Retina: An open-source tool for flexible analysis of RTC traffic

Retina: An Open-Source Tool For Flexible Analysis of RTC Traffic

Gianluca Perna[a], Dena Markudova[a], Martino Trevisan[a,*], Paolo Garza[a], Michela Meo[a], Maurizio M. Munafò[a]

[a]*Politecnico di Torino, Italy*

Abstract

Retina is an open-source command-line tool that produces rich and complex statistics from real-time communication (RTC) traffic. Starting from raw packet captures, it creates summaries of observed streams with flexible statistics and tracks the evolution of the stream over time. *Retina* is modular and highly configurable, providing the ability to configure output statistics, temporal resolution as well as many other parameters. Furthermore, if the packet captures are accompanied by application logs, it can reconcile the data and enrich its output with application and QoE- related statistics.
Retina helps troubleshoot RTC applications and enables the use of Machine Learning models for traffic classification and Quality of Experience assessment. We believe *Retina* can be extremely useful for researchers studying RTC traffic and network professionals interested in effective traffic analysis.

Keywords: Network Traffic Monitoring, Real-Time Communications

1. Context and Motivation

In recent years, the proliferation of broadband Internet access and mobile networks has spurred the adoption of Real-Time Communication (RTC) applications that allow people to communicate via voice and video. They are now essential for both leisure and business, helping people to reach friends and relatives and enabling remote working. The importance of RTC was particularly evident during the COVID-19 pandemic, when social distancing and lockdown measures adopted to curb the outbreak forced millions of people to communicate exclusively through RTC platforms. This led to a global increase of RTC traffic by more than 200% [1, 2]. It is therefore of utmost importance that researchers and practitioners are supported with tools to analyze RTC traffic and gain insights into the operation of RTC applications.

In this paper we present *Retina*, an easy-to-use command-line tool that extracts advanced network statistics for RTC sessions found in packet captures. It also generates graphical output with various charts and visualizations of the statistics for easy analysis. *Retina* focuses on the Real-Time Protocol (RTP) [3] protocol used in most RTC applications [4], with its encrypted version SRTP (which however leaves the packet headers in clear). *Retina* goes deeper than general tools in understanding RTC traffic. Starting from a capture, *Retina* searches for RTC traffic, identifies streams and outputs more than 130 statistics on packet characteristics, such as timing and size, and tracks the evolution of the stream over time bins of a chosen duration. It is highly configurable, and the user can customize the output statistics as well as a number of other parameters. *Retina* can enrich its output by merging the information available in the RTC application logs to provide the ground truth required for many classification problems.

Retina is open-source and available to the research community and network practitioners.[1] We believe it can be useful for traffic monitoring, and we have successfully used it for data processing and feature extraction to feed Machine Learning (ML) algorithms in the context of RTC-aware network management.

1.1. Literature Review

Several tools already perform in-depth traffic analysis, and packet dissectors such as *Wireshark*[2] (and its command-line version *Tshark*) are the first resources for network troubleshooting. Flow monitoring is also commonly used to analyze traffic summaries [5], and NetFlow [6] is the de facto standard for collecting and processing flow records. Sophisticated network meters also expose application-level statistics using Deep-Packet Inspection on Layer-7 protocols. Tstat [7], for example, provides global statistics on RTP streams, while nProbe [8] offers a VoIP plugin as a closed-source commercial product. In contrast to these works, *Retina* provides comprehensive statistics both per time unit and per flow. It specializes in RTC traffic and detects numerous RTC applications, including some that modify the RTP protocol. It also offers a wide range of parameters for personalized log creation.

*Corresponding author
Email address: martino.trevisan@polito.it (Martino Trevisan)

[1] https://github.com/GianlucaPoliTo/Retina
[2] https://www.wireshark.org/

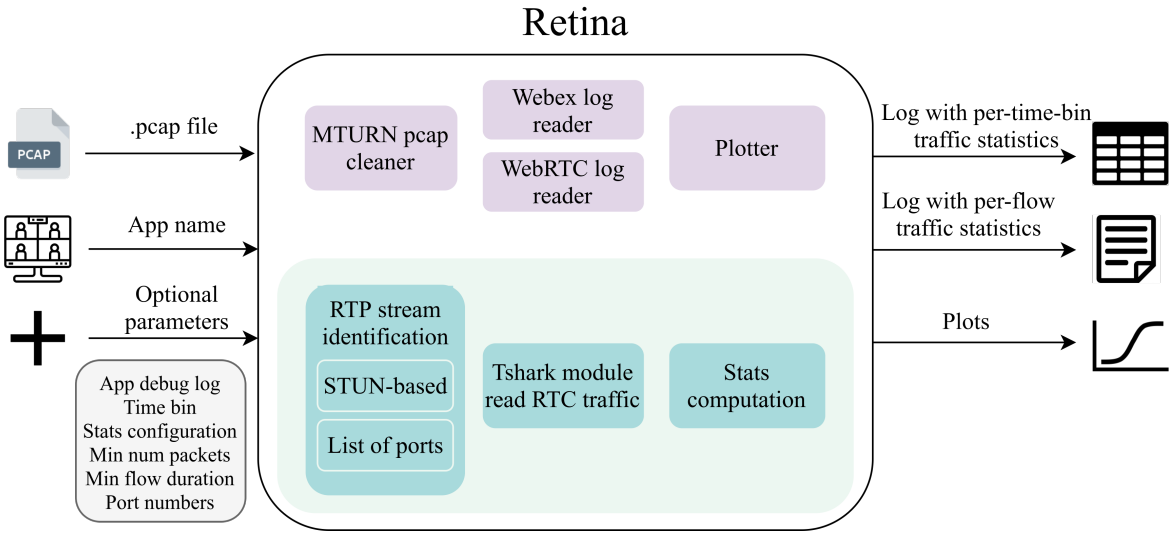

Figure 1: *Retina* architecture.

2. System overview

In this section, we describe *Retina*'s operation. As input, *Retina* takes one or more packet captures as well as optional configuration parameters. It processes the traffic and outputs the desired output in various forms. We depict its overall architecture in Figure 1. *Retina* is written in Python and depends on *Tshark* and a number of modules that can be installed via the package manager `pip`. We also provide a dockerized version to allow the use as a standalone container.[3]

2.1. Inputs and Configuration

Retina requires the user to specify one or more captures in PCAP format, the most common format used in many traffic capture softwares (*Wireshark*, *TCPdump*, etc.). *Retina* can also process an entire directory by searching for all captures in it. If it finds more than one, *Retina* uses multiprocessing to process multiple files at once. The number of processes is a configurable parameter.

For some RTC applications, the user can provide application log files that *Retina* uses to calculate additional statistics and enrich the output. The application logs typically contain details about the media sessions, including the Source Identifiers of the RTP streams, the type of media (audio, video, or screen sharing), the video resolution, the number of frames per second, etc. When available, *Retina* uses this additional information to provide finer-grained per-second statistics – e.g., media type, video resolution or concealment events at the codec level. Currently, *Retina* supports log files of: (i) Cisco Webex [4], which logs second-by-second details for each RTP stream, and (ii) Google Chrome when collecting WebRTC debugging logs with WebRTC [5] browser-based RTC services.[6]

In *Retina*, the user can customize a variety of parameters. All are optional, with carefully set default values. *Retina* has personalized features for many RTC applications, which can be enabled by specifying the name of the RTC application whose traffic is included in the capture as an input parameter. While it supports all applications that use RTP at their core, we have tested it extensively for Webex, Jitsi, Zoom, and Microsoft Teams. *Retina* accepts threshold parameters, such as the minimum number of packets or the minimum duration of a stream for it to be considered valid. The user can also control the statistics computed at each time bin (see Section 2.3) and can ask *Retina* to create (interactive) graphs. The full list of parameters can be found in the documentation, while in the rest of the paper we will only mention the most important ones.

2.2. System core

We show the overall architecture of *Retina* in Figure 1, with the middle rectangle indicating the building blocks at its core. At the bottom, in blue, there are the basic functionalities, while, on the top, in purple, the optional modules. We also show a sample command line at the top of Table 1.

The basic functionalities of *Retina* analyze the raw packets contained in the input PCAP captures and gather statistics, organized in tables per stream and per time-bin. For example, consider a PCAP capture collected at a user side, containing RTP traffic from a two-party call consisting of 4 RTP streams (outgoing and incoming audio and video). Setting a time bin duration of 1 s, *Retina*

[3] The dockerized version is available at: `https://hub.docker.com/r/gianlucapolito/retina`

[4] `https://www.webex.com/`

[5] `https://webrtc.org/`

[6] These logs can be obtained by creating and downloading a dump at `chrome://webrtc-internals`

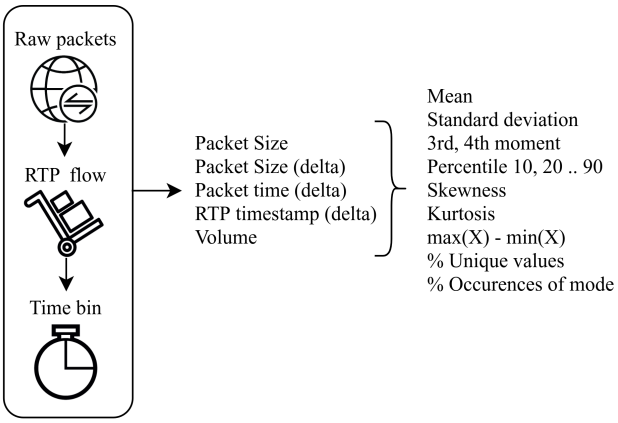

Figure 2: Aggregation process and some of the statistics computed by *Retina*.

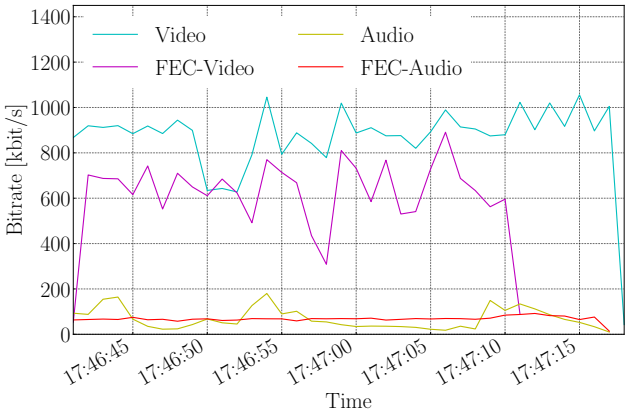

Figure 3: Example plot of the stream bitrate in a call.

maintains a table where, for each of the 4 streams and for each second, it accumulates several statistics. Given a packet characteristic, such as packet size or interarrival time, *Retina* calculates several statistical indicators, such as mean, median, third and fourth moments, or percentiles. We report the list of packet features and available statistics in Figure 2, which summarizes the whole process of statistics extraction. The user can configure the duration of the time bin for this aggregation of packets, which is 1 s by default. The duration of the time bin directly affects the number of packets used to compute the statistics, and should therefore be varied judiciously. For example, in 1 s of audio, 50 packets are sent, while, in 1 s of HD video, more than 200. Clearly, if the time window is 200 ms for audio, no significant features can be computed, while this would be fine for video.

To identify RTP streams in traffic, *Retina* relies internally on *Tshark*, the command-line version of *Wireshark*. This step is not straightforward, as RTP packets often appear in a UDP flow along with other protocols. In fact, many applications use STUN [9] to establish the media session and/or TURN [10] to relay the streams if no direct connection between peers is possible. In addition, it is common to use DTLS [11] interleaved among RTP packets to exchange control information such as encryption keys. *Retina* supports two methods for identifying RTP streams: (i) with a user-defined list of ports or (ii) by examining the STUN-initiated UDP flows. *Retina* attempts to decode the UDP payload as RTP and verifies that the protocol headers are compatible with RTP. We define an RTP stream using the combination of IP addresses and ports (the classic *tuple*) plus the RTP Synchronization Source Identifier (SSRC), which is used to multiplex multiple streams within a single UDP flow. For some RTC applications, we also use the RTP Payload Type (an RTP field that specifies the media codec). *Retina* maintains internal data structures to efficiently collect statistics for each RTP stream.

Retina has a number of optional modules that target RTC applications, for which we have implemented special support. First, the traffic of some popular RTC applications (Zoom and Microsoft Teams) needs to be preprocessed to become standard RTP traffic. This is because they use the RTP protocol in a non-standard form. Microsoft Teams encapsulates RTP in a proprietary version of TURN called MTURN, while Zoom adds its own undocumented header. To make *Retina* work for these RTC applications, we have created specific modules that can also be used as standalone command line tools. They can be found in a separate folder in the code repository.

Second, *Retina* can read and process the application log of (i) Webex and (ii) Google Chrome, as mentioned in Section 2.1. *Retina* can parse these logs and provide additional information about the RTP flows. If the application logs are available, we enrich the output logs from *Retina* with information such as the video resolution, employed codec, frames per second, jitter, codec concealment events, etc. We also provide a classification of media types into 7 classes, such as audio, FEC streams, 3 different qualities of video and screen sharing, for easier recognition. Note that the information in the application logs is particularly useful for training ML models, as it contains the necessary ground truth for many problems and *Retina* can match it with the network traffic.

Lastly, *Retina* includes a plotting engine based on the Matplotlib and Plotly libraries[7] to create both static and responsive graphs of all RTP streams. It draws the time-series of stream characteristics, such as bitrate or interarrival time, so that the user can easily get an overview of the traffic or debug an RTC application. It also draws several histograms for each stream to show the stream-wise distribution of packet characteristics (e.g. packet size). For an example graph, see Figure 3. Here we show the bitrate of 4 RTP streams present in a portion of a Webex call. The plotting engine also labels the time-series with their media type (audio, video, FEC etc.), if the information is provided (e.g. through an application log file).

[7] `https://matplotlib.org/`, `https://plotly.com/`

Command line: `./Retina.py -d capture.pcap -so webex -log webex.log`

Timestamp	Packet size (mean)	Packet size (std dev)	Bitrate (kbit/s)	Interarrival (max)	Packets/s	Frame width	Frame height	Frames/s
2021-06-08 14:32:11	1041.84	66.74	1163.93	0.043	143	480	270	30
2021-06-08 14:32:12	1080.72	100.75	1578.86	0.045	187	640	360	30
2021-06-08 14:32:13	1023.49	72.21	1023.49	0.045	128	640	360	30
2021-06-08 14:32:14	1076.80	52.91	1362.82	0.043	162	640	360	30
2021-06-08 14:32:15	1055.50	52.41	1410.08	0.044	171	640	360	30
2021-06-08 14:32:16	1074.62	62.71	1989.73	0.089	237	640	360	30
2021-06-08 14:32:17	1055.22	40.09	2588.59	0.033	314	640	360	30
2021-06-08 14:32:18	1057.73	51.67	1479.17	0.040	179	640	360	30

Table 1: Example command line and *Retina* log for an RTC stream. The last three columns are derived from the application logs.

2.3. Outputs

Retina produces a CSV file for each RTP stream found in the input capture, reporting the selected statistical features for each time bin. The logs contain different columns according to user preferences and additional stream information if the RTC application log is provided. We show an example output log in Table 1, along with the command line used to create it. Optionally, *Retina* creates a summary log file in which it reports stream-wise statistics. The file contains the most important information for each stream – i.e., the source and destination IP addresses and ports as well as general statistics such as the number of packets, duration, etc. Having per-stream information is useful for many applications that rely the analysis of flow/stream records for e.g., traffic accounting. Additionally, *Retina* provides traffic plots, which we described in Section 2.2.

Finally, *Retina* also provides a dashboard for analyzing RTC traffic through an interactive interface.[8] The dashboard requires an input `.pickle` file, which can be produced by passing one or more packet captures to *Retina* and specifying an argument for the plot. Here the user can see interactive plots of stream statistics and compare streams of interest.

3. System design assets

We have designed *Retina* following principles of scalability and modularity, so that it can be easily extended. It adopts a multiprocessing architecture, so when there are multiple PCAP files to process, it uses an independent process for each of them and stores separate output log files. These files can then be merged at the end of the processing. This also increases the robustness of the tool.

Retina is highly modular, with separate functions organized into logical modules for all the different operations. This also allows for extensibility, as a user can write new functionalities with minimal effort. For example, it is easy to support the application log of a new RTC application (e.g. Microsoft Teams), as it is only necessary to add a parser function and call it with an argument.

Retina can be used to analyze any kind of RTP traffic, and it is not limited to video conference applications. For example, we have successfully used *Retina* to gain insights into the operation of cloud gaming applications running over the browser [12]. Similarly, our parser for the Chrome WebRTC log works seamlessly for any type of browser-based application.

Finally, *Retina*, as described in Section 2.1, is highly configurable. The user can limit the statistics to be computed (potentially speeding up the computation), the desired time aggregation, and several internal parameters - e.g., the minimum length of an RTP stream for it to be considered - which are detailed in the README file.

4. Publications enabled by the software

Retina was first developed at the end of 2019, and within 2 years of its existence, it has already been a valuable asset for 4 scientific publications that target RTC traffic. *Retina* sits at the core of [13], where we used it to engineer features and extract the ground truth for an ML classifier that distinguishes media types. Using these features, we developed a Decision Tree classifier that performed with 97% accuracy. We further built on it in [14], to do data preprocessing and identify RTC streams in traffic. It also served for data characterization in [4], where we compare 13 different RTC applications. We also successfully employed it to study cloud gaming traffic, and it allowed us to understand the networking operation behind Google Stadia, GeForce NOW and PSNow in [12].

5. Limitations and Future work

While *Retina* supports most RTC applications, it still does not support those that do not use RTP (or a modified version of it), like *GoToMeeting* or *Telegram*. Moreover, it relies on the RTP headers, so if in a future protocol version these are encrypted, the tool will need major revisions.

As future work we aim to make *Retina* work in real-time and be able to support traffic at high speeds (e.g. 40 Gb/s links). We also want to introduce better support

[8] An online demonstrator of the dashboard is available at: `https://share.streamlit.io/gianlucapolito/retina-dashboard/main/dashboard.py`

for gaming traffic, cover different cloud gaming platforms, and output more gaming-specific ML features. We also plan to support *Retina* in the long run and follow the future developments of the underlying protocols such as RTP, STUN, and TURN, as well as tackle new protocols from novel providers.

6. Conclusion

This article presented *Retina*, a flexible command-line tool for extracting advanced statistics from network traffic of RTC applications. We provided a schematic description of all its features: the inputs, the system core and the outputs with examples. We also highlighted the design strengths of *Retina*, its modularity, scalability and configurability. We believe *Retina* can help both the scientific community in studying RTC applications and network administrators in troubleshooting RTC traffic. Our final goal is to make in-network devices regain visibility of RTC traffic and promote network management policies that favor this type of traffic. In particular, we designed it to be used directly for feature engineering of ML algorithms, since it can provide the ground truth for classification problems by processing the application log files.

7. Acknowledgements

This work has been supported by the SmartData@PoliTO center for BigData and Data Science and Cisco Systems Inc.

References

[1] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, G. Smaragdakis, The Lockdown Effect: Implications of the COVID-19 Pandemic on Internet Traffic, in: Proceedings of the ACM Internet Measurement Conference, IMC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1–18.

[2] T. Favale, F. Soro, M. Trevisan, I. Drago, M. Mellia, Campus traffic and e-Learning during COVID-19 pandemic, Computer Networks 176 (2020) 107290.

[3] R. Frederick, S. L. Casner, V. Jacobson, H. Schulzrinne, RTP: A Transport Protocol for Real-Time Applications, RFC 1889 (Jan. 1996). doi:10.17487/RFC1889.
URL https://rfc-editor.org/rfc/rfc1889.txt

[4] A. Nistico, D. Markudova, M. Trevisan, M. Meo, G. Carofiglio, A comparative study of RTC applications, in: 2020 IEEE International Symposium on Multimedia (ISM), IEEE, 2020, pp. 1–8.

[5] R. Hofstede, P. Čeleda, B. Trammell, I. Drago, R. Sadre, A. Sperotto, A. Pras, Flow monitoring explained: From packet capture to data analysis with netflow and ipfix, IEEE Communications Surveys & Tutorials 16 (4) (2014) 2037–2064.

[6] B. Claise, Cisco Systems NetFlow Services Export Version 9, RFC 3954 (Oct. 2004). doi:10.17487/RFC3954.
URL https://rfc-editor.org/rfc/rfc3954.txt

[7] M. Trevisan, A. Finamore, M. Mellia, M. Munafò, D. Rossi, Traffic analysis with off-the-shelf hardware: Challenges and lessons learned, IEEE Communications Magazine 55 (3) (2017) 163–169.

[8] L. Deri, N. SpA, nProbe: an open source netflow probe for gigabit networks, in: TERENA Networking Conference, 2003, pp. 1–4.

[9] J. Rosenberg, C. Huitema, R. Mahy, J. Weinberger, STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs), RFC 3489 (Mar. 2003). doi:10.17487/RFC3489.
URL https://rfc-editor.org/rfc/rfc3489.txt

[10] P. Matthews, J. Rosenberg, R. Mahy, Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN), RFC 5766 (Apr. 2010). doi:10.17487/RFC5766.
URL https://rfc-editor.org/rfc/rfc5766.txt

[11] E. Rescorla, N. Modadugu, Datagram Transport Layer Security, RFC 4347 (Apr. 2006). doi:10.17487/RFC4347.
URL https://rfc-editor.org/rfc/rfc4347.txt

[12] A. Di Domenico, G. Perna, M. Trevisan, L. Vassio, D. Giordano, A network analysis on cloud gaming: Stadia, GeForce Now and PSNow (2021). arXiv:2012.06774.

[13] G. Perna, D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafò, G. Carofiglio, Online Classification of RTC Traffic, in: 2021 IEEE 18th Annual Consumer Communications Networking Conference (CCNC), 2021, pp. 1–6. doi:10.1109/CCNC49032.2021.9369470.

[14] D. Markudova, M. Trevisan, P. Garza, M. Meo, M. M. Munafo, G. Carofiglio, What's my App?: ML-based classification of RTC applications, ACM SIGMETRICS Performance Evaluation Review 48 (4) (2021) 41–44.