Protocol Dissector Tool for Deoding in Band Packet Header on A Switch

Dipayan Sinha, Shobha G.

Abstract: Serviceability of networks is a vital part of network management which helps in isolating faults and triaging network issues. Packet analyzers help in identifying faults, security threats and other implementation flaws in the networking software by capturing network traffic and analyzing it. Packet analyzing is heavily based on protocols which need to be decoded from the raw format and presented to the user in an understandable format. In this work, a Command Line Interface based protocol dissector tool has been developed which runs on the operating system of a switch and performs packet decoding by capturing in band packets flowing between control and data plane of the switch. The tool also provides support for packet filtering in order to only capture packets which the user needs. Existing packet dissectors run on Wireshark in the form of Lua plugins. However, in this work the implementation of the entire system is based on C. Some of the public protocols decoded by this tool involve IPv4, IPv6, UDP, TCP, ARP, ICMP and so on. Also, this tool supports decoding of private protocols as well.

Keywords: Decoding, In Band Packet, Network Element, Packet Capture

I. INTRODUCTION

Computer networks are responsible for interconnection of networking elements around the world which enable users to communicate over the Internet. In computer networking terms, a network element is a manageable logical entity which unifies multiple physical devices that can be maintained from a single management perspective. A computer network is based on several public and proprietary protocols which define the format of messages over which communication can take place as well as rules for exchanging messages between different networking elements. A network protocol defines the syntax and semantics of communications. Network analysis, also called packet sniffing is the method of capturing the network traffic and analyzing it to understand how the network is behaving. Packet sniffing helps in determining the origin of the packet, destination of the packet, data carried by it as well as other information specific to the protocol. A packet analyzer decodes the captured packets which follow protocol definitions that are known to the analyzer. The decoded contents are then displayed in a human-readable format to the user. Network analysis helps in overcoming situations of network outrages. Understanding of traffic flow in the network helps in overcoming flaws in the design of the network software as well as any security flaws which may compromise privacy. A network outrage refers to the failure of switches, routers and other network elements in the network infrastructure. It may occur due to failure of hardware & software components as well as configuration errors in the network setup. Failure of switches and routers can lead to significant amount of network downtime which can be damaging. In order to triage network issues, serviceability of network elements is needed. Serviceability is an important feature for identifying faults and debugging as well as isolating faults to root cause analysis. Packet analyzers are efficient tools for triaging network issues. They are used for testing and monitoring networks, devices and software implementations running on them. It may also be used for malicious purposes. Protocol dissectors are codes that run on a packet analyzer to dissect and analyze the captured network data stream. Dissectors are specific to protocols. New protocols are coming up and hence new dissectors are written to decode such protocols. The network element under study in this paper is a switch. This work studies the serviceability of network switches by decoding the in-band packets flowing between control plane and data plane of a switch. A Command Line Interface tool is developed to run on a switch which analyses packets and shows the header and packet data in an easily understandable format. The remainder of the paper is organized as follows. Section II comprises of survey of recent works on Packet Analyzers and Dissectors. Section III defines the problem statement and Section IV identifies the objectives of the work. This is followed by Section V which describes the methodology. Section VI comprises of the System Architecture followed by the Implementation details in Section VII. Section VIII describes the results of the work. Section IX and X contain concluding remarks and future work respectively.

Revised Manuscript Received on April 25, 2020.
* Correspondence Author

Dipayan Sinha*, Dept. of Computer Science & Engineering, RV College of Engineering, Bangalore, India. Email: romulus.sinha@gmail.com
Dr. Shobha G., Dept. of Computer Science & Engineering, RV College of Engineering, Bangalore, India. Email: shobhag@rvce.edu.in

II. LITERATURE SURVEY

The authors of [1] have developed an automated feature to study the expected behaviour of network protocols and all possible variations. The tool helps in automating the process of analyzing network packets. Their model takes a labelled PCAP file as an input and a hint of how to fix the error in that PCAP file. The output produced by the tool is a model which describes the protocol behaviour and provides an interface to utilize the model for diagnostic activities. In [2], the researchers studied the traffic coming from the internet by applying filters on HTTP protocols and presented them in a graphical format using Wireshark. The system development method used in this research is NDLC – Network Development Lifecycle. It involves network analysis, designing network monitoring schedule and evaluation of network monitoring. The authors of [3] compare 2 packet sniffing tools namely Tcpdump and Wireshark. They concluded that Tcpdump performs better than Wireshark with respect to battery, memory and processor usage. However, Wireshark is more capable than Tcpdump in packet analysis and speed of capture. The drawback identified in this work is that Wireshark is a GUI based tool and TCP dump is a CLI tool, hence, the memory & processor usage is not an apt comparison. In [4], the authors provide a method to monitor and analyze the traffic flowing in the network using UDP which removes the existing deficiency of older tools used for studying network traffic. A hierarchical network model has been implemented in this work. In [5], the researchers have developed a custom protocol dissector using Lua scripting language for Wireshark. They used this dissector to identify new DDOS variants over a period of five-month study. A Botnet Protocol dissector was developed using Lua and Wireshark for identifying protocol fields.

The researchers of [6] have developed a tool called Flowscope. This tool helps to capture and store packets in an in-memory ring buffer. The packets can be filtered and dumped to disk on the occurrence of a specified trigger event. The authors reported results of 120 Gbit/s with 128-byte packets. The authors of [7] discuss about language security and static analysis of Lua, an open source framework. This is the first of its kind tool for public Static Analysis for Security Testing (SAST). This work focusses on detecting web vulnerabilities. In [8], the authors have implemented an embedded packet logger (EPL) which is used for packet sniffing in switches. This system analyses audit log data collected from server and network equipment such as switches and routers. The authors of [9] have discussed the working of Wireshark tool, disadvantages, and used traceback mechanism to improve its functioning. They have developed a packet marking algorithm in Wireshark to improve its Intrusion detection capability. The authors of [10] have proposed an intrusion detection system using a packet sniffer which traces packets by linking to a network adapter in promiscuous mode. The packets are then traced based on filters set by the network administrator.

In [11], the authors demonstrate the application of Wireshark in protocol diagnosis and its purpose to discover traditional network attacks such as port scanning, ICMP-based attacks, BitTorrent-driven denial service, etc. The authors of [12], provide a comparison between Wireshark and Colasoft Capsa as Packet Sniffing Tools. They are compared on basis of their features, characteristic behaviour as well qualitative and quantitative parameters. This paper concludes that Wireshark is a more powerful tool in comparison to Colasoft Capsa with respect to throughput, response time and packet drops however Colasoft Capsa is more user friendly. In [13], the authors analyze the TCP and UDP packets sent through email through an open source tool called Wireshark. Moreover, the decoding of TCP and UDP fields are explained in detail. The authors of [14], have considered the architecture of network traffic of an institute for monitoring and analyzing various protocols like TCP/IP, HTTP, ARP, ICMP.

In [15], researchers have focused on the basics of Packet Sniffing tools by studying their working mechanism in a comparative manner. Different packet sniffers such as Wireshark, TCPdump, Nmap, Zenmap, Kismet, Caspa, Ntop, Dsniff have been analyzed. The author of [16] discusses the need for generating custom protocol dissectors for new protocols which are not present in Wireshark. The author has come up with a tool that automates the generation of protocol dissectors using Lua for Wireshark. The authors of [17] describe the working of a packet sniffer in both switched and non-switched environment. Also, the authors study about its practicality by comparing its positive and negative aspects. It involves the use of Wireshark for recognizing Network, Transport and Application Layers. The existing system had only packet capturing capability which involved showing the size of the packets and no packet sniffing. Packet sniffing concept allows the tracing of source and destination addresses. This research work is used for Network Traffic Analysis and Intrusion Detection System. However, there is potential for exploitation of IDS by hackers due to ARP Spoofing. In [18], the authors discuss about the different packet capture mechanisms such as DashCap and nCap. This work explains the packet capture techniques in real time network traffic by comparison of different software based and hardware-based solutions. The authors of [19] discuss the creation of a packet sniffer tool just like Wireshark for decoding packet fields. A library known as Libcap has been used for capturing packets. Libpcap library helps in capturing packets directly from the network adapter. In [20], the authors have illustrated the application of Wireshark in the form of a sniffing tool in networks. An experimental setup demonstrates the efficiency of detection of a malicious packet in any network.

III. PROBLEM STATEMENT

The survey on the existing works reveals a wide variety of tools for packet capturing and analysis on host machines. Also, it is evident from previous works that Wireshark is the most prominent packet analyzer tool available as open source software. However, packet analyzers do not come directly packaged with the operating system running on the switch and often needs to be set up separately before analysis can take place. Hence, there is a need to develop a packet analyzer and decoder which can work seamlessly on the switch by integrating it with the operating system running on the switch which can perform the same tasks like Wireshark.

IV. OBJECTIVES

The aim of this study is to develop a protocol dissector for decoding the in-band header of a packet flowing between data plane and the control plane of a switch through a generalized approach. This feature is developed in the form of a Command Line Tool for triaging faults on the switch with extensive support for filter-based packet decoding.

V. METHODOLOGY

The packet sniffing process can be dividing into 3 major stages namely Capturing, Decoding and Analyzing. In the capturing phase, the raw binary data is captured from the physical interface which is converted into a more user understandable form by the decoder. Finally, the decoded packet is analyzed for extracting information from the packet.The primary step in protocol decoding is to understand the format of the protocol. The protocol may be either a public or a proprietary protocol. This is done by creating a JSON configuration file which stores the metadata fields of the protocol header. The configuration file stores information regarding the offset, mask, field value, field name as well as different types and options for each field value. This JSON configuration file then needs to be parsed for decoding the header fields when the packet is captured and stored in the capture buffer. A packet capture module is developed for capturing and storing the incoming and outgoing in band packet into a capture buffer. The packets coming from the data plane are captured at the control plane and utilized for header decoding. Packet decoding is done by developing a custom dissector in C by parsing the JSON file and populating C structures with metadata details. Capture Filter Module is developed for applying filters to the capturing process in order to capture only those packets which are of interest to the developer. Finally, a command line interface tool in the management plane is developed to enable user to specify packet filters as well as start and stop packet capture.

VI. SYSTEM ARCHITECTURE

The system architecture consists of 'n' network elements as shown in Fig. 1. which are capable of communicating with each other. Each network element consists of:

A. Management Plane

This plane is used for interaction between the user and the network element. The network element under consideration is a switch. This layer contains the OS that runs on the switch and allows users a CLI environment as well as SNMP for management activities.

B. Control Plane

This layer operates between the Management and the Data Plane. The software in the Control Plane uses a General-Purpose CPU. It is a logical layer which has two kinds of packet flows – In band and Out of Band Flows. In band flows have data and control flowing in the same path. Out of Band flows have data and packets flowing over different paths. Out of Band Packets are used for programming the Data Plane which houses the physical hardware for forwarding. In band packets are of two kinds:

- Punt Packet – Packets flowing from Data Plane to Control Plane
- Inject Packet – Packets flowing from Control Plane to Data Plane

C. Data Plane

This layer contains the hardware (ASIC) for forwarding packets between different switches. This is the physical layer in the switch where actual decoding happens.

The in-band decoder is a functional module which is invoked at the control plane upon arrival or transfer of an in-band packet.

VII. IMPLEMENTATION

The implementation of the system on the switch consisted of the following modules as shown in Fig. 2.

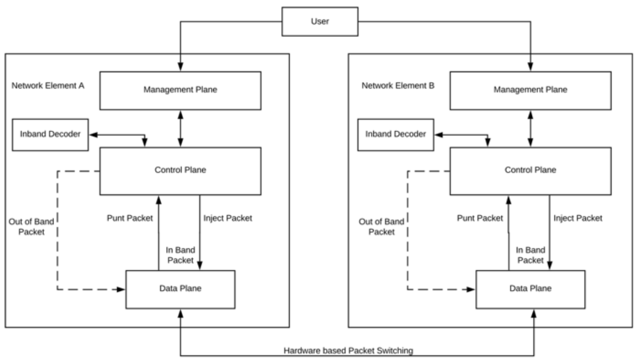


Fig. 1. System Architecture

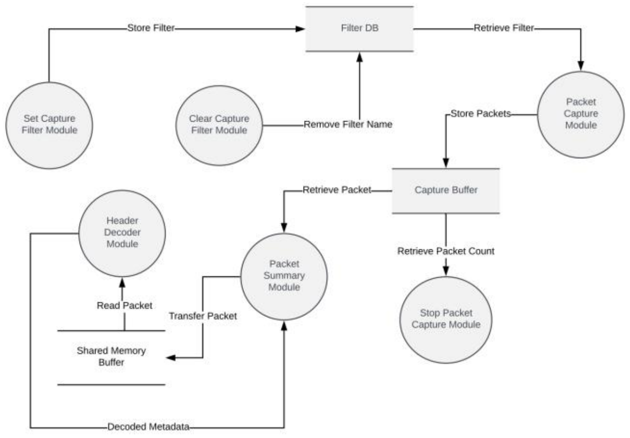


Fig. 2. Modular Design of the System

A. Packet Capture Module

This module is responsible for initiating packet capture. In band packets flowing between the data plane and the control plane are captured based on the type of filter set by the user and then stored in a capture buffer. A capture thread is spawned after acquiring a mutex lock. This thread receives the packet, processes the packet header and then stores it in the capture buffer. If the lock is already acquired, packet capture is not permitted.

B. Set Packet Filter Module

This module deals with the setting up of a capture filter based on Wireshark standards. Standard filters such as ARP, IPv4, IPv6, ICMP, UDP and MAC are supported by this in addition to Punt and Inject proprietary filters. The filter is set by the user at the time of initiating packet capture. The filter string entered by the user on CLI needs to be validated to ensure that it is a valid string and then only it should be put into a filter database. If the filter string is invalid, an appropriate error message is shown to the user. The packets are then collected in the capture buffer only if they match the filter, else they are discarded.

C. Stop Packet Capture Module

This module is responsible for putting an end to the capturing process by releasing the mutex lock and killing the capture thread. The total number of captured packets are then retrieved from the capture buffer and presented to the user.

D. Show Packet Summary Module

Once, the packet capture process is complete, the contents of the packet need to be decoded and then displayed to the user in an understandable format. This module reads the packets from the buffer in an iterative manner and invokes the decoder module to parse the packet contents. The results of the parsing are then presented to the user on the CLI.

E. Packet Header Decoder Module

This module parses the packet header in a byte by byte manner. The header decoding begins by parsing the ethernet frame comprising of the destination and source MAC address followed by the ether type field. The ether type field is a 2-byte field which indicates the type of protocol in the payload. Based on the protocol type, the remainder of the packet is decoded. Public and private protocol structures are contained in the form of JSON configuration files which is parsed to decode the protocol fields. Each protocol field has a definite mask value, offset, byte length, options and a shift value which is essential to the decoding process. After the entire packet has been parsed, the result is sent to the Show Packet Summary Module which displays the decoded contents to the user for further analysis.

F. Clear Packet Filter Module

This module enables the user to remove the existing filters from the database and clears the packets captured in the capture buffer. The system is again reset to the start state.

The topology of the experimental setup is as described in Fig. 3. Three switches are connected via ethernet interfaces to each other as well as well as to a common gateway. The protocol dissector tool is run on one of the switches which captures the packets arriving at its interface where it is processed and displayed in the CLI tool to the user.

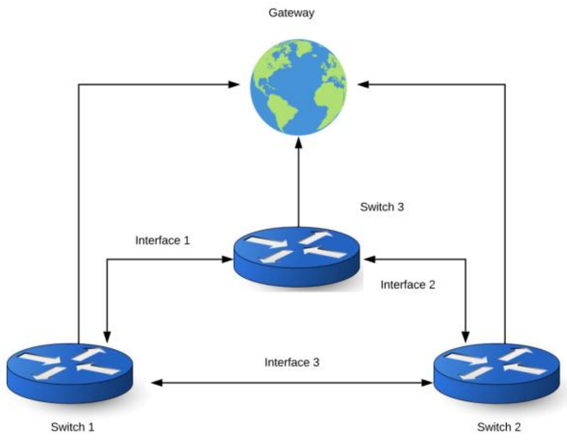
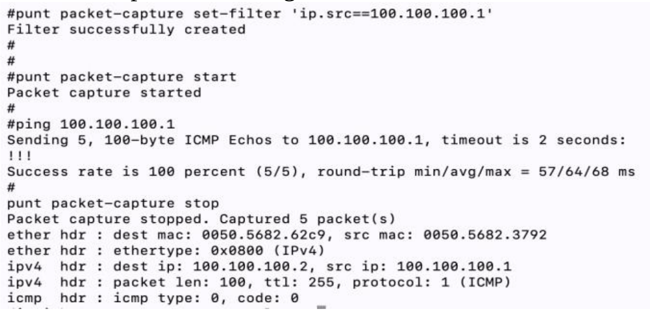


Fig.3 Topology of the experimental setup

VIII. RESULTS AND DISCUSSION

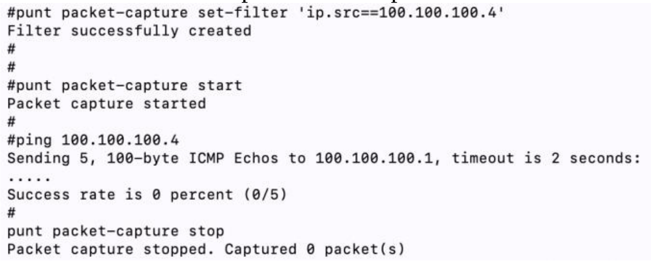
Fig. 4. shows the setting up of filter for a punt packet coming from Source IP Address 100.100.100.1. A total of 5 ICMP echo request packets is sent to 100.100.100.1 in order to test the working of the capture filter. Once packet capture is stopped, it is seen that 5 packets were captured and on decoding the packet contents it was seen that these packets were ICMP packets coming from source IP 100.100.100.1.



```
#punt packet-capture set-filter 'ip.src==100.100.100.1'
Filter successfully created
#
#
#punt packet-capture start
Packet capture started
#
#ping 100.100.100.1
Sending 5, 100-byte ICMP Echos to 100.100.100.1, timeout is 2 seconds:
!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 57/64/68 ms
#
punt packet-capture stop
Packet capture stopped. Captured 5 packet(s)
ether hdr : dest mac: 0050.5682.62c9, src mac: 0050.5682.3792
ether hdr : ethertype: 0x0800 (IPv4)
ipv4  hdr : dest ip: 100.100.100.2, src ip: 100.100.100.1
ipv4  hdr : packet len: 100, ttl: 255, protocol: 1 (ICMP)
icmp  hdr : icmp type: 0, code: 0
```

Fig. 4. Demonstration of Packet Filter Process

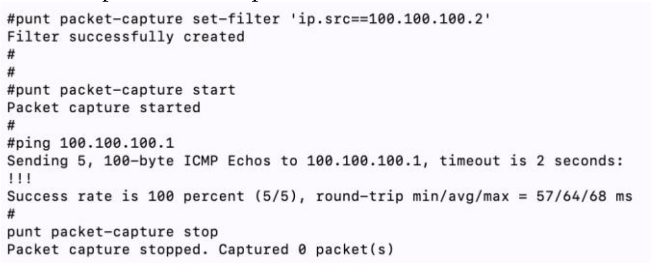
Fig. 5. illustrates the case when a filter is set for Source IP Address 100.100.100.4 but there is no interface with that IP Address and hence no packets are captured.



```
#punt packet-capture set-filter 'ip.src==100.100.100.4'
Filter successfully created
#
#
#punt packet-capture start
Packet capture started
#
#ping 100.100.100.4
Sending 5, 100-byte ICMP Echos to 100.100.100.1, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)
#
punt packet-capture stop
Packet capture stopped. Captured 0 packet(s)
```

Fig. 5. Case where 0 packets are captured

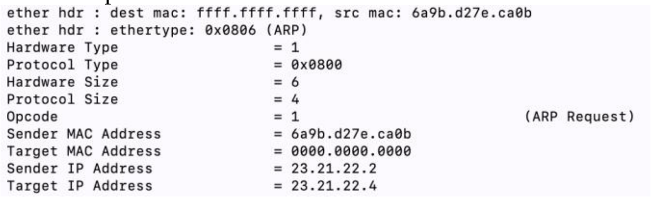
Fig. 6. illustrates the situation when a filter is set for Source IP 100.100.100.2 but ICMP echoes are sent to 100.100.100.1. In this situation packets from 100.100.100.2 are encountered but due to the capture-filter they are not stored in the buffer. Hence, 0 packets are captured.



```
#punt packet-capture set-filter 'ip.src==100.100.100.2'
Filter successfully created
#
#
#punt packet-capture start
Packet capture started
#
#ping 100.100.100.1
Sending 5, 100-byte ICMP Echos to 100.100.100.1, timeout is 2 seconds:
!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 57/64/68 ms
#
punt packet-capture stop
Packet capture stopped. Captured 0 packet(s)
```

Fig. 6. Successful Packet Filtering Process

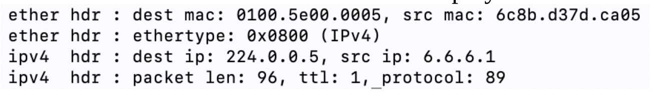
Fig. 7. shows a decoded ARP header of an incoming inject packet. This packet was obtained by sending an ARP request from the source interface with MAC address 6a:9b:d2:7e:ca:0b to a neighbouring interface (23.21.22.4) in order to update the MAC address table on the switch.



```
ether hdr : dest mac: ffff.ffff.ffff, src mac: 6a9b.d27e.ca0b
ether hdr : ethertype: 0x0806 (ARP)
Hardware Type                = 1
Protocol Type                = 0x0800
Hardware Size                = 6
Protocol Size                = 4
Opcode                       = 1                     (ARP Request)
Sender MAC Address           = 6a9b.d27e.ca0b
Target MAC Address           = 0000.0000.0000
Sender IP Address            = 23.21.22.2
Target IP Address            = 23.21.22.4
```

Fig. 7. ARP Packet Decoding

Fig. 8. shows the decoded ethernet header of the packet followed by the ether type of IPv4 (0x8000) based on which the fields of IP header are decoded and displayed on the CLI.



```
ether hdr : dest mac: 0100.5e00.0005, src mac: 6c8b.d37d.ca05
ether hdr : ethertype: 0x0800 (IPv4)
ipv4  hdr : dest ip: 224.0.0.5, src ip: 6.6.6.1
ipv4  hdr : packet len: 96, ttl: 1, protocol: 89
```

Fig. 8. Basic IPv4 Decoding

Fig. 9. shows a sample IPv4 packet which was captured by the tool and Fig. 10. presents the decoded header of the packet for analysis.

```
01 00 5E 00 00 05 6C 8B D3 7D CA 0B
08 00 45 C0 00 64 0E 36 00 00 01 59
9E 2E 16 16 16 02 E0 00 00 05
```

Fig. 9. Captured IPv4 Packet

```
Internet Protocol Version 4, Src: 22.22.22.2, Dst: 224.0.0.5
Version: 4
Header Length: 20 bytes
Total Length: 100
Reserved bit: Not set
Don't fragment: Not set
More fragments: Not set
Fragment offset: 0
Time to live: 1
Protocol: OSPF IGP (89)
Header checksum: 0x9e2e
```

Fig.10 Detailed IPv4 Decoding

Fig. 11. represents a sample IPv6 header which was captured by the tool and Fig. 12. shows the decoded form of the IPv6 header based on the ether type (0x86DD).

```
33 33 11 11 00 05 6C 2B D2 3F 8A 1a 86 DD 6C 00 00 00
00 2C 59 01 45 80 00 00 00 00 00 00 24 8B 13 FF 44 1F
24 0D 23 02 00 00 00 00 00 00 00 00 00 00 00 00 00 05
```

Fig. 11. Captured IPv6 Packet

```
ether hdr : dest mac: 3333.1111.0005, src mac: 6c2b.d23f.8a1a
ether hdr : ethertype: 0x86DD (IPv6)
ipv6  hdr : dest ip: 2302::5
ipv6  hdr : src ip : 4580::248b:13ff:441f:240d
ipv6  hdr : payload len: 44, hop count: 1, next hdr: 89
```

Fig. 12. Basic IPv6 Decoding

The packet decoding tool can be used as a serviceability tool to test the working of the software loaded on the switch. It can be used to determine if the packet is coming from the right source or whether it is going to the correct destination. For instance, by decoding the ARP header of the packets, we can see whether the packet transmission is happening from the correct interface based on which the MAC table is constructed on the switch. Not only that, the captured packets can give information about the data that is flowing through the switch which can be used to detect privacy and security issues in the network. This tool can be packaged in the switch without the need for a separate Wireshark tool to act as a packer analyzer. Such a feature is very useful for test engineers and developers who are constantly developing and providing updates on the switch software.

IX. CONCLUSION

This work aims to perform capturing and decoding of in band packets on a switch by developing a CLI tool which runs on the switch. Packet capture, decoding and analysis can help in identifying design and implementation issues in the software running on the switch. This work explains the different types of packets flowing between data plane and control plane of the switch. It also describes the design of how a packet capture and decoder tool can be implemented for any networking element in the absence of a supporting tool like Wireshark. Finally, some examples of header decoding of public protocols such as ARP, IPv4 and IPv6 have been shown to give an idea of how the system works in practice.

FUTURE WORK

This is a preliminary tool which works on limited protocols. This system needs to be extended for a variety of custom proprietary protocols. Also, the process of creating the configuration JSON file can be automated to generate the protocol structure of new protocols without any user interface. Lastly, a GUI implementation in the form of an app on the switch can make the tool more user friendly.

REFERENCES

1. Holkovicˇ, M., Ryšavý, O. and Polcˇák, L. ‘Using Network Traces to Generate Models for Automatic Network Application Protocols Diagnostics’,
   In Proceedings of the 16th International Joint Conference on e-Business and Telecommunications (ICETE 2019), pages 37-47
2. Siswanto, Apri & Syukur, Abdul & Abdul Kadir, Evizal & Suratin,. (2019). ‘Network Traffic Monitoring and Analysis Using Packet Sniffer’, 10.1109/COMMNET.2019.8742369.
3. Piyush Goyal, Anurag Goyal, ‘Comparative Study of two Most Popular Packet Sniffing Tools- Tcpdump and Wireshark’, 9th International Conference on Computational Intelligence and Communication Networks, 2017, pp.77-81
4. Md Ruhul Islam, Tawal K. Koirala and Ferdousi Khatun, ’ Network Traffic Analysis and Packet Sniffing Using UDP’, 9th International Conference on Computational Intelligence and Communication Networks, 2017, pp.907-914
5. Max Gannon, Gary Warner, Arsh Arora, ‘An Accidental Discovery of IoT Botnets and a Method for Investigating Them With a Custom Lua Dissector’, Annual ADFSL Conference on Digital Forensics, Security and Law, 2017, pp.26-3
6. Paul Emmerich, Maximilian Pudelko, Sebastian Gallenmu ̈ller, Georg Carle, ‘FlowScope: Efficient Packet Capture and Storage in 100 Gbit/s Networks’, IFIP Networking Conference, Sweden, 2017
7. Andrei Costin, ‘Lua code: security overview and practical approaches to static analysis’, IEEE Symposium on Security and Privacy Workshops, 2017, pp.133-142
8. Chanankorn Jandaeng, ‘Embedded Packet Logger for Network Monitoring System’, Springer International Publishing Switzerland 2016, pp.1093-1102
9. S.Pavithirakini,D.D.M.M.Bandara, C.N.Gunawardhana, K.K.S.Perera, B.G.M.M.Abeyrathne, Dhishan Dhammearatchi, ‘Improve the Capabilities of Wireshark as a tool for Intrusion Detection in DOS Attacks’, International Journal of Scientific and Research Publications, Volume 6, Issue 4, April 2016
10. Yash Ketkar, Wasim Khan, Deep Makwana, Vikrant Nemade, Ankush Hutke, ‘A Protocol Based Packet Sniffer’, International Journal of Computer Science and Mobile Computing, Vol.4 Issue.3, March-2015, pg. 406-410
11. Ndatinya, Vivens & Xiao, Zhifeng & Manepalli, Vasudeva & Meng, Ke & Xiao, Yang. (2015). ‘Network forensics analysis using Wireshark’, International Journal of Security and Networks. 10. 91. 10.1504/IJSN.2015.070421.
12. Nedhal A. Ben-Eid, ‘Ethical Network Monitoring Using Wireshark and Colasoft Capsa as Sniffing Tools’, International Journal of Advanced Research in Computer and Communication Engineering Vol. 4, Issue 3, March 2015
13. Dr. Mahesh Kumar , Rakhi Yadav, ‘Tcp & Udp Packets Analysis Using Wireshark’, International Journal of Science, Engineering and Technology Research (IJSETR), Volume 4, Issue 7,2015, pp.2470-2474
14. Amanpreet Kaur, Monika Saluja, ‘Investigating TCP/IP, HTTP, ARP, ICMP Packets Using Wireshark’, International Journal of Emerging Technology and Advanced Engineering, 2014, pp.191-198.
15. Inderjit Kaur, Harkarandeep Kaur, Er. Gurjot Singh, ‘Analysing Various Packet Sniffing Tools’, International Journal of Electrical Electronics & Computer Science Engineering Volume 1, Issue 5, 2014
16. Jarmo Luomala, ‘A Tool for Generating Protocol Dissectors For Wireshark in Lua’, , Department of Computer Science and Engineering, University of Oulu, Oulu, Finland. Master’s thesis, 2013, 78 p.
17. Rupam , Atul Verma , Ankita Singh, ‘An Approach to Detect Packets Using Packet Sniffing’, International Journal of Computer Science & Engineering Survey (IJCSES) Vol.4, No.3, June 2013, pp.21-33.

18. Alias, Syazwina & Manickam, Selvakumar & Kadhum, Mohammad. (2013), 'A Study on Packet Capture Mechanisms in Real Time Network Traffic',pp.456-460. 10.1109/ACSAT.2013.95.
19. Mohammed Abdul Qadeer, Mohammad Zahid, 'Network Traffic Analysis and Intrusion Detection using Packet Sniffer', Second International Conference on Communication Software and Networks, 2010, pp.313-317
20. Usha Banerjee, Ashutosh Vashishtha, Mukul Saxena, 'Evaluation of the Capabilities of WireShark as a tool for Intrusion Detection', International Journal of Computer Applications (0975 – 8887) Volume 6– No.7,2010, pp.1-5

AUTHORS PROFILE

Dipayan Sinha is an Undergraduate Scholar pursuing Computer Science & Engineering in R.V. College of Engineering, Bangalore. He is passionate about Computer Networking and Machine Learning and has a keen interest on pursuing these in his higher studies. He has 2 publications under his name in the IoT and Mobile Agents domain. He is a rank holder in the department of Computer Science, RVCE. He is currently working under the mentorship of Dr. Shobha G.

Dr. Shobha G is a Professor in R.V. College of Engineering Bangalore. She has over 25 years of experience in teaching and over 14 years of experience in research. Her primary interests lie in Data Mining, Image Processing and Networking. She has 123 publications in International journals and conferences. She has also filed 4 patents and reviewed several books. She has published a chapter on 'Machine Learning Handbook of Statistics'. She was the former Head of Department of Computer Science Department of RVCE.