libparistraceroute: a framework to support probe-based network measurement tools

Jordan Augé, Marc-Olivier Buob,
Timur Friedman *
UPMC Sorbonne Universités
Paris, France
fistname.lastname@lip6.fr

Ines Ribeiro †
HSC
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 takes as its basis 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. Many tools and techniques have been developed to measure features as diverse as paths, delays, and bandwidth. A large portion of these take a common approach of sending specially crafted probes that are intended to elicit responses from the network. They mainly differ in the probes they create and send, and in the interpretation they give to the corresponding replies from the network, but much development work is duplicated.

This paper proposes a framework to support the development of active measurement tools. It consists of a modular C library called *libparistraceroute*[1] that implements abstractions and reusable components intended to relieve developers from much of the effort required to develop a new 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 formalism and the abstractions that the library presents to a developer, and in making 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 it offers Paris Traceroute's ability to trace multiple load-balanced paths for the first time as part of a library, rather than as 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 go into more detail on the levels of abstraction, namely the probe

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

†This work was completed while the author was at UPMC Sorbonne Universités

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

layer (Sec. 4), the network layer (Sec. 5), and the algorithm layer (Sec. 6). Sec. 6 is illustrated with the Paris Traceroute implementation as a proof of concept of the efficiency with which the framework can implement measurement tools. 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

Active measurement tools share several common capabilities, such as the crafting of probe packets and sniffing for and interpreting responses. Several existing projects partially fit the need for a framework to support these activities.

Scapy [1] is the work closest to libparistraceroute. 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 extreme flexibility, allowing packets of all sorts 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. Whereas Scapy enables one to send crafted packets of any kind and to retrieve the corresponding reply, libparistraceroute also provides higher level abstractions that can readily be used by measurement algorithms. For example, it offers a convenient way to describe 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, a 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 offer the probe management layer that makes libparistraceroute convenient for writing active measurement algorithms, not to mention the other higher layers that libparistraceroute offers.

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.

The final element of related work is Paris Traceroute [17]. In refactoring the code for this tool, we saw the limitations of a monolithic stand-alone program and the occasion to write a highly flexible library design to implement any active measurement tool.

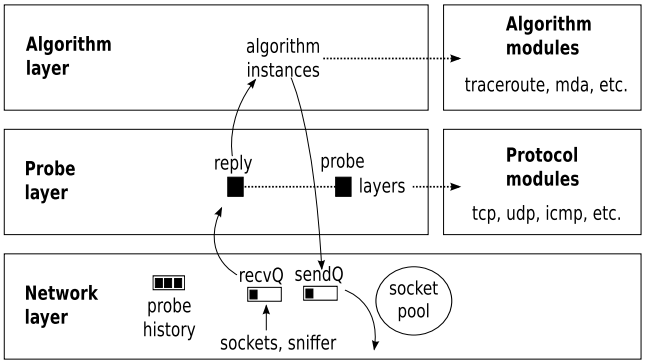

Figure 1: Library architecture

3. LIBRARY OVERVIEW

We have designed libparistraceroute to support developers who wish to write measurement algorithms. The library offers three layers of abstraction, allowing them 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.

As Fig. 1 depicts, the library provides three layers of abstraction:

Algorithm layer. Algorithms orchestrate the crafting and sending of probes and interpret the raw replies. Tools needing ready-made algorithms, such as pings, traceroutes, or available bandwidth estimation, can call them at this layer. Algorithms can also be constructed from other algorithms in a modular and extensible fashion. For instance, a traceroute consists of a succession of calls to a query-response algorithm.

Probe layer. Probe packets and their replies are the means by which algorithms learn about the network. This layer provides a language for describing how packets should be sent and how their replies should be interpreted. Developers can express *structural constraints*, which determine packet header content, and *temporal constraints*, which guide the scheduling of packets. This layer also assists the developer in managing probe replies, through pattern matching, protocol dissectors, helpers for interpretation, and decoders for information encoded into probe packets. The probe layer consists of modules for protocols at several levels in the network

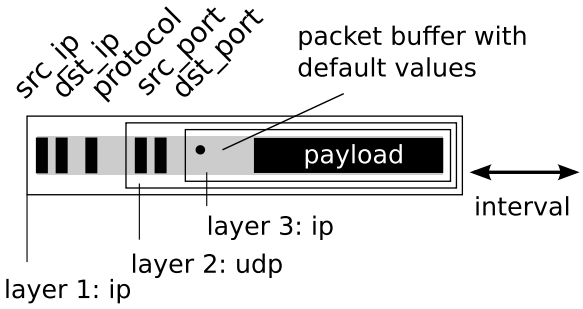

Figure 2: Construction of a packet buffer thanks to the probe interface (black boxes represent the fields set by the developer).

stack and is extensible to new protocols.

Network layer. Probe packets sent to the network are matched with replies, duplicates, or timeouts by the network layer. 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 at 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. Socket interaction and operating-system-dependent aspects of communication are handled in this layer.

4. PROBE LAYER

We start our discussion of the abstraction layers in the middle, as the probe layer is at the core of libparistraceroute. This layer is responsible for translating high level structural and temporal constraints into the information needed by the network layer. Sec. 4.1 details how the layer manages these constraints. Secs. 4.3 and 4.2 describe additional aspects of the layer. Then Sec. 4.4 explains how the layer handles probe replies.

4.1 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 IP/TCP/IP in Fig. 2); for each related protocol, a protocol description listing the *fields* that it provides; for each field, which values must be set.

A developer is free to use any protocol encapsulation (IPv4/UDP by default) and to edit any field in the packet. As explained for Scapy [1], it is not for the library to decide what might have meaning for the developer.

Once the protocol pattern is defined, the probe layer sets up an adequate buffer containing the required structures and initializes it with some convenient or meaningful default values. For example, libparistraceroute by default fills a IPv4/UDP packet with a null payload and with default header values that should fit most of the situations, just as Scapy does. Adding a DNS layer would automatically set the UDP destination port to 53. Of course, libparistraceroute never overrides a value set by the developer.

The developer can then edit the packet, focussing 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 (much like a database view). 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 (for instance "ip_src" might concern both the 1st and the 3rd layers in Fig. 2's example). Once the fields have been set, libparistraceroute adjusts fields such as length, checksum, and protocol at the lower layer to complete a well-formed packet.

For each protocol, there is a dedicated *protocol module*, that indicates each of its field's offset, size, and type. Note that more than one protocol may share a given key (for example, "src_port" in TCP and UDP). Currently libparistraceroute supports IPv4, UDP, and ICMP, and others are being added. The collection can be easily extended by adding custom protocol modules.

To control probe packet timing, an *interval* field is associated with each packet. It stores an interval of time used by the network layer to delay the emission of the next probe (see Sec. 4.2).

4.2 Probe groups and generators

We provide a facility for developers to work on groups of packets. Each group is treated as a coherent unit upon which the scheduler can enforce constraints. The library will ensure that the replies to a group of probes also form a group. The developer is presented with a hierarchy of probes and replies that is entirely managed by the library. All operations on a group are propagated to its members.

One way to define a group is through an extension of the key-value interface for probe manipulation that allows a developer to express constraints and distributions in addition to fixed values. Accordingly, a developer can define a set of possible probe packets, and a number of packets from the set can be generated on demand. Examples of constraints are that a given value should be less than x, or that a set of values should all be different. It is possible to ask for the interval field to be random, or exponentially distributed with a given mean, which allows for the creation of packet bursts with given characteristics. Groups are also implicitly

created for all probes sent by a given algorithm.

4.3 Metafields

A *metafield* is a generalization of the notion of a field, allowing a developer to refer at a high level to a group of fields. A typical example of the need for a metafield is when referring to a flow identifier. In TCP/IPv4, this consists of a five-tuple of fields from both of the protocol headers: source and destination IP addresses and ports, and the transport protocol. The flow identifier is different for ICMP/IPv4 and for UDP/IPv6 and for other protocol combinations. Defining a "flow" metafield allows one to define algorithms independently of the particular underlying protocols, improving code modularity and readibility.

A metafield is defined by a unique string identifier, a protocol pattern specifying the contexts in which it applies, and the fields that are involved. For greater convenience, libparistraceroute also provides helpers to compute sequential valid values for a metafield according to the structural constraints defined for a given probe group.

4.4 Extracting information from replies

As for probe crafting (Sec. 4.1), replies that come from the network are made accessible through the mechanism of key/value pairs. For example, a traceroute algorithm will seek to obtain the source IP address of a returned ICMP packet. To do this, the algorithm would query the reply for the value of the `src_ip` field.

This mechanism is similar to protocol dissectors found in many tools. We extend them by adding new keys that allow one to interpret the contents of a probe. For example, we provide a field that indicates whether the checksum is correct or not. 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 and subprotocol recognition and for validating the existence of a metafield for a given probe.

5. NETWORK LAYER

Once a probe has been crafted, it falls under the control of the network layer, which schedules its sending and listens for matching answers from the network.

5.1 Scheduling packets

At the probe layer, the timing specifications for probes or groups of probes (rate, interval distributions, etc.) have been encoded into the interval field. At the network layer, a scheduler uses this field 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.

The scheduler is currently relatively simple. We are planning to add more sophisticated functionality, such as the ability to rate limit so as to avoid triggering ICMP rate limiting or misinterpretation of probe traffic as a DDoS attack.

5.2 Matching probes and replies

After transmitting a packet, the library listens to the network for possible matching replies. The sender will systematically be notified of any probe/reply pair, or of any probe timeout while waiting for a reply. Matching is dictated by the various protocol modules, which know which replies to expect.

For example, an IPv4/UDP probe can either trigger a response from the destination, or raise an error on its way. A response will be a packet with the same protocol structure flowing in the return direction, which is examined by the relevant protocol modules. Errors triggered while a packet is in transit are generally communicated to the sender through the ICMP protocol with the header of the original packet repeated in the payload. Traceroute intentionally uses the error mechanism to trigger TTL expired messages from routers, thus revealing their interfaces.

5.3 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. For matching to work, a probe's header 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.

5.4 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, and libparistraceroute does this.

6. ALGORITHM LAYER

This section describes how to design an algorithm to run in libparistraceroute. It goes on to describe how we implemented Paris Traceroute using the library. It also presents the support offered by the library for adding a command line interface, and explains how it is designed for modification of the behavior of existing algorithms.

6.1 Presentation

Algorithms are the highest level abstraction in the library's framework. The algorithm layer offers developers the possibility to develop 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 that provides a set of handler function to react to a set of events. The main events are algorithm starting and termination, probe reply reception or a timeout, or the arrival of information from another algorithm that has been called. It is possible to enrich this basic set to allow for more interactivity; for instance, a traceroute module might want to inform its callers that it has finished exploring hops at a given TTL by raising a specific event.

6.2 Paris Traceroute implementation

We have implemented the Paris Traceroute mechanisms, including the Multipath Discovery Algorithm (MDA) [16], using functions we have described in previous sections. This example illustrates the efficiency of the framework to express measurements tools.

Thanks to the support provided by the probe layer, the resulting code is quite concise (compared to other implementations) and it is architecture and OS independent. The code highlights that the most complex part of the realization of MDA resides in the right choice of the flows to send to the different hops so as to accurately measure links. In addition, the use of the flow metafield relieves us from a protocol-dependent realization, since any protocols that possess a notion of flow can be used.

While Paris Traceroute mechanisms have not yet been proposed for IPv6 networks, where the status of load balancing is not known, we believe the required changes would simply consist in extending the definition of a flow in this context and ensuring adequate packet tagging.

6.3 Command line tools

Individual instances of measurement tools are generally manipulated through a command line interface, and output is directed to the standard output. We propose helpers in the library to build such interfaces with minimal effort. The intention here is to accept the full range of possible parameters in a consistent manner. Naming used is the library is as close as possible to that found in well known tools, and is made easy through the characterization of all manipulated objects (protocols, fields, algorithms, options) through strings. For Paris Traceroute, it was nevertheless important to propose options exactly similar to the ones provided by the regular traceroute tools to allow for seamless replacement of one tool by another. This was achieved through the use of aliases.

The library proposes a set of output functions to display the content of the data structures containing measurements 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.

6.4 Reusing and tuning algorithms

Separating the algorithms from the user interface (e.g., option parsing, outputs, etc.) makes them directly callable in various contexts (in a C program, through a GUI, in a script binding, etc.). This should ease their integration into 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].

Sometimes, it is desirable to make a few changes in the behavior of an algorithm or in the structure of the probes it sends. Classic tools do not in general make it easy to develop variations that have not been previously imagined by their author, and they tend to do so in non-consistent ways.

It is important to notice that an algorithm in fact only really manipulates a subset of the packet headers. In a classic traceroute for example, only the TTL is instrumented to elicit responses from the different hops of a path towards a destination. Bounding the TTL values or specifying the way a path is explored are options and thus act as parameters. Such a design might help in proposing a fast traceroute able to simultaneously send probes towards every hop in a path, allowing the measurement of transitory paths during a routing change for instance.

As for the content of probes, the library allows a developer to pass a probe skeleton to an algorithm. The skeleton serves as a template for building the final packet. A model can easily be created from the command line options, and be further adapted to be passed to the set of called algorithms.

Classic tools often mix those options with those involving the protocols used or the value of their fields; and it is often the case that only a subset of them is proposed. Thanks to such a separation, 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.

7. APPLICABILITY AND FUTURE WORK

The design of the library reflects our wish to promote its adoption by large scale measurement infrastructures, such as Ark [8], DIMES [12], iPlane [10], and TopHat [2]. These infrastructures are characterized by a heterogeneity of languages and architectures, and the need for performance to support large numbers of measurement agents acting in parallel on a given host. In this context, it is important to separate raw measurements from their interpretation and to be able to easily collect the generated data. Going further than approaches such as Scamper [9], in which the basic unit that a researcher can manipulate is a measurement tool, libparistraceroute offers greater modularity in the design of measurement tasks.

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 measuremnt 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, a library written in C under a BSD-like licence 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 and 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 tool.

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.

9. 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.