FlowChecker: A TCP Flow Analytical Framework for Network Measurements

Vinícius F. Garcia[1], Anderson M. da Rocha[1], Thales N. Tavares[1], Nilton C. B. da Silva[1], Leonardo da C. Marcuzzo[1]

[1]Federal University of Santa Maria - Post-graduate in Computer Science
Network Technologies Research Nucleus (NuPTeR)

{vfulber, amonteiro, tntavares, nbatista, lmarcuzzo}@inf.ufsm.br

Abstract. *Networks are in continuous growth and with that many economic interests emerges that can eventually results in network neutrality breaking. Specialized tools aim to detect this neutrality violation by realizing measurements and analyzing network metrics such throughput and errors amount. This work presents FlowChecker, a Python framework that can execute these measurements in TCP traffic and generate results for five different networks metrics. These results can be used as a background system for network neutrality breaking tools creation.*

1. Introduction

Network neutrality is conceptualized as the non-performance upgrade or downgrade for determined flows crossing Internet Service Providers (ISPs), in other words, it is the equal treatment for every flow independently of source, destiny, protocol or service. In the total neutral side, the network is just seen as a group of packages.

Many works [Zhang et al. 2009] [Tariq et al. 2009] [Kanuparthy and Dovrolis 2010] describe techniques for network neutrality breaking detection, but there are many limitations on the analysis methods. These solutions work with a specific application, source, destination or considering a single metric. This simple analysis can get an overall idea if network neutrality is or not being broken, but a complex process, considering many flows characteristics and metrics, may result in more correct conclusions and enable a deep check for the neutrality breaking motivations and consequences.

A possible solution to realize a deep analysis in a determined TCP traffic is identifying each flow crossing the system. Considering the quintuple formed by source IP, destiny IP, source port, destiny port and application as the flow definition [Dischinger et al. 2010], every different one crossing a determined system can have packets separated from the others. After flows identification process, it is possible to find the target direct flow and its reverse flow, carrying TCP packets responses and ACKs, to be analyzed.

This work presents a Python [Van Rossum et al. 2007] framework for TCP flows analysis called FlowChecker. With three modules, the framework is implemented aiming to receive a Comma-Separated Value (CSV) [Shafranovich 2005] file with information recovered by a PCAP. The first module identifies every flow present in the input file and creates individual CSV files containing every packets information for each flow. With this separated files, the operator must determine what flow and associated reverse flow will be passed to the second module that will recovery information about Round Trip Time (RTT), jitter, Packets Per Second (PPS), throughput and errors. Finally, every data generated by the second module can be sent to third module where informative charts are created using LaTeX [Lamport 1994].

This work is structured as follow: Section 2 presents network neutrality checker tools where some network analysis metrics considered in our study are used. Section 3 shows the overall framework architecture and each module operation. Section 4 presents a case study to verify the considered metrics behavior in a neutral and non-neutral scenario. Finally, Section 5 shows the conclusions and future works.

2. Related Works

There are works that use different approaches in flow separation to discover if occur or not network neutrality breaking. Many times these approaches are specific for a determined scenario or do not contemplate different network performance metrics at same time, such throughput, latency, jitter or packet loss rate.

The work developed by Mukarram Tarig [Tariq et al. 2009] presents a distributed measurement platform to determinate if an ISP induces the performance degradation for specific service classes. The proposed measurement is realized in a passive form where the user generated traffic is continuously accompanied by a monitor that sends information to a server periodically. In the server, the data flows are compared with information sent by other users. Finally, with a similarity factor scale between different clients of same service, the platform compares the performance to assess if there is or not network neutrality breaking.

Another work presents the Glasnost [Dischinger et al. 2010], a tool where final user connects yourself in the Glasnost web server to execute network tests, these tests generates flows containing application level data. In this way it is executed a measurement between the client traffic and the server traffic. The tool realizes the traffic differentiation using the flow types characterized by IP header (source and destiny IP), TCP header (source and destiny port) and by the packet payload.

The method exposed by [Kanuparthy and Dovrolis 2010] aims to detect if an ISP realizes traffic discrimination using packets loss rate and delay. The developed tool generates two flows, one with low priority traffic classification and another with normal priority. The proposed method realizes active measurements in clients machines in links of Measurement Lab (M-LAB) platform to detect any discrimination type based in traffic mechanism. In this way, the traffic differentiation can be detected using the queue management information (*e.g.*, Weighted Fair Queueing and Random Early Detection).

The system proposed in [Zhang et al. 2009] detects the traffic differentiation based in routing information, packet headers and application layer information. The system presented by the authors use an intelligent way selection to detect both the content differentiation and routing differentiation. The data obtained using the analysis is compared by the aggregate loss rates of different flows to deduce network neutrality breaking in a determined ISP.

The work presented in [Basso et al. 2011] shows a tool for distributed network measurements where an agent periodically monitors the latency and throughput of user connection and keeps the results in a centralized server. An important characteristic highlighted in the work is the continuous final user's connection monitoring.

The proposed framework, FlowChecker, does not realizes a neutrality analysis by itself, but provides a tool that recoveries common network metrics used in the related works. None of cited works used at same time RTT, jitter, PPS, throughput and errors amount for a deep correlation and behavior patterns definition. This work is a first step that makes possible to execute a complex analytical process to neutrality breaking detection in an easy way.

3. FlowChecker Architecture and Operation

The FlowChecker framework[7] consists of three operational modules that complements each other programmed as classes in Python language, in other words, the last executed module provides data for next one. A preprocessing stage is necessary for correct information achievement from PCAP and exportation to a CSV file, to do that we used the WireShark tool [Combs et al. 2007] to separate only the information that will be processed by FlowChecker (Time, Source, Destination, SrcPort, DstPort, Protocol, Frame, Ack, Bytes, Payload). Figure 1 presents the system data flow and the main modules are resumed below:

- Fragmentation: receives a CSV file and returns CSVs for each flow.
- Checker: receives a direct and reverse flow CSV file and returns dictionaries with selected metrics data.
- Chart: receives the Checker output and returns a file with LaTeX charts.

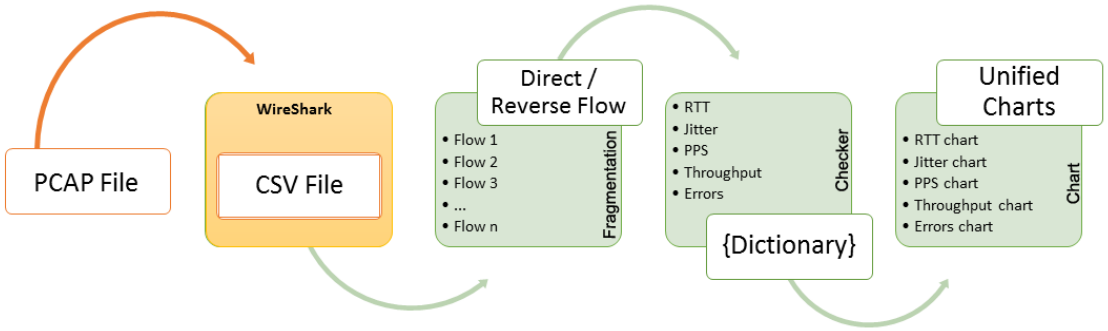

Figure 1. FlowChecker Data Flow

3.1. Fragmentation Module

The fragmentation module receives a CSV file containing ten fixed columns, these columns are used both in fragmentation module and in checker module:

- **Time:** relative time since the traffic start, the first packet presents time 0, used by checker module.
- **Source:** source IP, used by fragmentation module as part of flow definition.
- **Destination**: destination IP, used by fragmentation module as part of flow definition.
- **SrcPort:** source port, used by fragmentation module as part of flow definition.
- **DstPort:** destination port, used by fragmentation module as part of flow definition.
- **Protocol:** defines the packet protocol or the packet application if it is identified. Also used by fragmentation module as a part of flow definition.
- **Frame:** frame number, used to identify packets by checker module.
- **Ack:** carried by a packet, confirm another packet receiving.
- **Bytes:** packet size in bytes.

[7]**GitHub: https://github.com/ViniGarcia/FlowChecker**

- **Payload:** packet data field, it carries important information for checker module.

A flow is identified by the fragmentation module as a quintuple (Source, Destination, SrcPort, DstPort, and Protocol) [Dischinger et al. 2010]. Every packet belonging to a flow is inserted in a CSV, each CSV file has an individual ID representing the flow. Finally, an identification file is created relating every ID to its corresponding flow.

3.2. Checker Module

The checker module receives a direct flow file, indicating which flow must be analyzed, and a reverse flow file for retrieve acks data. This module returns five network metrics as a Python dictionary:

- **RTT:** uses the Frame, Ack and Time data to calculate every received packet RTT.
- **Jitter:** uses the Frame and Time to calculate the delay variation.
- **PPS:** sort packets by Time and count second by second.
- **Throughput:** sort packets by Time and uses the Byte field to determine throughput second by second.
- **Errors:** sort packets by Time and verifies the Payload field searching for retransmissions, fast retransmission or out-of-orders packets counting them and grouping second by second.

Every metric has a specific method that returns a dictionary containing the requested data. For RTT and jitter the dictionary key is the frame number, for other metrics the key is the moment in seconds starting at 1.

3.3. Chart Module

The chart module generates vector images by using LaTeX [Lamport 1994] with TiKZ and PGFPlots [Tantau 2008] libraries, for that we selected a standard line chart that properly adjusts the axis scale independently of data limits. Each metric has a specific method for the chart generation and a general method which creates only one document with all charts. Figure 2 exemplifies the chart module output.

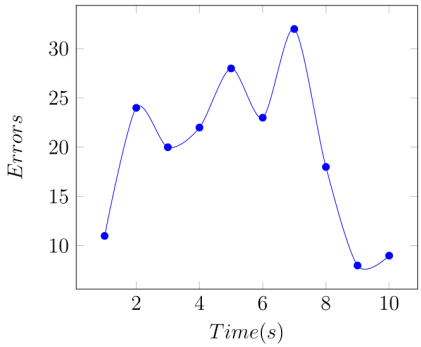

Figure 2. Errors Amount Chart

After the charts generation process, a tex file is created and must be manually processed by a LaTeX compiler, such MikTex [Schenk et al. 2006] or TexLive [Berry 2007], resulting in a PDF with charts images.

4. Metrics Behavior in a Case Study

To check the applicability of the chosen metrics we evaluated their behavior in a case study simulating a neutrality breaking situation. Our scenario consists in two hosts, first one as a client and other as a FTP server. Every packet crosses a router in the middle between client and server, two polices are applied to this router:

- Totally neutral router, all packets cross with no discrimination.
- FTP discrimination router, all FTP flows suffer with 20% dropping rate, on average.

The client and the server machine are Ubuntu virtual machines. A Debian virtual machine executing the Click Modular Router framework [Kohler et al. 2000] acts as router, the policies are directly programmed in the router Click code. The traffic is generated by the D-ITG application [Avallone et al. 2004] and least 7 seconds. The simulated FTP flows crosses the port 21 and each packet carries 512 Bytes. Figure 3 depicts the described scenario.

Figure 3. Test Scenario

We sniffed all traffic during the simulation using the Wireshark tool. Also, we turned the PCAP file in a CSV file and recovered the same FTP flow both in neutral router and non-neutral router test with the target fields to be computed by FlowChecker. Figures 4, 6, 8, 10, and 12 presents, respectively, RTT, jitter, PPS, throughput, and errors amount for the neutral router scenario, and Figures 5, 7, 9, 11, and 13 for the FTP discrimination router scenario.

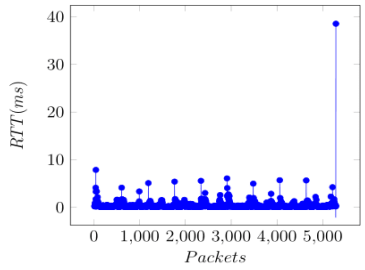

Figure 4. Neutral Router RTT

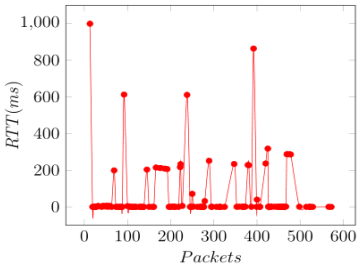

Figure 5. Non-neutral Router RTT

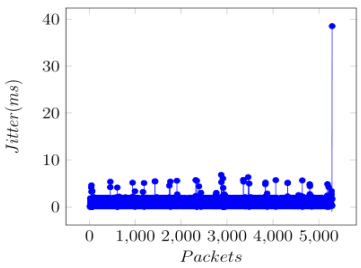

Figure 6. Neutral Router Jitter

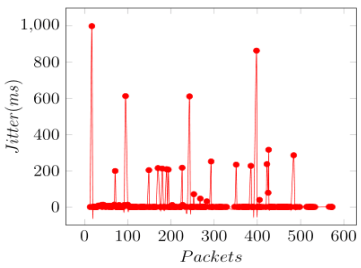

Figure 7. Non-neutral Router Jitter

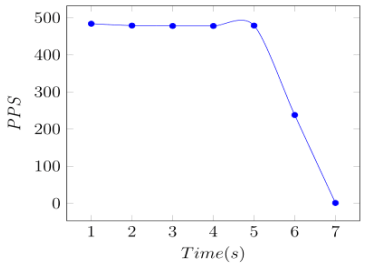

Figure 8. Neutral Router PPS

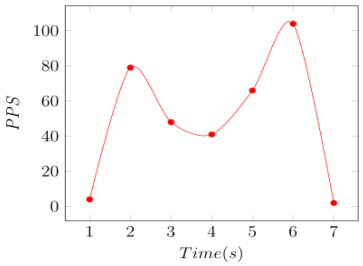

Figure 9. Non-neutral Router PPS

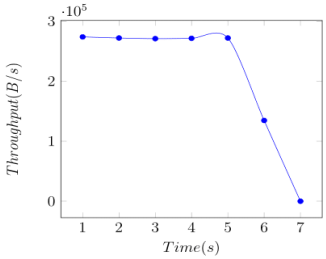

Figure 9. Neutral Router Throughput

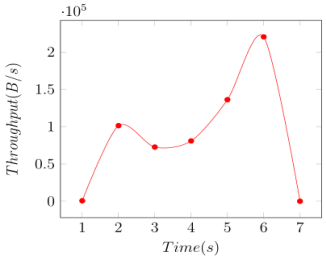

Figure 10. Non-neutral Router Throughput

When network neutrality breaking occurs, such FTP discrimination, different anomalies in the FTP flow can be verified. Big RTT and jitter measurements due the TCP ACK waiting, fewer packets successfully sent and low throughput rates, and a bigger number of retransmission end out of order errors are examples of these anomalies. Once these metrics can evidence these behaviors, they are appropriate to be correlated and analyzed for network neutrality breaking checking purpose.

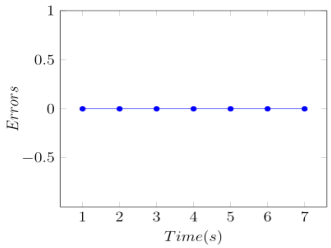

Figure 11. Neutral Router Errors

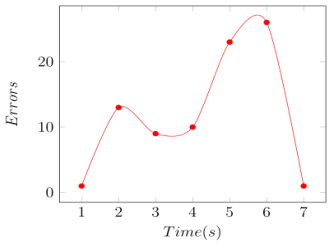

Figure 12. Non-neutral Router Errors

5. Conclusion

With the increasing networks popularization it is necessary ensure the correct and isonomic treatment of all packages crossing the ISPs. Programs and solutions try to identify network neutrality breaking events realizing network measurements that results in analytical metrics.

Many analytical metrics can be considered in network neutrality breaking checking process. Round Trip Time (RTT), jitter, Packets Per Second (PPS), throughput and errors are great candidates to be analyzed, but considering a single metric may not be sufficient to realize a reliable analysis.

This work presents a Python framework called FlowChecker that makes a full TCP flows traffic analysis returning important network metrics. This framework fundamentally aims to serve as a background part of network neutrality breaking detection tools and, as a secondary function, provides a visual easy way to see the analysis results presenting them in charts.

We used the framework to realize a case study where a FTP flow was evaluated in a non-discrimination scenario and when a induced drop rate occurs, breaking the network neutrality. The framework considered metrics presented significant variations when the two tested scenarios are compared.

In future works we will improve the framework to generate results for different granularity, so a single flow from a traffic PCAP can be analyzed in high time steps (minutes) or small ones (milliseconds). Finally, we will improve the chart module to create more than only line charts, making possible other results viewpoints.

References

Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-itg distributed internet traffic generator. In Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the, pages 316–317. IEEE.

Basso, S., Servetti, A., and De Martin, J. C. (2011). The network neutrality bot architecture: a preliminary approach for self-monitoring of internet access qos. In Computers and Communications (ISCC), 2011 IEEE Symposium on, pages 1131–1136. IEEE.

Berry, K. (2007). The tex live guide. PDF document. Leggibile con texdoc texlive nella distribuzione TeX Live.

Combs, G. et al. (2007). Wireshark. Web page: https://www.wireshark.org/.

Dischinger, M., Marcon, M., Guha, S., Gummadi, P. K., Mahajan, R., and Saroiu, S. (2010). Glasnost: Enabling end users to detect traffic differentiation. In NSDI, pages 405–418.

Kanuparthy, P. and Dovrolis, C. (2010). Diffprobe: detecting isp service discrimination. In INFOCOM, 2010 Proceedings IEEE, pages 1–9. IEEE.

Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kaashoek, M. F. (2000). The click modular router. ACM Transactions on Computer Systems (TOCS), 18(3):263–297.

Lamport, L. (1994). LATEX: a document preparation system: user's guide and reference manual. Addison-wesley.

Schenk, C. et al. (2006). Miktex.

Shafranovich, Y. (2005). Common format and mime type for comma-separated values (csv) files.

Tantau, T. (2008). The tikz and pgf packages.

Tariq, M. B., Motiwala, M., Feamster, N., and Ammar, M. (2009). Detecting network neutrality violations with causal inference. In Proceedings of the 5th international conference on Emerging networking experiments and technologies, pages 289–300. ACM.

Van Rossum, G. et al. (2007). Python programming language. In USENIX Annual Technical Conference, volume 41, page 36.

Zhang, Y., Mao, Z. M., and Zhang, M. (2009). Detecting traffic differentiation in backbone isps with netpolice. In Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference, pages 103–115. ACM.