PROGRAM   ##
#######################

print "Call the external TCPFlow program"
subprocess.call(['Call-TCPFlow.bat'])

print "Set up the HTML output file"
entropylist = 'output\\EntropyTable.html'
htmlheader1 = '<html><head><title>Entropy of TCPFlow Output Files</title>'
htmlheader2 = '<style>table,tr{border:1px solid grey;border-collapse:collapse;}'
htmlheader3 = 'table{border-spacing:15px;}td{padding: 10px;}</style></head><body>'
htmltopbody = '<h1 align="center">Entropy of TCPFlow Output Files</h1>'
htmltopbody = htmltopbody + '<table align="center">'
htmltableheader = '<tr><th>TCPFlow Output File</th><th>File Size</th>'
htmltableheader = htmltableheader + '<th>Entropy</th></tr>'
htmlfooter = '</table></body></html>'

f = open(entropylist, 'a')
output = htmlheader1 + '\r\n' + htmlheader2 + '\r\n' + htmlheader3 + '\r\n' +
htmltopbody + '\r\n' + htmltableheader + '\r\n'
f.write(output)
f.close()

print "Get list of filenames in report.xml"
dom = xml.dom.minidom.parse('output\\report.xml')
filelist = dom.getElementsByTagName('filename')
size = dom.getElementsByTagName('filesize')

print "Generate the output by processing each TCPFlow output file"
```

```
for x in range(0, len(filelist)):
    filename = dom.getElementsByTagName('filename')[x]
    filesizeitem = dom.getElementsByTagName('filesize')[x]
    filesize = int(filesizeitem.firstChild.wholeText)
    if filesize > 0:
        print "   Processing " + filename.firstChild.wholeText
        graph_TCPFlow_file_entropy(filename.firstChild.wholeText, entropylist)

print "Close up the HTML output file"
f = open(entropylist, 'a')
output = htmlfooter + '\r\n'
f.write(output)
f.close()
print "Processing is complete."
```

8.4.2. Call-TCPFlow.BAT Source Code

```
@echo off
@Echo "Purging the output directory
del output\*.* /Q
@echo.
@echo.
@echo.
Echo "Please wait for TCPFlow to process the pcap file..."
@echo.
tcpflow64.exe -r capture.pcap -o output
@echo.
@echo.
@echo.
Echo "Done"
```

8.5. Appendix 5 – Log-Connections.ps1

The Log-Connections.ps1 file is a PowerShell Script that Logs active TCP connections and includes the process ID (PID) and process name for each connection on a Microsoft Windows computer. The log file name is a parameter that is passed to the script at run time. A log entry is created every time that the list of processes with open connections or listening ports changes. The source code, example output files and complete documentation is available at http://www.kennethghartman.com/projects/Log-Connections.html

8.5.1. Log-Connections.ps1 Source Code

```
#* FileName: Log-Connections.ps1
 #*=============================================================================
 #* Script Name: Log-Connections
 #* Created:     [12/1/2012]
 #* Author:      Kenneth G. Hartman
 #* Email:       KGH@kennethGHartman.com
 #* Web:         http://www.KennethGHartman.com
 #*
 #* CREDITS:    Author of the Get-NetworkStatistics Function is
 #*             Shay Levy   http://PowerShay.com  http://poshcode.org/2701
 #*        http://blogs.microsoft.co.il/blogs/scriptfanatic/archive/2011/02/10/
 #*        How-to-find-running-processes-and-their-port-number.aspx
 #*============================================================================

#*============================================================================
 #* REVISION HISTORY
 #*============================================================================
 #* Date: [12/16/2012]
 #* Description: Initial Version
 #*
 #*============================================================================

[CmdletBinding()]
Param(
   [Parameter(Mandatory=$True,Position=0)]
   [string]$FilePath,

   [Parameter(Mandatory=$False,Position=1)]
   [string]$ProcName='*',

   [switch]$PassThru
)

#*******************************************
#* Get-NetworkStatistics Function          *
#*******************************************

function Get-NetworkStatistics
{
        [OutputType('System.Management.Automation.PSObject')]
        [CmdletBinding(DefaultParameterSetName='name')]

        param(
                [Parameter(Position=0,ValueFromPipeline=$true,ParameterSetName='port')]
                [System.Int32]$Port='*',

                [Parameter(Position=0,ValueFromPipeline=$true,ParameterSetName='name')]
                [System.String]$ProcessName='*',

                [Parameter(Position=0,ValueFromPipeline=$true,ParameterSetName='address')]
                [System.String]$Address='*',

                [Parameter()]
                [ValidateSet('*','tcp','udp')]
                [System.String]$Protocol='*',

                [Parameter()]

        [ValidateSet('*','Closed','CloseWait','Closing','DeleteTcb','Established','FinWait1',`
                'FinWait2','LastAck','Listen','SynReceived','SynSent','TimeWait','Unknown')]
                [System.String]$State='*'

        )

        begin
        {
                $properties = 'Protocol','LocalAddress','LocalPort'
                $properties += 'RemoteAddress','RemotePort','State','ProcessName','PID'
        }

        process
        {
            netstat -ano | Select-String -Pattern '\s+(TCP|UDP)' | ForEach-Object {

                $item = $_.line.split(' ',[System.StringSplitOptions]::RemoveEmptyEntries)

                if($item[1] -notmatch '^\[::')
                {
                    if (($la = $item[1] -as [ipaddress]).AddressFamily -eq 'InterNetworkV6')
                    {
                       $localAddress = $la.IPAddressToString
                       $localPort = $item[1].split('\]:')[-1]
                    }
                    else
                    {
                        $localAddress = $item[1].split(':')[0]
                        $localPort = $item[1].split(':')[-1]
                    }

                    if (($ra = $item[2] -as [ipaddress]).AddressFamily -eq 'InterNetworkV6')
                    {
                       $remoteAddress = $ra.IPAddressToString
                       $remotePort = $item[2].split('\]:')[-1]
                    }
                    else
                    {
                       $remoteAddress = $item[2].split(':')[0]
                       $remotePort = $item[2].split(':')[-1]
                    }

                                $procId = $item[-1]
                                $ProcName = (Get-Process -Id $item[-1] -ErrorAction
SilentlyContinue).Name
                                $proto = $item[0]
                                $status = if($item[0] -eq 'tcp') {$item[3]} else {$null}


                                $pso = New-Object -TypeName PSObject -Property @{
                                        PID = $procId
                                        ProcessName = $ProcName
                                        Protocol = $proto
                                        LocalAddress = $localAddress
                                        LocalPort = $localPort
                                        RemoteAddress =$remoteAddress
                                        RemotePort = $remotePort
                                        State = $status
                                } | Select-Object -Property $properties



                                if($PSCmdlet.ParameterSetName -eq 'port')
                                {
                                        if($pso.RemotePort -like $Port -or $pso.LocalPort -like
$Port)
                                        {
                                            if($pso.Protocol -like $Protocol -and $pso.State -
like $State)
                                                {
                                                        $pso
                                                }
                                        }
                                }

                                if($PSCmdlet.ParameterSetName -eq 'address')
                                {
                                        if($pso.RemoteAddress -like $Address -or
$pso.LocalAddress -like $Address)
                                        {
                                            if($pso.Protocol -like $Protocol -and $pso.State -
like $State)
                                                {
                                                        $pso
                                                }
                                        }
                                }
                                if($PSCmdlet.ParameterSetName -eq 'name')
                                {
                                        if($pso.ProcessName -like $ProcessName)
                                        {
                                                if($pso.Protocol -like $Protocol -and $pso.State
-like $State)
                                                {
                                                        $pso
                                                }
                                        }
                                }
                }
            }
        }
<#
.SYNOPSIS
        Displays the current TCP/IP connections.

.DESCRIPTION
        Displays active TCP connections and includes the process ID (PID) and Name for each
connection.
        If the port is not yet established, the port number is shown as an asterisk (*).

.PARAMETER ProcessName
        Gets connections by the name of the process. The default value is '*'.

.PARAMETER Port
        The port number of the local computer or remote computer. The default value is '*'.

.PARAMETER Address
        Gets connections by the IP address of the connection, local or remote. Wildcard is
supported. The default value is '*'.

.PARAMETER Protocol
        The name of the protocol (TCP or UDP). The default value is '*' (all)

.PARAMETER State
        Indicates the state of a TCP connection. The possible states are as follows:

        Closed          - The TCP connection is closed.
        CloseWait       - The local endpoint of the TCP connection is waiting for a connection
termination request from the local user.
        Closing - The local endpoint of the TCP connection is waiting for an acknowledgement of
the connection termination request sent previously.
        DeleteTcb       - The transmission control buffer (TCB) for the TCP connection is being
deleted.
        Established     - The TCP handshake is complete. The connection has been established and
data can be sent.
        FinWait1        - The local endpoint of the TCP connection is waiting for a connection
termination request from the remote endpoint or for an acknowledgement of the connection
termination request sent previously.
        FinWait2        - The local endpoint of the TCP connection is waiting for a connection
termination request from the remote endpoint.
        LastAck - The local endpoint of the TCP connection is waiting for the final
acknowledgement of the connection termination request sent previously.
        Listen          - The local endpoint of the TCP connection is listening for a connection
request from any remote endpoint.
        SynReceived     - The local endpoint of the TCP connection has sent and received a
connection request and is waiting for an acknowledgment.
        SynSent - The local endpoint of the TCP connection has sent the remote endpoint a segment
header with the synchronize (SYN) control bit set and is waiting for a matching connection
request.
        TimeWait- The local endpoint of the TCP connection is waiting for enough time to pass to
ensure that the remote endpoint received the acknowledgement of its connection termination
request.
        Unknown         - The TCP connection state is unknown.

        Values are based on the TcpState Enumeration:
```

```
        http://msdn.microsoft.com/en-
us/library/system.net.networkinformation.tcpstate%28VS.85%29.aspx

.EXAMPLE
        Get-NetworkStatistics

.EXAMPLE
        Get-NetworkStatistics iexplore

.EXAMPLE
        Get-NetworkStatistics -ProcessName md* -Protocol tcp

.EXAMPLE
        Get-NetworkStatistics -Address 192* -State LISTENING

.EXAMPLE
        Get-NetworkStatistics -State LISTENING -Protocol tcp

.OUTPUTS
        System.Management.Automation.PSObject

.NOTES
        Author: Shay Levy
        Blog  : http://PowerShay.com
#>
}


#*******************************************
#*                 MAIN PROGRAM            *
#*******************************************

#Add Header to the CSV File
[string]$Previous = "TimeStamp,Protocol,LocalAddress,LocalPort,RemoteAddress,"
$Previous += "RemotePort,State,ProcessName,PID"
Add-Content $FilePath $Previous

#Initiate an infinite loop that calls the Get-NetworkStatistics Function repeatedly
#and formats the output as appropriate
while ($true) {
    $Observation = Get-NetworkStatistics $ProcName
    [string]$Current = $Observation | Out-String
    if ($Previous -ne $Current) {
        [string]$TimeStamp = Get-Date -Format o
        $Previous = $Current
        ForEach ($Socket in $Observation) {
            $Record = $TimeStamp + "," + $Socket.Protocol + "," + $Socket.LocalAddress + "," `
                        + $Socket.LocalPort + "," + $Socket.RemoteAddress + "," +
$Socket.RemotePort + "," `
                        + $Socket.State + "," + $Socket.ProcessName  + "," + $Socket.PID
                        Add-Content $FilePath $Record
            if ($PassThru) {
                                $pso2 = New-Object -TypeName PSObject -Property @{
                                        PID = $Socket.PID
                                        ProcessName = $Socket.ProcessName
                                        Protocol = $Socket.Protocol
                                        LocalAddress = $Socket.LocalAddress
                                        LocalPort = $Socket.LocalPort
                                        RemoteAddress = $Socket.RemoteAddress
                                        RemotePort = $Socket.RemotePort
                                        State = $Socket.State
                                        TimeStamp = $TimeStamp
                                } | Select-Object -Property $properties

                                Write-Output $pso2
                                }
            }
        }
    }
```

```
<#
.SYNOPSIS
        Creates a log of current TCP/IP connections and optionally passes them through to the
pipeline.

.DESCRIPTION
        Logs active TCP connections and includes the process ID (PID) and Name for each
connection.
        If the port is not yet established, the port number is shown as an asterisk (*).

.PARAMETER FilePath
        The path and file name of the log file. Mandatory.

.PARAMETER ProcName
        Log only connections with the name of the process provided. The default value is '*'.

.SWITCH PassThru
        Return a process object to the screen or the pipeline.

.EXAMPLE
        Log-Connections mylog.csv

.EXAMPLE
        Log-Connections mylog.csv svchost

.EXAMPLE
        Log-Connections mylog.csv svchost -PassThru

.EXAMPLE
        Log-Connections -Filepath mylog.csv -ProcName svchost

.EXAMPLE
        Log-Connections mylog.csv svchost -PassThru | Format-Table

.EXAMPLE
        Log-Connections mylog.csv svchost -PassThru | Out-GridView

.OUTPUTS
        System.Management.Automation.PSObject

.NOTES
        Author : Kenneth G. Hartman
        Blog   : http://www.KennethGHartman.com
        Credits: Author of the Get-NetworkStatistics Function is
             Shay Levy   http://PowerShay.com  http://poshcode.org/2701
#>
```

8.6. Appendix 6 – Plot-IOStats.py

The Plot-IOStats.py program will create a graph of the bytes transmitted over time for each connection in a file created by the Log-Connections.ps1 PowerShell script. (Refer to the Log-Connections User Guide for more information.)

The purpose of the program is to visualize the relative timing of various remote connections relative to each other. Plot-IOStats.py can accommodate a large number of remote connections by creating multiple PDF pages when the number of plots per page is exceeded.

The source code, example output files and complete documentation is available at http://www.kennethghartman.com/projects/Plot-IOStats.html

8.6.1. Plot-IOStats.py Source Code

```
import subprocess
from datetime import datetime
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.backends.backend_pdf import PdfPages

log_file = "log.csv"
tshark_path = '"c:\\program files\\wireshark\\tshark"'
pcap_path = '"capture.pcap"'
stats_file = "IOstats.txt"
tshark_batchfile = "Generate.bat"
pdf_output = 'TrafficPlots.pdf'
number_plots_per_page = 10
suppress_y_tick_labels = False

#Read the list of remote IP addresses from the log file and uniquify
ipdata = np.genfromtxt(log_file,delimiter=",", skip_header=1, skip_footer=2, usecols=4,
dtype="S30",autostrip=True)
ipList = list(set(ipdata) - set(['','*','0.0.0.0','RemoteAddress']))
if len(ipList) == 0:
    raise Exception, "The list of IP addresses is empty."

#Construct the tshark filter string
filter = '"io,stat,1'
for ipAddress in ipList:
    filter = filter + ",tcp&&ip.src==" + ipAddress +",tcp&&ip.dst==" + ipAddress
filter = filter +'"'

#Construct the Windows shell command and call it
# Reference http://www.wireshark.org/docs/man-pages/tshark.html
exec_string = tshark_path + " -q -nr " + pcap_path + " -t ad -z " + filter + " > " + stats_file
#Write the shell command to a batch file for reference
f = open(tshark_batchfile, 'w')
f.write(exec_string)
f.close()

proc = subprocess.Popen(exec_string, shell=True)
proc.wait()
if proc.returncode == 0:
    print("\r\nThe command:\r\n")
    print(exec_string)
    print("\r\nwas executed to produce TSHARK IO Statistics output file at:\r\n")
    print(stats_file)
else:
    raise Exception, "Issue creating TSHARK IO Statistics output."



#Determine the columns of data to use in the stats_file and their type
columnList = [1]
datatype="S30"
for i in range(len(ipList)):
    columnList.append((i+1)*4+1)
    columnList.append((i+1)*4+3)
    datatype = datatype + ",int,int"
```

```
#Read the data out of the IO Stats file
header_length = (len(ipList) * 2) + 10
data = np.genfromtxt(stats_file,delimiter="|", skip_header=header_length, skip_footer=2,
usecols=columnList, dtype=datatype,autostrip=True)
datestamp = [datetime.strptime(row[0], "%Y-%m-%d %H:%M:%S") for row in data]

#Clean up data
rows = len(data)
cols = len(data[0])
for j in range(1,cols):
        for i in range(rows):
                if data[i][j] > 65000000:
                        data[i][j] = 0

#Set up the Subplots
#fig = plt.figure()
fig = plt.figure(figsize=(8.5,11),dpi=100)
nb_plots = len(ipList)
nb_pages = int(np.ceil(nb_plots / float(number_plots_per_page)))
grid_size = (number_plots_per_page, 1)
pdf_pages = PdfPages(pdf_output)
ax = []
for i in range(nb_plots):
    # Create a figure instance (ie. a new page) if needed
    pltnum = i % number_plots_per_page
    if pltnum == 0:
        fig = plt.figure(figsize=(8.5, 11), dpi=100)
    # Make the Plot
    ax.append(plt.subplot2grid(grid_size, (pltnum, 0)))
    plt.ylabel(ipList[i], weight='bold')
    plt.ylabel(ipList[i])
    if not suppress_y_tick_labels:
        ax[i].tick_params(axis='y', which='major', labelsize=6)
    else:
        ax[i].set_yticklabels("")
    ax[i].set_ylabel(ipList[i], rotation='horizontal')
    ax[i].tick_params(axis='x', which='major', labelsize=6)
    rxbytes = [row[i*2+1] for row in data]
    txbytes = [-row[i*2+2] for row in data]
    ax[i].plot(datestamp, rxbytes, color='red')
    ax[i].plot(datestamp, txbytes, color='blue')
    ax[i].grid(True)
    # Close the page if needed
    if (i + 1) % number_plots_per_page == 0 or (i + 1) == nb_plots:
        #fig.autofmt_xdate()
        plt.tight_layout()
        pdf_pages.savefig(fig)

# Write the PDF document to the disk
pdf_pages.close()
```

8.7. Appendix 7 – SimpleGeoIPcity.py

The `SimpleGeoIPcity.py` program will perform a lookup of the geolocation data published by MaxMind.com and will report back the city and country as well as the longitude and latitude based on data from an Internet Registry’s Whois database.

The source code, example output files and complete documentation is available at http://www.kennethghartman.com/projects/SimpleGeoIPcity.html

8.7.1. SimpleGeoIPcity.py Source Code

```
# File: SimpleGeoIPcity.py
#
# Description: Perform a lookup of GeoLocation Data for IP Addresses
# Using the GeoIPCountryWhois.txt file published by www.maxmind.com
#
# Author: Kenneth G. Hartman
# Site: www.KennethGHartman.com
#
# http://geolite.maxmind.com/download/geoip/database/GeoLiteCity_CSV/
# ======================================================================

import numpy as np
import os

#Set the File Names
cityBlocksFile = 'GeoLiteCity-Blocks.csv'
cityLocationsFile = 'GeoLiteCity-Location.csv'
IPListFile = 'IPList.csv'
filename = 'IP-GeoLocateCity.csv'

#Open a new file for output
def delete_file(filename):
    try:
        os.remove(filename)
        filename + ' deleted'
    except:
        print filename + ' did not exist'
f1 = open(filename, 'a')
f1.write('IP_dotted,country,region,city,postalCode,latitude,longitude,metroCode,areaCode\n')


#Load the arrays
print "Loading GeoLocation Data...This may take a few minutes..."
textconv = lambda x: str(x).replace('"', '')
floatconv = lambda x: float(str(x).replace('"', ''))
GeoLocs = np.genfromtxt(cityLocationsFile,delimiter=",", skip_header=2,
usecols=(0,1,2,3,4,5,6,7,8), dtype="float,S2,S2,S40,S10,float,float,int,int",autostrip=True,
converters={1: textconv, 2: textconv, 3: textconv, 4: textconv})
GeoBlocks = np.genfromtxt(cityBlocksFile,delimiter=",", skip_header=2, usecols=(0,1,2),
dtype="float,float,float",autostrip=True, converters={0: floatconv, 1: floatconv, 2: floatconv})
IPList = np.genfromtxt(IPListFile,delimiter=",", skip_header=1, usecols=(0),
dtype="S16",autostrip=True)

#Define the fields in the GeoBlocks Array
startIpNum = [row[0] for row in GeoBlocks]
endIpNum = [row[1] for row in GeoBlocks]
locId_blk = [long(row[2]) for row in GeoBlocks]

#Define the fields in the GeoLocs Array
locId_loc = [long(row[0]) for row in GeoLocs]
country = [row[1] for row in GeoLocs]
region = [row[2] for row in GeoLocs]
city = [row[3] for row in GeoLocs]
postalCode = [row[4] for row in GeoLocs]
latitude = [row[5] for row in GeoLocs]
longitude = [row[6] for row in GeoLocs]
metroCode = [row[7] for row in GeoLocs]
areaCode = [row[8] for row in GeoLocs]


def convertIP(address):
    temp = address.split(".")
    convertedAddress = (int(temp[0]) * 16777216) + (int(temp[1]) * 65536) + (int(temp[2]) * 256) +
(int(temp[3]))
    return convertedAddress
```

```
for j in range(len(IPList)):
    IP_dotted = IPList[j]
    IP_number = convertIP(IP_dotted)
    for i in range(len(GeoBlocks)):
        if ((IP_number > startIpNum[i]) and (IP_number < endIpNum[i])):
            idx = locId_loc.index(locId_blk[i])
            outstring = IP_dotted +','+ str(country[idx]) +','+ str(region[idx]) +','+
str(city[idx]) +','+ str(postalCode[idx])
            outstring = outstring +','+ str(latitude[idx]) +','+ str(longitude[idx]) +','+
str(metroCode[idx])
            outstring = outstring +','+ str(areaCode[idx]) + '\n'
            print outstring
            f1.write(outstring)
            continue

f1.close()
```