Rationale, Design, and Implementation of the Network Neutrality Bot

Simone Basso* Antonio Servetti+ Juan Carlos De Martin*

simone.basso@polito.it, antonio.servetti@polito.it, demartin@polito.it

* NEXA Center for Internet & Society – DAUIN – Politecnico di Torino, Italy
+ DAUIN – Politecnico di Torino, Italy

Abstract

The "Network Neutrality Bot" (Neubot) is a software application that measures, in a distributed way, Internet access quality of service with a specific emphasis on detection of potential network neutrality violations (such as peer-to-peer traffic discrimination). It is based on a lightweight, open-source computer program that can be downloaded and installed by ordinary Internet users. The program performs background tests: the results are sent to a centralized server (or collection of servers), which publishes them, thus rebalancing, at least in part, the current deep information asymmetry between Internet Service Providers and users. The collected data will allow constant monitoring of the state of the Internet, enabling a deeper understanding of such crucial infrastructure, as well as a more reliable basis for discussing network neutrality policies.

1 Introduction

The issue of "network neutrality" has recently been one of the predominant topics in the worldwide debate on Internet policies. The basic question is whether network operators should be allowed to differentiate the Internet traffic that goes through their infrastructure or whether network neutrality should be explicitly safeguarded by the law, thereby enshrining what has been a characteristic of the Internet since its birth.

Indeed, the Internet was designed to be neutral with respect to different kinds of applications, senders and destinations. Such a design choice made very fast packet switching possible, and, at the same time, preserved a strong openness toward unforeseen uses of the Internet Protocol. The result has been an extraordinary outburst of innovation, as well as a level playing field for citizens, associations and companies worldwide.

However, with the advent of "deep packet inspection" and other classification technologies, fine-grained discrimination of Internet flows is now possible, and allows Internet Service Providers to block certain classes of flows and/or to differentiate the service according to the priority assigned to each class. In particular, packets that belong to low priority classes might be: (i) diverted on slower and/or more congested "virtual links", with traffic engineering technologies like MPLS (Multiprotocol Label Switching), (ii) dropped from the router queues in cases of congestion, (iii) scheduled for forwarding after higher priority packets.

The ability to block or slow down the traffic, which could be used to prevent the spreading of spam, viruses, botnets, and other malwares, could also be used by Internet Service Providers to implement very questionable policies. Apart from the so-called "Great Firewall of China" and other censorship efforts, which are beyond the scope of this paper, differentiating technologies could also be employed to throttle: (i) the "seeding" file-sharing traffic that flows out of ISPs networks (traffic considered "bad" because often providers are charged per-Megabyte for the traffic exchanged with their upstream Internet Providers), (ii) the file-sharing traffic generated

during rush hours (in an effort to avoid the collapse of underprovisioned access networks,) (iii) the traffic of some "over-the-top" (OTT), possibly free, services that compete with "managed services" that an ISP sells (for example Skype competing with ISP own Voice-over-IP solution.)

In particular, the conflict between managed and OTT services, e.g. YouTube, Skype, is becoming more and more relevant (particularly in the US). Providers (a) have started to offer additional managed services along with the Internet connection, such as television, video, and voice communication, and (b) often employ differentiating technologies, and other pratices such as *bandwidth caps*, to guarantee that there is always enough bandwidth to carry the managed services (while OTT services get the traditional "best effort" treatment.) The conflict between Internet companies and service providers is an ongoing legal battle concerning the power of the F.C.C. to impose some network neutrality principles on Internet Service Providers.

However one might question the relevance of the legal framework in light of the ability of Internet operators to directly control the basic infrastructure (including edge routers) and "control points" [1] [2] of relevance to the end-users' Internet traffic, as well as the high information asymmetry [3] [4] that characterizes the "market" under consideration.

For an informed debate on network neutrality, it is necessary to have the proper tools and methodologies for analyzing operators' policies and practices and any changes in their behavior due to relevant regulatory decisions. Thus, the Neubot [5] project will help to examine ISPs throttling of Internet traffic aimed at penalizing certain types of traffic depending on the protocol used and service offered. In addition the public availability of such data will help to inform the international debate on network neutrality providing real, per-host, network measurements.

2 Architecture

The architecture of Neubot consists of an open-source client application that volunteer end-users shall install on their computers and on a set of Neubot servers. The client application runs in the background and automatically performs a set of transmission tests between the Neubot computer and one or more test servers (client-server mode), and between the Neubot computer and other Neubots (peer-to-peer mode.) Periodically, test results are sent back to a central server (or a set of distributed servers.)

2.1 Client-server mode and component description

The diagram in Figure 1 shows the basic components and operations of the architecture when in client-server mode. Note that it is either possible to deploy the Coordinator, the TestNegotiator, the TestProvider, and the Collector into separate hosts, for scalability, as depicted in figure, or it is possible to install all of them into the same physical machine, for simplicity and maintainability.

The Coordinator keeps track of all the available servers, and provides each Client with the list of tests to be performed. Server-side, tests are implemented by (at least) one TestNegotiator and one (or more) TestProvider for each relevant protocol. The TestNegotiator negotiates the test parameters (such as time, type, and duration) with the connecting Client, while the TestProvider implements the server side of the test for a given protocol[1]. The Collector receives and collects the results of the tests. The Client periodically pulls and executes a list of tests.

When started, the Coordinator does not know the address of any TestNegotiator. However, there is a mechanism for TestNegotiators to register (at startup) and unregister (at shutdown)

[1]Note that the TestNegotiator and the TestProvider are separated components because it should be possible for a single TestNegotiator to negotiate the parameters for several TestProviders, installed on different hosts for scalability.

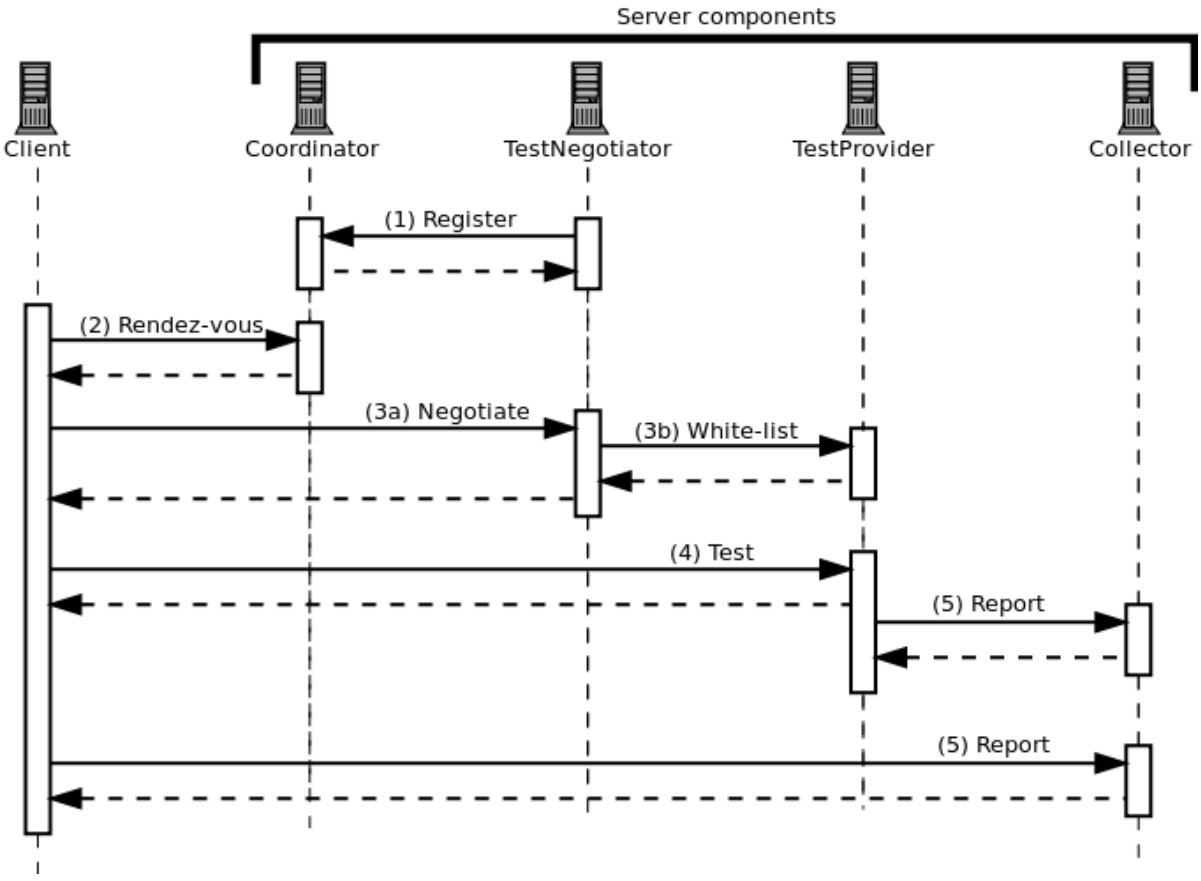

Figure 1: Interaction between the server and client components of the Neubot architecture, when the client is running in client-server mode

with the Coordinator (1). This way, when the Client connects (2), the Coordinator is able to return a list of suitable TestNegotiators, together with the address of (one of) the Collector(s)[2], and, possibly, information on available updates (only when the Client version is older than the Coordinator one.) The fact that the Coordinator populates the list of TestNegotiators to be returned allows the tests to be very flexible: different TestNegotiators could be returned depending on the circumstances.

Given the list, the Client chooses one TestNegotiator (possibly at random) and sends a test request (3a), proposing certain values for the test parameters. The TestNegotiator checks whether there is an idle TestProvider available for the Client. If all the TestProviders are busy, the response contains an error code, together with the number of seconds to wait before retrying the test request. Otherwise, the TestNegotiator generates an unique identifier for the test, and sends the TestProvider such unique identifier and the Client IP address (3b), which will be temporarily white-listed[3]. Then, the TestNegotiator sends the Client a response, containing the unique identifier, the TestProvider address, and the test parameters (with possibly some differences from the ones the client proposed, in case the incoming values were out of the acceptable range.)

Negotiated the parameters, there is the transmission test between the Client and the TestProvider (4). During the test, the Client and the TestProvider measure some specific performance metrics that depend on the employed protocol. The target protocols include, but are not limited to: (a) HTTP, the Hypertext Transfer Protocol, (b) BitTorrent, the protocol em-

[2]There is not a mechanism to learn the list of available Collectors. Such list is known in advance and is part of the Coordinator configuration.

[3]The TestProvider closes the connection at the beginning of the test unless the connecting IP address is in the white-list.

powering several popular peer-to-peer file sharing applications, (c) RTP, the IETF Real-Time Transport Protocol, used by many Internet-based multimedia applications, and (d) the Skype's proprietary peer-to-peer Voice-over-IP protocol. In addition, during the test, the Client and the TestProvider quantify the hosting computer load – measuring network, memory and CPU usage.

Once the tests are completed the Client and the TestProvider report the results to the Collector (5). The result set includes the unique identifier, the type, the IP address of the Client, the IP address of the TestProvider, the duration, the size, the measured performance metrics, the date, the time, and the measurements regarding the hosting computer load (that will allow to infer with some confidence whether the result has been significantly biased by the user activity.)

After the test, the Client goes to sleep for a certain amount of time, before repeating again the procedure explained above.

2.2 Peer-to-peer mode

The difference between client-server mode and peer-to-peer mode is that, in between each test, a Neubot client running in peer-to-peer mode behaves like a server, to give another client the chance to perform a peer-to-peer test. More in detail: the Neubot client enters in peer-to-peer mode, starts a TestNegotiator and a TestProvider, registers with the Coordinator, waits for the other client to negotiate and perform the test, reports the result of the test to the Collector, unregisters with the Coordinator, stops the TestNegotiator and the TestProvider, and, finally, leaves peer-to-peer mode.

3 Implementation

In this section we provide some more details regarding the (ongoing) implementation of Neubot. The program is deployed as a Python script and a library of Python classes (the Neubot Python Library), implementing both the client and the server. Depending on the command line options, the startup script might spawn the Graphical User Interface, start the Neubot client in the background, run a background process that hosts all the server-side components, or selectively start each server-side component as a standalone background process (allowing for different components to run on separate network hosts.)

The diagram in Figure 2 shows the components that are part of the Neubot Python Library. For simplicity, the diagram assumes that all the server-side components are running on the same process. Note that on the client-side there are one (or more) TestNegotiator and TestProviders, for the peer-to-peer mode.

In this section we discuss particularly the low level components that were not described in the previous section:

Event-based I/O This component allows to handle multiple concurrent small messages without consuming too much resources (in particular it consumes a small amount of memory because it does not need to spawn extra processes or threads, and it does not employ more than one CPU core.) This is relevant both on client-side (where the Neubot should have minimal impact) and on server-side (where we expect several incoming small messages from the clients.) In addition, this component allows to wrap sockets with Secure Socket Layer, that might be exploited to provide encryption (and possibly authentication) for the client-to-server communication.

HTTP This component is implemented on the top of the Event-based I/O component, and provides the interface to send and receive HTTP messages. It is employed in all the

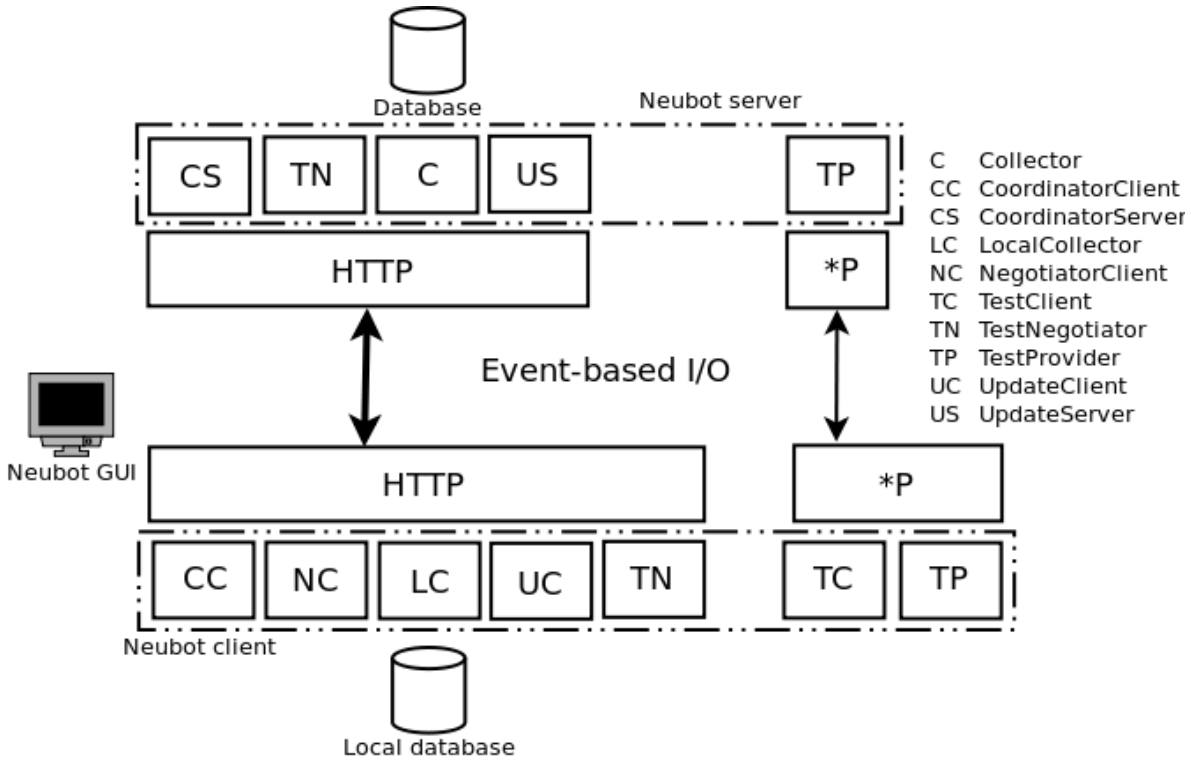

Figure 2: The software components of the Neubot client and of the Neubot server

communications between client and server components described in Figure 1. Indeed, the server-side components are implemented as RESTful web services, i.e. web services following the REST (Representational State Transfer) principles. By these principles, each sever component exports one or more resource through an URI, and for the client components it is possible to invoke the standard HTTP methods (such as GET, PUT, and POST) to change the state of such resources. For example: the communication (2) of Figure 1 is actually implemented by a `POST /rendez-vous` request, with attached a JSON that declares the type of tests the client would like to perform, and (unless there were errors) by a `200 Ok` response, with attached a JSON representing the list of available TestNegotiators. In addition, this component is also employed to implement all HTTP-based transmission tests.

***P** This component is just a place-holder representing all the other implemented protocols (not including HTTP.) Each protocol component is implemented on the top of the Event-based I/O component, and provides the interface to send and receive messages using that protocol. The tests are performed comparing different protocols when performing similar tasks. For example, to compare HTTP with BitTorrent, we force HTTP to transfer several fixed-size pieces of the same resource, employing the Range header.

UpdateClient (UC) This component downloads the updated version and installs it (it is disabled if the operating system provides a package management system.)

UpdateServer (US) This component is just a web-server that allows clients to download the most updated version.

Finally, there are the client-side components that were not explicitly mentioned in the previous section, because we were considering the client as a whole. They are: the CoordinatorClient (CC), that contacts the CoordinatorServer (2), the per-protocol NegotiatorClient, that negoti-

ates the test (3a), the TestClient, that implements the client-side of the transmission test (4), and the LocalCollector, then stores results locally and sends them to the Collector (5).

4 Related work

In literature, most of the related work regarding network neutrality tools is focused on testing only one specific kind of disruption. Notable is, for example, Diffprobe [8], that tries to infer whether there is protocol-dependent shaping.

It's also Worth mentioning: (i) the work of Weaver, et al. [9], that provides a rich set of heuristics to classify incoming RST segments, being able not only to identify the spoofed ones, but also to tell, with a certain degree of confidence, which device should have injected the segments; (ii) the Glasnost project [10], that provides an user-friendly Java applet that performs a differential test (BitTorrent vs. random data) trying to detect whether the user's ISP blocks BitTorrent traffic; (iii) the work of Zhang, et al. [11], that studied the aumount of traffic differentiation in the backbone, employing a tool working like a protocol-aware traceroute.

More general approaches, that are able to quantify a broad range of network neutrality violations, as in Neubot, are NANO and Grenouille.

The former one, NANO (Network Access Neutrality Observatory) [7], differently from Neubot employs passive measurement: The client continuously monitors user-generated traffic, and periodically sends throughput, round-trip-time, and other general performance metrics to the server, as well as ancillary information including the state of the hosting computer, the geographic location, the browser, and the operating system. Interestingly, the server relies on stratification to cluster the clients in strata where the difference in performance depends only on the fact that different ISPs have employed different policies.

The latter one, Grenouille [6], has an architecture similar to Neubot, but a slightly different goal: to measure ISP backbone congestion. Every 30 minutes, the client connects to a server standing near the edge of the ISP network, to avoid traversing (many) other ISPs networks. This way, the FTP upload, FTP download, and ping tests are not (much) biased by other ISPs, and hence it is possible to evaluate the average quality of service. To avoid overloading the servers, each client displaces tests in time by a random amount of seconds. Tests results are reported to a central server, unless the client finds that the user has consumed too much network resources during the test. The central server analyzes the results, producing daily and monthly, global and per-city charts.

5 Conclusion and Future Work

The value of the Neubot architecture lies in the fact that it works in a distributed way. Using a bottom-up methodology it collects and publishes data to counter the power of control by Internet operators over the infrastructure, rebalancing, at least in part, the current deep information asymmetry between Internet Service Providers and users.

Regarding the implementation, the software is currently in alpha stage. The development roadmap will have a first public release by the end of September, 2010. This first release will implement the architecture described in this paper, except the support for peer-to-peer measurement mode, and will perform HTTP transmission tests only. We will add more features in the following releases, and, before the end of January, 2011, Neubot will grow to include the support for: the BitTorrent transmission tests, the peer-to-peer measurement mode, the measurement of the hosting computer load. Further future developments could possibly include Skype and other VOIP testings and the release of Neubot for mobile devices.

Finally, notice that, as soon as the data collected by Neubot will become available, another relatively independent line of research will start, allowing constant monitoring of the state of the Internet and enabling a better understanding of network neutrality policies.

References

[1] Dame A., Guettler J.H., Leeson K., Schultz M., Jensen B.T., 2003, *Regulatory implications of the introduction of next generation networks and other new developments in electronic communications*, Study Report for the European Commission.

[2] Lessig L., 1999, *Code and Other Laws of Cyberspace*, Basic Books.

[3] Pindyck R.S., Rubinfield D.L., 2005, *Microeconomics, 6th edition*, Prentice Hall.

[4] Shapiro C., Varian H.R., 1998, *Information Rules: A Strategic Guide to the Network Economy*, Harvard Business School Press, Cambridge (MA).

[5] De Martin J.C., Glorioso A., 2008, *The Neubot project: A collaborative approach to measuring internet neutrality*, IEEE International Symposium on Technology and Society.

[6] Grenouille.com, `http://www.grenouille.com`

[7] Tariq M.B., Motiwala M., Feamster N., Ammar M., 2009, *Detecting network neutrality violations with causal inference*, Proceedings of the 5th international ACM conference on Emerging networking experiments and technologies.

[8] Kanuparthy P, Dovrolis C, 2010, *DiffProbe: Detecting ISP Service Discrimination*, Proceedings of International Conference on Computer Communications.

[9] Weaver N., Sommer R., Paxson V., 2009, *Detecting Forged TCP Reset Packets*, Proceedings of Network and Distributed System Security Symposium.

[10] Dischinger M., Mislove A., Haeberlen A., Gummadi K.P., 2008, *Detecting bittorrent blocking*, Proceedings of the 8th ACM Special Interest Group on Data Communications conference on Internet measurement.

[11] Zhang Y., Mao Z.M., Zhang M., 2009, *Detecting traffic differentiation in backbone ISPs with NetPolice*, Proceedings of the 9th ACM Special Interest Group on Data Communications conference on Internet measurement.