Redundant Failover Seamless IP Stack

Institution: Indiana University – Purdue University Fort Wayne, Department of Computer Science

Team Members: Brice Aldrich
Devin Aspy
Zach Pratt

Faculty Advisor: Zesheng Chen, Ph.D.

1. Introduction and Background

The backup to critical systems is an important property for telecommunications. Most existing solutions in this field apply the backup at the application level. That is, when an application server fails, a user has to specifically switch to another server for the service. Such a backup mechanism is easy to implement, but introduces a relatively long delay for service recovery and affects the quality-of-service (QoS) to customers. In this senior capstone project, we research a different approach and attempt to provide a backup solution below the application level. Specifically, we study a redundant failover seamless IP stack (RFSIS), which is a software library used for producing a backup system without any inputs or supports from the client side. That is, in our solution, a user is not aware of the failure of a server, and all on-going connections are hold without tearing-down and re-connecting.

Our designed system can have a wide application to telecommunications and the Internet of Things (IoT) [1]. For example, our proposed RFSIS can be used for the backup to the security cameras. When a critical camera fails due to attacks or other reasons, our solution can enable the backup camera (almost) at the same time when the original camera failed. Therefore, our project can potentially lead to much robuster systems.

In this paper, we first overview the functionality of our RFSIS system and then describe in detail the system architecture. Next, we emphasize the importance of applying standards in our design. Finally, we discuss the testing process and conclude the paper.

2. System Functionality

The following two figures (i.e., Figures 1 and 2) show the primary functionality of our designed system. Initially, the RFSIS library is deployed on two different servers. One instance is

configured as the active or main server, and the other as the backup server, as shown in Figure 1. The active server communicates with both the client and the backup server. The main server services incoming requests from the client, while maintaining a heartbeat with the backup server. Once detecting a heartbeat failure, the backup server assumes the role of the main server and claims the IP address of the main server via the address resolution protocol (ARP) [2] announcement messages. At this point, the backup has fully assumed the role of the main server and begins to service client requests, as shown in Figure 2. During the entire switching process, the client is not aware of the failure of the main server, and all on-going connections are maintained.

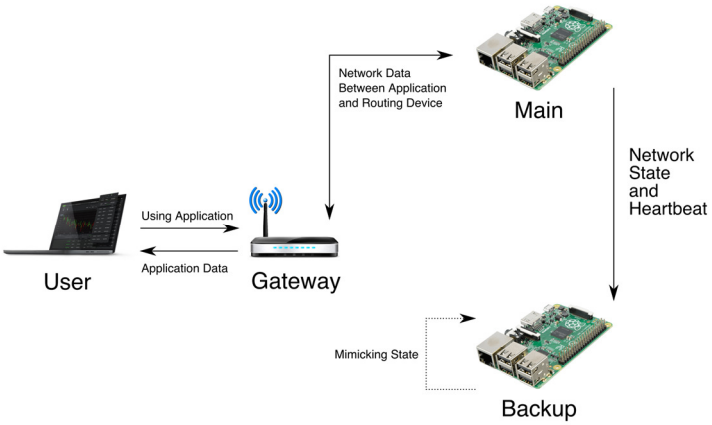

Figure 1. Pre-failover

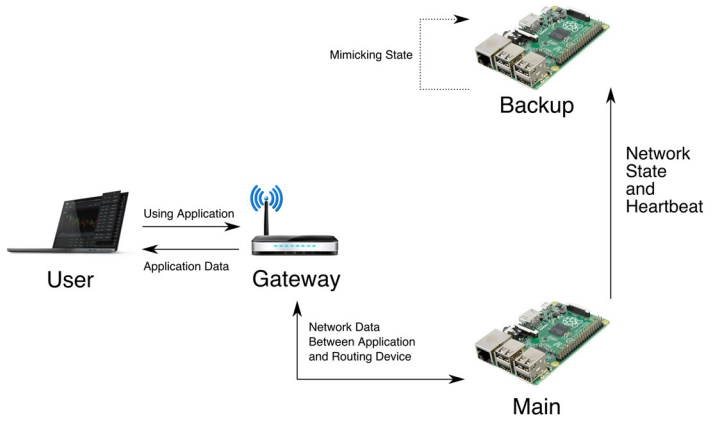

Figure 2. Post-failover

3. System Architecture

In this section, we first describe in detail the system modules and then apply the uniform modeling language (UML) [3] to introduce the class diagram of the RFSIS library.

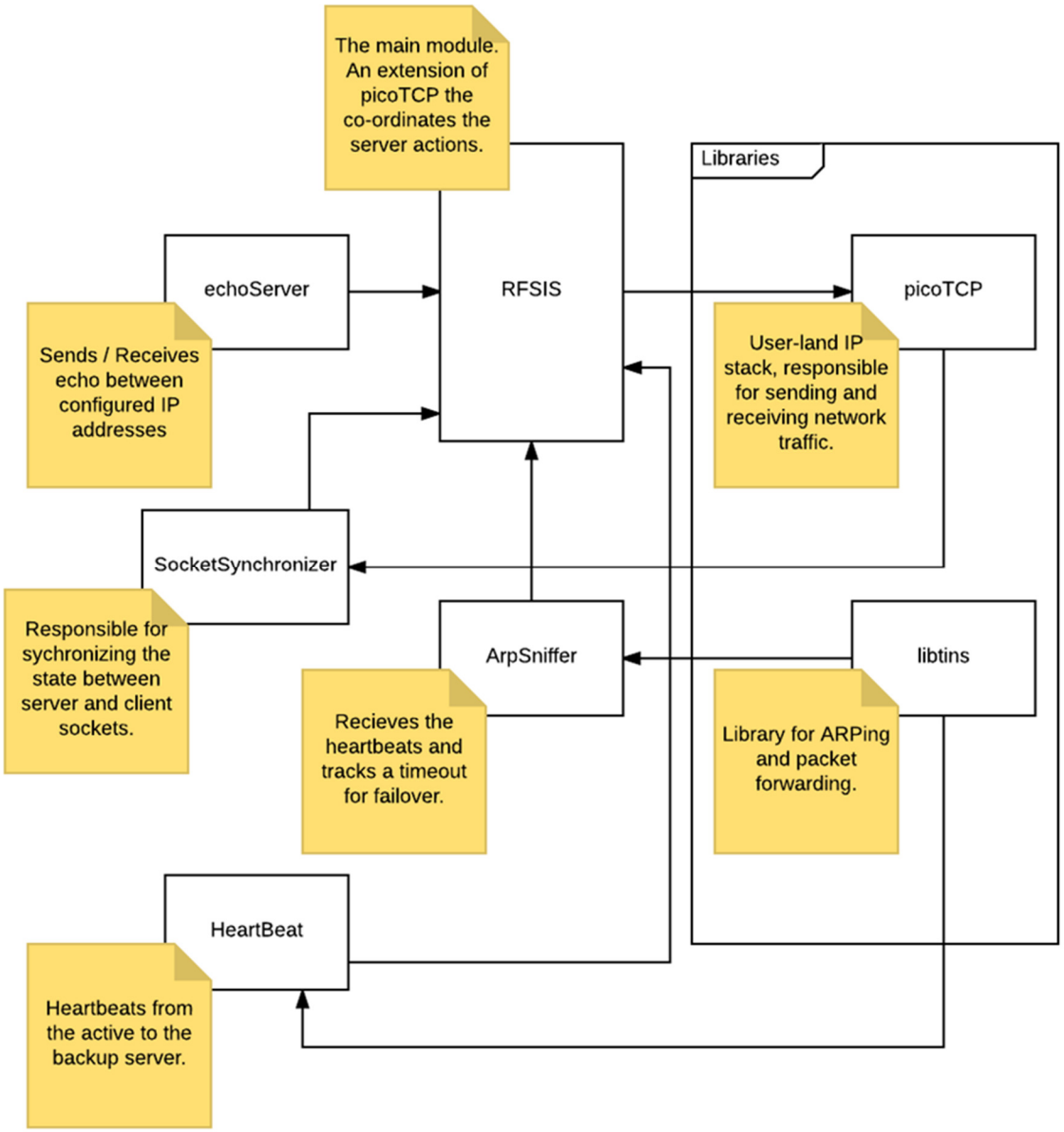

Figure 3. System Modules

2.1 System Modules

Our designed RFSIS software library is built on top of the open-source IP stack picoTCP [4], which provides a free TCP/IP stack implementation. The picoTCP is under the GPLv3 license. As shown in Figure 3, the entire RFSIS system contains the following 7 modules.

2.1.1 RFSIS Module

The RFSIS module is the main module for handling the failover of the connections. All other modules interact with this module and provide services for the backup functionality. The primary functions of the RFSIS module include taking in the callback for the server application, initializing the IP stack, and running the server application on top of the initialized IP stack.

2.1.2 PicoTCP Module

The picoTCP module is basically the library from picoTCP [4]. It abstracts all network communications through its internal memory structure and clock, and provides function calls to set up devices and initiate socket communications with the clients. The picoTCP library was built to consume little resources and can be deployed on many different operating systems such as Windows, Mac OS, and Linux.

2.1.3 Libtins Module

The libtins module is another open-source library used in our system. Specifically, libtins is a high-level, multiplatform C++ network packet sniffing and crafting library [5]. We use the libtins library to sniff packets at the backup server for incoming ARPs from the main server.

2.1.4 EchoServer Module

The echoServer module is a written application utilizing the service provided by the RFSIS module. It is an either user datagram protocol (UDP) [6] or transmission control protocol (TCP) [7] server that connects to an outside client such as Netcat [8]. When the echoServer module receives a string/message from the outside client, it then sends that same string/message back to the client. We use this simple application as an example of our failover system. A developer, who builds an application using our failover system, would not need to include this application in their code, but refers to this example for the way to call functions in the RFSIS module.

2.1.5 ArpSniffer Module

The ArpSniffer module handles the detection of the failure of the main server using the timed-out mechanism at the backup server. It uses the libtins library to sniff ARP messages from the main server. Specifically, if the ArpSniffer module have not received ARP messages for a specified time duration based on a timer, then this module will notify the RFSIS module that the main server has timed out and that the backup should now take over as the active server.

2.1.6 HeartBeat Module

The HeartBeat module is only used at the main server and sends ARP messages periodically to the backup server. We call such an operation *heartbeating*, as it lets the backup server know that the main server is still alive and functioning. The HeartBeat module has a timer that is used to send an ARP message every specified number of milliseconds.

2.1.7 SocketSynchronizer Module

The SocketSynchronizer module is used to synchronize the TCP/IP internal status, such as SYN and ACK numbers for a TCP connection [7], between the main server and the backup server. The implementation of this module applies UDP for communications between two servers and uses JavaScript Object Notation or JSON [9] as the data format. Specifically, the main server periodically sends the TCP/IP internal status information through JSON and UDP to the backup server so that the backup server can update the internal status of TCP/IP stack and prepare for future takeover. Unfortunately, at this point the SocketSynchronizer module is not fully functional.

2.2 Class Diagram

Figure 4 lays out a class diagram for the RFSIS application, based on the UML. It can be seen that the RFSIS class is the central class, interacting with other classes. An application defined in the Main class uses the RFSIS class and passes the callback function (e.g., startServer() function in the EchoHelper class) to it. Moreover, the same code is installed to both main and backup servers. We use a configuration text file to indicate which instance a server is running.

The source code of our RFSIS project can be found from the GitHub repository [10].

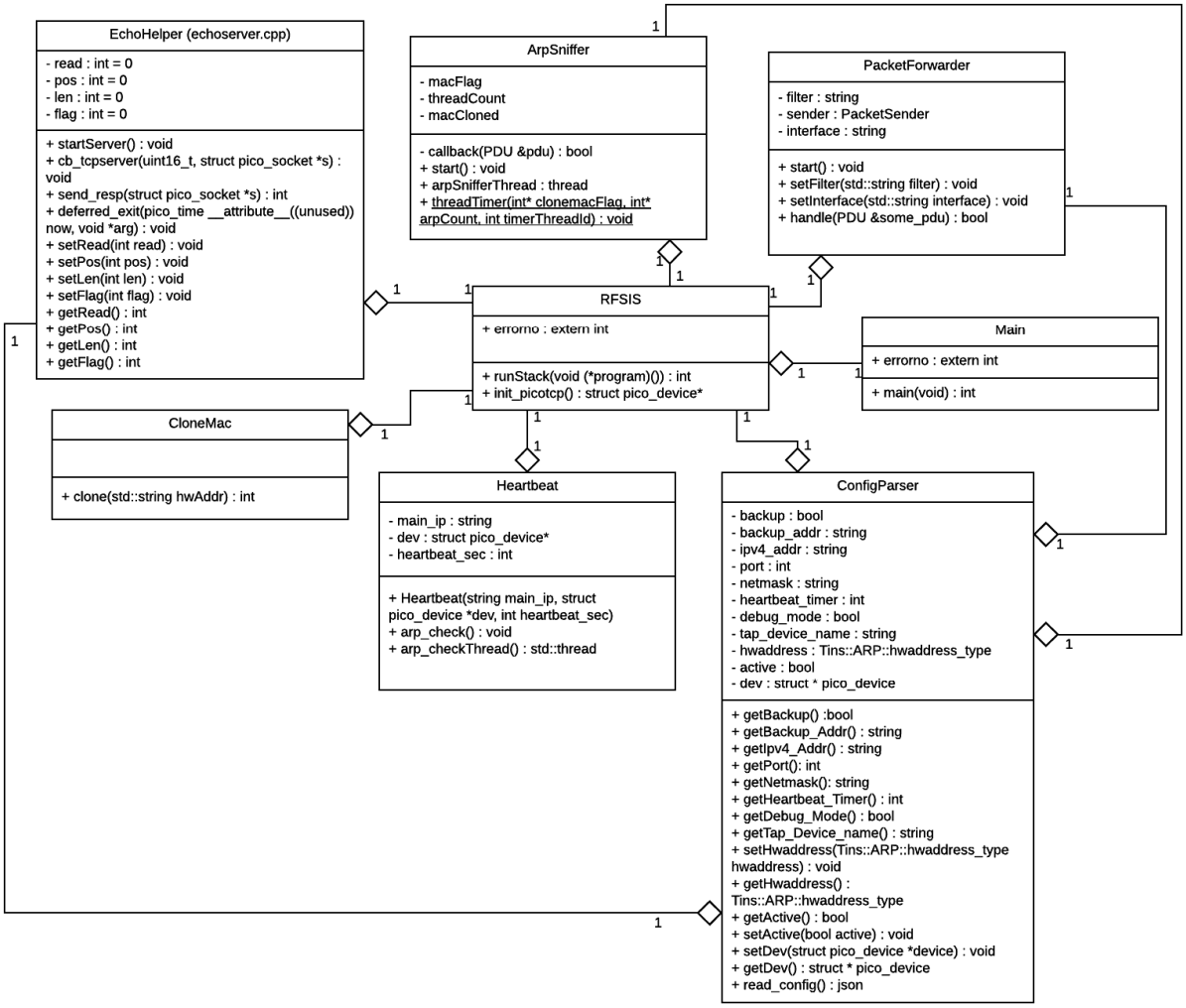

Figure 4. Class Diagram

3. Standards Applied

In this section, we discuss the standards applied to our RFSIS system. Specifically, we emphasize how a standard or protocol affects our design.

3.1 Internet Protocol (IP) [11]

The IP standard specifies the way to identify a computer in the Internet. Currently, there are two versions of the IP standard in use: version 4 and version 6. In this project, we focus on version 4, i.e., IPv4, where a host uses 32 bits as its address. Since our goal is to let the client side not be aware of the failure of the main server, both main and backup servers need to use the same IP address to communicate to the client. As the result, when the backup server detects the failure of

the main server, it should spoof the IP address of the main server and use it to continue the communication with the client. We implemented such a spoofing function in the RFSIS module based on the services provided by the picoTCP library.

3.2 User Data Protocol (UDP) [6]

The UDP standard specifies an unreliable, but fast communication between two computers. In our project, we applied UDP for two different purposes. First, we used UDP in the SocketSynchronizer module for fast communication between the main server and the backup server. Second, we also applied UDP in the echo server application to demonstrate the usefulness and the functionality of the RFSIS library. Based on the services provided by the picoTCP library, we implemented the UDP communication for both purposes.

3.3 Transmission Control Protocol (TCP) [7]

The TCP standard specifies a reliable, ordered, and error-checked communication between two computers. TCP is fundamental to the Internet, but is very complex for implementation. As shown in Figure 5 from [12], TCP contains multiple states and defines the conditions for state transitions. In out project, when the backup server detects the failure of the main server, it needs to be able to communicate with the client immediately without going through the TCP three-way handshaking [7]. To do so, the backup server should be able to transit from the “LISTEN” state to the “ESTABLISHED” state based on the TCP/IP internal status information received from the main server.

Using the TCP standard was one of the biggest challenges in our project. A majority of our time and resources went into researching and designing ways to communicate TCP structures between two servers and understanding the source code of the picoTCP library for implementing TCP. Unfortunately, at this point we have not fully implement this part.

3.4 Address Resolution Protocol (ARP) [2]

The ARP standard specifies the mapping between an IP address and a computer hardware address. We applied ARP for two purposes. First, the main server uses ARP to communicate with the backup server in the HeartBeat module for heartbeating. Second, when the backup server detects the failure of the main server, it spoofs the IP address of the main server through the ARP announcement messages. We implemented these two functionalities based on the function calls provided by the picoTCP library.

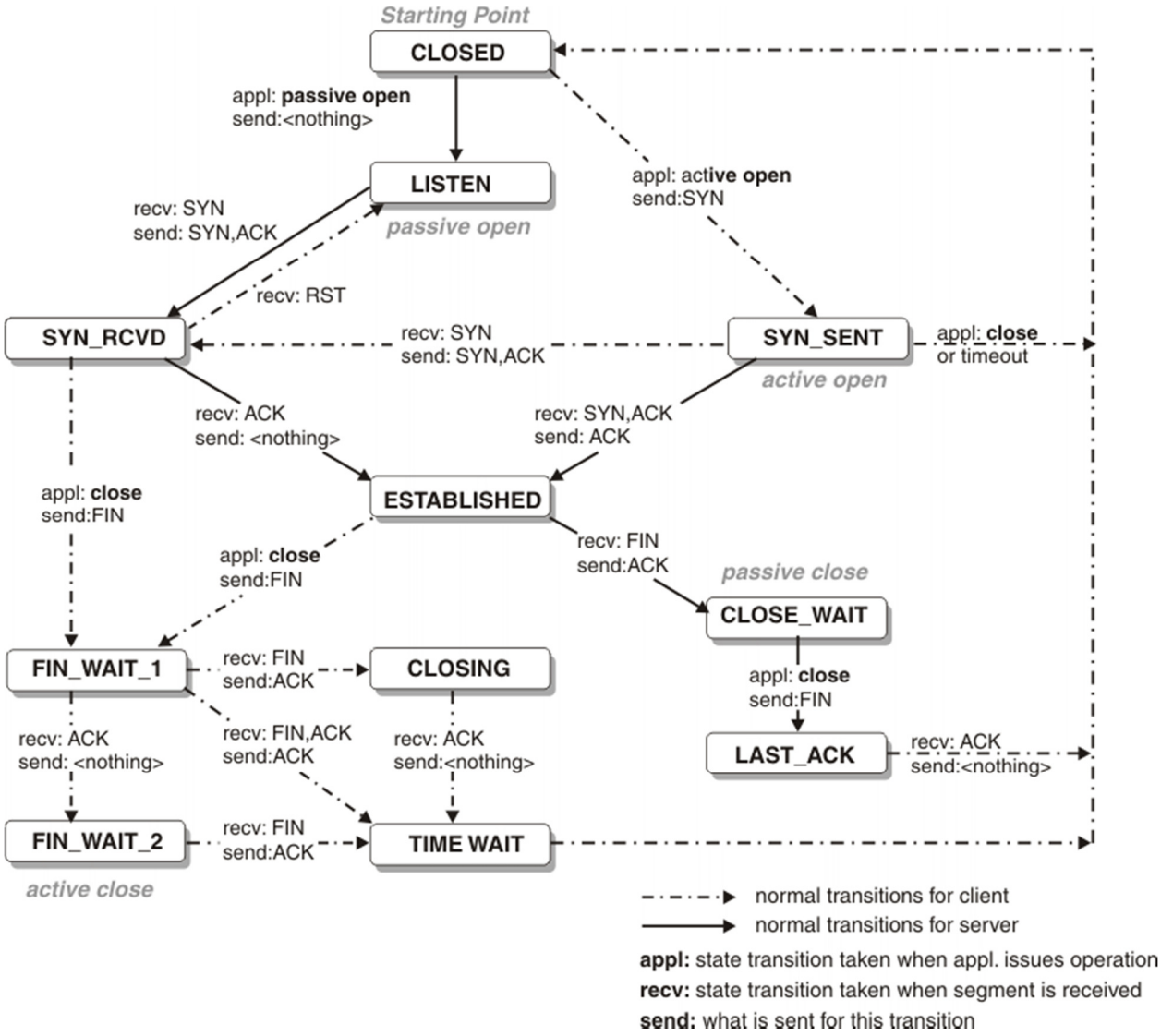

Figure 5. TCP State Transitions [12]

4. Testing

Testing was done using a set of Raspberry Pi devices [13] running the latest Raspbian (Debian-based) Linux. Devices were connected through a router on a local network. The first two devices were primarily used to test the failover functionality. The first device was configured as the main server, and the second as the backup server.

The primary test was as follows. A simple echo server application was written and passed into the RFSIS library through the use of callback functions. We then used a UDP or TCP client (Netcat) to send messages to the main server. The main server will consume the messages that were sent from the client and send back an exact copy. Next, the main server was manually

terminated. Failover was then verified by the manual inspection of the IP address of the backup server to ensure that the IP address was successfully spoofed from the original main server. Messages were again sent to the original IP address, and we observed and verified a proper echo response from the backup (now active) server.

Specifically, these steps can be summarized as the following test scenarios for the UDP echo application:

(1) The main server talks to the client for the echo application.
(2) The main server talks to the backup server through the heartbeat.
(3) The backup server can detect the failure of the main server.
(4) Once detecting the failure of the main server, the backup server can claim the IP address of the main server in a short time.
(5) The backup server can talk to the client and continue the communication.

The testing steps for the TCP echo application are similar to those for the UDP echo application, except that the main server needs to send TCP/IP internal status information to the backup sever and that the backup server can apply this information to create the TCP socket state to communicate with the client when the main server fails, which is not completed at this point.

5. Summary

In this senior capstone project, we have designed and implemented the RFSIS library that complies with the IP, UDP, TCP, and ARP standards defined in RFCs [2, 6, 7, 11]. We found these standards to be very helpful in guiding us to build a product that is easy to use and is a useful groundwork for seamless failover applications requiring a solution outside the kernel and the application layer.

References

[1] "Special Report: The Internet of Things," *IEEE Institute*, March 2014, available at: http://theinstitute.ieee.org/static/special-report-the-internet-of-things, accessed on May 2017.

[2] RFC 826, An Ethernet Address Resolution Protocol, https://tools.ietf.org/html/rfc826, accessed on May 2017.

[3] Ian Sommerville, "*Software Engineering*," Addison-Wesley (10th Edition) 2015, ISBN 978-0-13-394303-0 (Pearson).

[4] picoTCP, https://github.com/tass-belgium/picotcp, accessed on May 2017.

[5] libtins, http://libtins.github.io/, accessed on May 2017.

[6] RFC 768, User Datagram Protocol, https://www.ietf.org/rfc/rfc768.txt, accessed on May 2017.

[7] RFC 793, Transmission Control Protocol, https://www.ietf.org/rfc/rfc793.txt, accessed on May 2017.

[8] Wikipedia, Netcat, https://en.wikipedia.org/wiki/Netcat, accessed on May 2017.

[9] Wikipedia, JSON, https://en.wikipedia.org/wiki/JSON, accessed on May 2017.

[10] Github, Redundant Failover Seamless IP Stack (RFSIS), https://github.com/zppratt/rfsis, accessed on May 2017.

[11] RFC 791, Internet Protocol, https://tools.ietf.org/html/rfc791, accessed on May 2017.

[12] IBM Knowledge Center, TCP State Transitions, https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.halu101/constatus.htm, accessed on May 2017.

[13] Raspberry Pi, https://www.raspberrypi.org/, accessed on May 2017.