Implementing active probe measurement tools with libparistraceroute

Jordan Augé[1], Marc-Olivier Buob[1], Inês Ribeiro[2], and Timur Friedman[1]

[1] UPMC Sorbonne Universités, Paris, France `<first.last@lip6.fr>` **

[2] HSC, Levallois-Perret, France `<ines.ribeiro.da@gmail.com>`

Abstract. The distributed nature of the Internet makes it difficult to measure. Nonetheless, the active measurement community has produced a variety of techniques, ranging from simple pings to more complex traceroutes and available bandwidth estimations, that yield significant insights into the network's structure and characteristics. This article is based on the observation that many of the tools that implement these techniques take a common approach of querying the network through the sending and receiving of specially crafted probes at specific points in time. They principally differ in how they orchestrate the probes and how they interpret the responses. We propose a high level abstraction for network probing that allows concise expression of measurement algorithms, while still allowing fine-grained measurement control through lower level abstractions. These are implemented in a free and open source library called libparistraceroute. We illustrate the expressiveness of this framework for the implementation of complex measurement algorithms through the ease with which we build a new version of the improved traceroute tool, Paris Traceroute. This implementation is itself another layer of abstraction in the library, enabling future tools to readily incorporate its capabilities.

1 Introduction

The distributed nature of the Internet, as an interconnected set of independent networks, causes us to rely upon active measurements in order to understand its structure and characteristics. A wide spectrum of tools and techniques based on specially crafted probes have been developed to measure as diverse network elements as paths, delays, bandwidth and so on. Despite their obvious differences, they have much in common structurally, and differ essentially in the content and scheduling of probes, and the interpretation they make of elicited responses.

This paper describes a free open source framework that supports the development of active measurement tools. It consists of a modular C library called *libparistraceroute*[3] that implements abstractions and reusable components intended to relieve developers from much of the effort required to develop a new

** The authors are affiliated with LIP6 Computer Science Laboratory and LINCS (Laboratory of Information, Network and Communication Sciences)

[3] The libparistraceroute library is available at `http://code.google.com/p/paris-traceroute/`

tool. Built into the library is a set of heuristics based upon experience of how to make measurements robust in the face of network host misbehavior (such as mixing up byte ordering). The library also provides guidelines intended to encourage best practices and a scientific approach to measurements.

Although some probe creation and packet interpretation tools [1] and libraries [13,15] already exist, the novelty of our work lies in the structured, modular, and powerful formalism and abstractions that the library presents to a developer, and in the fact that it makes a flexible packet probing tool available in the highly useful form of a C library. As free open source software under a BSD-like license, this library is a contribution to the Internet measurement community as a whole, for unlimited incorporation within other tools.

The libparistraceroute library has its origins in the *Paris Traceroute* tool [17]. A major additional contribution of this work is that Paris Traceroute's ability to trace multiple load-balanced paths is for the first time provided by a library, rather than a stand-alone program. This is especially important for those who wish to make use of the stochastic multipath discovery algorithm (MDA) [16], which is not simple to program. We use Paris Traceroute as an example throughout the article, while also pointing out the more general applicability of the library.

This paper is organized as follows. Sec. 2 describes existing libraries and systems that support the design of new measurement algorithms. Sec. 3 provides a high level overview of the library. Secs 4 to 6 present the different layers provided by the library and are illustrated with the Paris Traceroute implementation. Sec. 7 follows with a discussion on how the library could be applicable to other types of measurements. Finally, Sec. 8 summarizes our conclusions, provides pointers to available code and resources related to the library, and indicates directions for future work.

2 Related work

Several projects partially meet some of the library's objectives. Scapy [1] is the work closest to libparistraceroute. As a packet manipulation program written in Python, it is able to forge custom probe packets and decode the corresponding replies. Nice features of Scapy include: (i) its flexibility, allowing a wide range of packets to be forged, and (ii) its recognition that probe replies can be read in different ways, with a corresponding treatment of their interpretation as a separate step. libparistraceroute offers these same features and improves on them. Beyond what Scapy provides, libparistraceroute also proposes higher level abstractions that can readily be used by measurement algorithms. For example, it allows convenient description of the high-level notion of a flow. Burns et al. [3] acknowledge that Scapy is slow due to its implementation in Python and its design. In principle, an independent C library should offer better performance (though we have yet to conduct side-by-side tests), as well as the greatest ease of reuse for developers working across a variety of programming languages and operating systems.

There are existing well-used C libraries for sending packets (libdnet [13]) and sniffing them (libpcap [15]). However these libraries do not expose an API as convenient as libparistraceroute's for writing active measurement algorithms.

NIMI [11], Scriptroute [14], and Scamper [9], are platforms that each make available a collection of tools for conducting Internet measurements, but they not provide a generic framework for conceiving new tools. One of the NIMI tools was zing, a packet sending and receiving program that offered fine grained control over packet sizes and timing. To our knowledge, zing was never released as free open source code, and NIMI is no longer available.

We of course also refer to the previous work on Paris Traceroute [17]. In refactoring the code for this tool, we saw the limitations of a monolithic stand-alone program and the opportunity to write a highly flexible library design to implement any active measurement tool.

3 Library overview

As Fig. 1 depicts, the library offers three layers of abstraction, allowing developers to write OS- and protocol-independent algorithms at the highest layer, while still providing them with fine-grained control over the lower probe and network layers, should they need this. A notable feature of the library is its implementation of common features required by most tools, such as probe creation, packet scheduling, probe/reply matching, etc., allowing developers to focus on their core competency of algorithm formulation. We adopted a modular design with no external dependencies, which allows easy removal of unnecessary parts, should the library be shipped on constrained devices such as sensor networks.

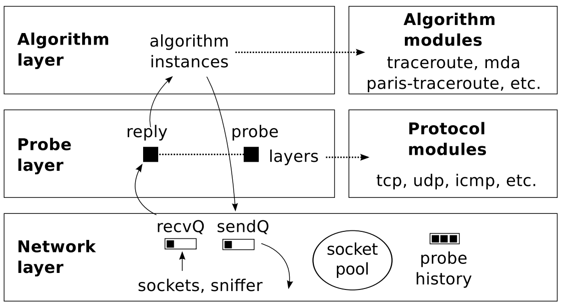

Fig. 1. Library architecture

The algorithm layer orchestrates the crafting and sending of probes and interprets the raw replies. This layer provides a set of modular measurement algorithms including traceroute. Algorithm modules express at a high level the mechanics of a measurement technique.

The probe layer exposes probes and associated replies to the upper layer thanks to a key/value abstraction. By focusing on field semantics, it allows algorithms to be expressed in a high-level, generic, and protocol independent fashion. The interface offers both *structural constraints*, which define the way the probe packets are crafted, and *temporal constraints*, which guide their scheduling.

The network layer aims at matching sent packets with their corresponding replies. As with the probe layer, the network layer is aware of a variety of protocols and matches packets accordingly. Knowledge of the ways in which hosts can send corrupted replies is encoded in this layer. This includes problems with byte ordering and NAT boxes replying with NATed addresses rather than public addresses. The network layer flags such replies so that they can be handled seamlessly by the higher layers. This layers also manages duplicated packets and timeouts. It separates higher layers from OS-dependent considerations.

4 Algorithm layer

Algorithms are the highest level abstraction in the library's framework. The algorithm layer allows one to design measurement tools and to interpret collected measurement data. The abstractions offered by the library ensure that algorithms are portable, and independent of architectures and operating systems. Algorithms can also run other algorithms so that complex functionalities can be implemented from simpler bricks.

Implementing an algorithm consists in simply plugging a new module into the library, providing handlers to react to a set of events: an algorithm beginning or terminating, reception of a probe reply, a timeout, etc. An algorithm can also generate its own events that enrich this basic set. For instance, the Paris Traceroute module might raise an event to inform its caller that it has finished probing at a given TTL.

Reusing and tuning algorithms A notable strength of libparistraceroute is its modularity. One can design an algorithm by composing other algorithms. This makes it easier to create variations on algorithms. For example, a hop-by-hop traceroute algorithm can be simply transformed into a faster version that simultaneously sends probes towards every hop along a path, allowing for measurement of transitory paths during a routing change. The library's modularity also allows one to easily adapt an algorithm to run over a different lower layer protocol.

Algorithms accept a probe skeleton as a parameter, which serves as a template for building the final probe packet. A skeleton can easily be created from command line options, and be further adapted to be passed to the set of called algorithms.

In classic tools, it is generally not easy to apply slight variations that have not been previously imagined by their author. And options are often restricted based on assumptions regarding the protocols that will be used or the possible values of packet fields. It is often the case that only a subset of the range of options is proposed. Thanks to libparistraceroute's separation of layers, it is straightforward to run a traceroute using any protocol providing that functions to tag and match packets are available. The same holds for Paris Traceroute as soon as a notion of flow is defined.

Command line tools Individual instances of measurement tools are generally manipulated through a command line interface, and output is directed to the standard output. libparistraceroute proposes helpers both for parsing options passed in the command line and to display the content on screen or into standard data formats. Raw probes and replies are stored in the library and future work will add the ability to export them to standard formats such as pcap. Separating the algorithms from the user interface makes them directly callable in various contexts and should ease their integration both in large scale measurement platforms requiring a large number of parallel measurements (see Sec. 7) and into third-party solutions such as FastMapping [4] and D-Track [5].

5 Probe layer

This layer is responsible for translating high level structural and temporal constraints into the information needed by the network layer. Sec. 5.1 details how the layer manages these constraints. Secs. 5.2 and 5.3 describe additional aspects of the layer. Then Sec. 5.4 explains how the layer handles probe replies.

5.1 Managing probe content

To forge appropriate packets, the probe interface requires a *protocol pattern* indicating which protocols are used and how they are nested, (for instance IPv4/TCP/IPv4 in Fig. 2). For each protocol, there is a dedicated *protocol module*, that indicates each of its field's offset, size, and type[4]. Currently libparistraceroute supports IPv4, IPv6, UDP, and ICMP, and others are being added. The collection can be easily extended by adding custom protocol modules.

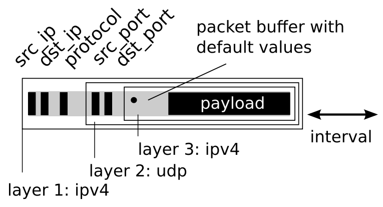

Fig. 2. Construction of a packet buffer thanks to the probe interface (black boxes represent the fields set by the developer).

[4] Note that more than one protocol may share a given key (for example, "src_port" in TCP and UDP).

The probe layer can then set up an appropriate buffer, initialized with some convenient or meaningful default values (by default, a IPv4/UDP packet with null payload and default header values that should fit most situations, as in Scapy). Of course, libparistraceroute never overrides a value set by the developer.

The developer can then edit the packet, focusing on only the fields of interest and the payload. Fig. 2 illustrates how a packet buffer is edited, with the probe layer providing ease of access to the packet buffer. Fields are set by a helper function to which we pass the packet, optionally the starting layer (1 by default), and a set of key/value pairs identifying the field names and the values they will take. Specifying the starting layer is only relevant if a field name is ambiguous (the source IP might both concern the 1st and 3rd layers in Fig. 2's example). Unless otherwise specified, libparistraceroute adjusts fields such as length, checksum, and protocol at the lower layer to complete a well-formed packet.

5.2 Probe groups and generators

We provide a facility that enables developers to generate a group of coherent and consistent probe packets. Such probe generators support two kind of constraints:

Temporal constraints allow one to specify how to schedule each consecutive probe produced by the generator (for instance randomly, exponentially distributed with a given mean, etc.). They use an *interval* field, which stores an interval of time used by the network layer to delay the emission of the next probes; this controls scheduling and allow the creation of various bursts.

Structural constraints generalize the notion of a probe skeleton. Instead of setting fields to a given value, as a probe skeleton does, these constraints only limit the set of possible values (for example $ttl < 10$, or values must be distinct, etc.).

Generated probes implicitly belongs to the same probe group as other probes originating from the same algorithm. Probe groups can also be manually specified. Such grouping has the advantages that replies are also associated together, and that it is possible to manipulate groups more easily and get information information about them, as presented later.

5.3 Metafields

A *metafield* is a custom field defined among several fields, typically to define high level-notions. For example, one may need to generate probes crafted such as to belong to the same flow. This can be done by defining a constraint which involves a "flow" metafield. Depending on the protocol pattern specified, such metafield might have different interpretations in terms of associated fields, and across several protocol headers[5]. Metafields thus improve the genericity and readability of

[5] For example, in IPv4/TCP and IPv4/UDP, the flow metafield consists of a fields from both headers: source and destination IP addresses and ports, and the transport protocol; with IPv4/ICMP echo requests, ports are replaced by the ICMP code and checksum. Finally IPv6 remains to be studied by it might involve both IP addresses together with the flow label.

algorithms, by relieving their implementation from network protocol considerations. Another use of metafields is to refer to a set of unused bits in the packet header in which one could encode information. This could be used to avoid storing state information due to local storage constraints, and collected in ICMP answers, or to communicate information between a sender and a receiver.

5.4 Extracting information from replies

As for probe crafting (Sec. 5.1), replies that come from the network are made accessible through the mechanism of key/value pairs, similarly to protocol dissectors found in many network tools. Some fields might also be added to interpret the contents of a probe (indicate whether a checksum is correct or not for instance). It is also possible to define more complex operations over a group of probes. An example of more sophisticated information that can be provided through the key/value mechanism would be a rate computation. Finally, the library provides a syntax for testing whether probes and replies match a given pattern. It is used internally by the library itself for protocol recognition and for validating the possibility to use a metafield for a given probe.

6 Network layer

6.1 Scheduling packets

Once a probe has been crafted, the network layer is responsible to schedule it and to listen for matching answers from the network. The scheduler uses the *interval* field set by the probe layer (see Sec. 5.2) to decide upon each packet's sending time. The packet is also at this stage attributed a socket among the available ones in the pool.

We are planning to add more sophisticated functionalities, such as the ability to rate limit so as to avoid triggering ICMP rate limiting or misinterpretation of probe traffic as a DDoS attack.

6.2 Matching probes and replies

After transmitting a packet, the library listens to the network for possible matching replies. As an illustration, a response to an IPv4 packets can either be a packet with the same protocol structure flowing in the return direction, an ICMP packet with the header of the original packet repeated in the payload, or even not exist (timeout). It can then inform the responsible layer accordingly (a traceroute program is for example interested in the ICMP TTL expired generated by routers along the path).

6.3 Tagging packets

The probe/reply mechanism works on the principle that replies be matchable with the original probes. In the case of an ICMP reply, this is done by examining the probe header that it encapsulates: it must be unique from all other probes that are in flight, or whose replies are in flight. We call the unique header signature a *tag*. Traceroute is an example of a program that tags its probes by varying the destination port number. Since this has unintended effects when traversing per-flow load balancing routers, Paris Traceroute tags packets using header fields other than the five-tuple that defines a flow [17].

The library manages tags through a "tag" metafield. In tagging probes, it avoids manipulating fields that have been defined by the developer, set as non-modifiable, or that are known to change in transit. The default tagging method is drawn from Paris Traceroute, but different choices of fields are possible. In the case of UDP, for example, the tag is encoded in the checksum field by writing it in the payload and swapping it with the computed checksum value (this way the checksum remains valid). A pool of values, depending on the number of bits that are available in header fields to encode tags, is managed by the library to ensure that two probes in flight don't get attributed the same tag (including an additional delay to detect duplicates). This is done in a manner that is transparent to the developer.

6.4 Allowing for erroneous replies

Researchers conducting large-scale Internet measurements often encounter malformed replies due to implementation errors in routers, or misbehaving hosts. Common errors include NAT boxes that fail to translate the private address space destination address of a probe header encapsulated in an ICMP packet, endianness mismatch in some packet fields, etc. Going beyond the functionality of classic tools, libparistraceroute includes mechanisms to relax the matching process so as to detect such cases. A corollary to the existence of these errors is that tags need to be robust enough to counter them. For instance, if we know that a given header field is susceptible to a reversal of byte ordering, we must take care to avoid having two probes in flight that could be confused in the event of such reversal.

7 Applicability and future work

We have reimplemented Paris Traceroute and MDA [16] using our framework (other tools are being worked upon, for example exploiting burst creation, the sender/receiver pattern, etc.). Thanks to the expressiveness of libparistraceroute the resulting code is quite concise (compared to other implementations), OS and protocol independent. The code highlights that in MDA, the main difficulty resides in choosing the flows to send to the different hops to accurately discover the network topology.

The design of the library reflects our wish to promote its adoption by heterogeneous large scale measurement infrastructures, such as Ark [8], DIMES [12], iPlane [10], and TopHat [2]. Such platforms have to support a large number of measurement agents acting in parallel and need an efficient and modular library such as libparistraceroute. Indeed, libparistraceroute goes further than approaches such as Scamper [9] by offering a greater modularity in the design of measurement tools, and as such could prove complementary.

An example of where this modularity is useful is in coordinated distributed measurements. Algorithms such as Tracetree [7] and DoubleTree [6] are not easily implemented using the standard traceroute tool. They adopt advanced measurement strategies that require probing from a specified starting point, probing in a reverse direction rather than always probing forward, and a set of stopping conditions. The usual way to deploy new algorithms such as these is to write tools from scratch. Looking ahead to future measurement algorithms that might require even finer granularity of control and more parallel measurement instances, we believe that libtraceroute provides a solution. The library is built around a single-threaded event-based architecture, and manages a pool of reusable sockets in order to cope with the performance requirements of such parallel measurements. We don't expect the proposed high-level interfaces to constitute bottlenecks in those situations.

Future development of the library will focus on improving the expressiveness of the framework. One particular aspect is an improved management of the timing and its accuracy. We are also looking at developing a standard interface for encoding additional information within unused packet fields, in the same way as the tag, to facilitate the design of tools involving coordination between several agents. While payload is sufficient for communication between a client and a server, this becomes necessary when the receiver of an ICMP error is different from the sender, in the case of spoofing for example.

8 Conclusion

This paper has presented libparistraceroute, an open-source library written in C under a BSD-like license and designed to support easy implementation of active measurement tools. The current Paris Traceroute tool uses this library.

To the best of our knowledge, libparistraceroute is the first library to offer a high level of expressiveness, permitting a developer to only focus on the way that a measurement algorithm works. By doing so, libparistraceroute supports concise, clear, generic, and reusable code. It also ensures that measurements are consistent and that best practices are respected. The library is designed to be easily extended; one may easily develop support for new network protocols and measurement techniques. We hope this library will be of use to people who wish to incorporate the capabilities of Paris Traceroute into their tools, as well as those who wish to develop other types of tools.

The authors wish to thank Xavier Cuvelier and Brice Augustin for their contributions in developing the original Paris Traceroute tool, from which this library emerged.

References

1. P. Biondi. Scapy, a powerful interactive packet manipulation program.
2. Thomas Bourgeau, Jordan Augé, and Timur Friedman. Tophat: supporting experiments through measurement infrastructure federation. In *Proceedings of TridentCom'2010*, Berlin, Germany, 18-20 May 2010.
3. B. Burns, D. Killion, J.S. Granick, S. Manzuik, and P. Guersch. *Security Power Tools*. O'Reilly Series. O'Reilly, 2007.
4. Ítalo Cunha, Renata Teixeira, and Christophe Diot. Measuring and characterizing end-to-end route dynamics in the presence of load balancing. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 235–244, Berlin, Heidelberg, 2011. Springer-Verlag.
5. Italo Cunha, Renata Teixeira, Darryl Veitch, and Christophe Diot. Predicting and tracking internet path changes. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 122–133, New York, NY, USA, 2011. ACM.
6. Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. Deployment of an algorithm for large-scale topology discovery. *IEEE Journal on Selected Areas in Communications, Special Issue on Sampling the Internet*, 24(12):2210–2220, December 2006.
7. Ramesh Govindan and Hongsuda Tangmunarunkit. Heuristics for internet map discovery, 2000.
8. Y. Hyun. Archipelago measurement infrastructure.
9. Matthew Luckie. Scamper: a scalable and extensible packet prober for active measurement of the internet. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 239–245, New York, NY, USA, 2010. ACM.
10. Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iplane: an information plane for distributed services. In *Proceedings of the 7th symposium on Operating systems design and implementation*, OSDI '06, pages 367–380, Berkeley, CA, USA, 2006. USENIX Association.
11. Vern Paxson, Andrew Adams, and Matt Mathis. Experiences with nimi, 2000.
12. Yuval Shavitt and Eran Shir. Dimes: let the internet measure itself. *SIGCOMM Comput. Commun. Rev.*, 35(5):71–74, October 2005.
13. Dug Song. The libdnet low-level network library.
14. Neil Spring, David Wetherall, and Tom Anderson. Scriptroute: a public internet measurement facility. In *Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, pages 17–17, Berkeley, CA, USA, 2003. USENIX Association.
15. C. Leres V. Jacobson and S. McCanne. libpcap: packet capture library, June 1994.
16. Darryl Veitch, Brice Augustin, Renata Teixeira, and Timur Friedman. Failure control in multipath route tracing. In *Proc. IEEE Infocom*, 2009.
17. Fabien Viger, Brice Augustin, Xavier Cuvellier, Clémence Magnien, Matthieu Latapy, Timur Friedman, and Renata Teixeira. Detection, understanding, and prevention of traceroute measurement artifacts. *Comput. Netw.*, 52(5):998–1018, April 2008.