Doubtless in Seattle: Exploring the Internet Delay Space

Philipp M. Eittenberger, Marcel Großmann, Udo R. Krieger
Faculty of Information Systems and Applied Computer Science
Otto-Friedrich University,
Bamberg, Germany
Email: philipp.eittenberger@uni-bamberg.de
marcel.grossmann@uni-bamberg.de

***Abstract*—In this paper we present the first distributed measurement campaign conducted by the P2P framework Seattle. Seattle is a distributed PlanetLab like network testbed that easily enables everyone to participate. The contributions of the paper are the following: First, we conduct a thorough analysis of the prevalent conditions in the Seattle network and compare them to reference measurements obtained in a controlled test environment. We use the results to evaluate the usability of the testbed for distributed measurement campaigns and characterize the key parameters relevant to measurements. Second, we present our implementation of a coordinated, tournament-based all-pairs-ping program, which is used to obtain several latency matrices between Seattle test nodes. Additionally, we use the obtained results to present a simple case study that investigates the relation between geographic distance and latency of Internet paths, which is a key element for the geographic location of Internet hosts.**

I. Introduction

The demand for large scale, globally distributed Internet measurement testbeds is still strong in the measurement community. Despite the existence of PlanetLab [1], there are objections regarding the extent to which it reflects the true conditions of the Internet [12]. A new alternative could remedy the problem, the cloud computing framework Seattle. In this study we are investigating its suitability for distributed measurement campaigns and present the first measurement study conducted by Seattle. But why does the Internet measurement community need such a distributed framework at all? For instance, there is still a struggle for optimal available bandwidth estimation techniques. In order to assess the performance and to compare the quality of such tools, it is vital to evaluate them under true-to-life conditions at large scale. That means in principle, one needs to test these tools under the typical conditions they would normally encounter in the Internet. As another example, many studies investigate and try to improve the performance of Vivaldi [4]. The latter is a network coordinate system that tries to embed nodes in a synthetic coordinate system to predict the latency between the hosts. However, a recent study of Steiner and Biersack [13] shows that in practice the basic statement of Vivaldi that embedding Internet hosts into an Euclidean space works well does not hold. A lot of studies evaluate such tools in a controlled test infrastructure or in particular parts of a research network yielding promising results, but this does not genuinely reflect the real conditions of the Internet. Instead, one needs a globally distributed Internet testbed to obtain a thorough large scale evaluation. Apart from bandwidth measurements or network coordinate systems, other metrics like packet loss, i.e. network tomography, delay measurements or geolocation need distributed testbeds to assess and compare the performance of the proposed mechanisms too.

Nowadays, PlanetLab is the prevalent testbed being used. It consists of a dedicated, monitored and managed infrastructure, which is mainly connected to global research and education networks. Thereby, two problems arise: On the one hand, the high bandwidth of research networks and the availability of the managed infrastructure does not reflect the reality of the Internet, where the majority of the accesses consists of DSL links and unmanaged end hosts. The second pitfall is given by the non-trivial prerequisites under which one can use PlanetLab. It cannot be easily accessed because individuals are not allowed to join PlanetLab directly and one's institution needs to be part of the PlanetLab consortium. This means that one provides a dedicated infrastructure (at least two servers) and spends a non negligible amount of time for the maintenance of the infrastructure. Apart from PlanetLab, few other alternatives are available. One aspirant is given by the Splay project [8], but at the time of the writing of this paper (May 2011) it is only available as an application running on the PlanetLab infrastructure. Emulab [15] is another network testbed that allows its users to build arbitrary network topologies inside a single data center. In the following experiments, we use Emulab to evaluate the Seattle platform in a controlled environment. But as stated, Emulab resides in a single data center and does not aim to represent the conditions of the Internet.

Seattle is a relatively new P2P framework that allows the comparison of such methods under true-to-life conditions in the Internet. The following studies already use Seattle as a platform for their experiments: Tutschku *et al.* [14] present a study that utilizes hosts of the Seattle network as relay nodes to investigate the influence of network virtualization on multimedia applications. Samuel *et al.* [10] use Seattle to present a prototype of a sandbox that allows the safe execution of untrusted code.

The paper is organized as follows: We introduce the Seattle framework in Section II. Subsequently, we investigate the

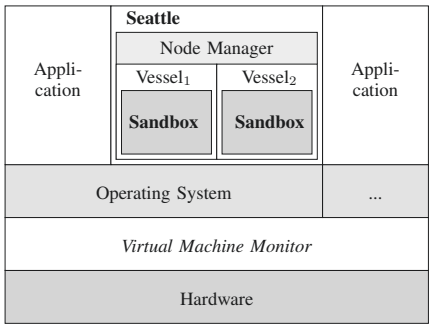

Figure 1. System Stack

conditions encountered when performing measurement campaigns by Seattle in Section III. Section IV presents an implementation of a coordinated, tournament based all-pairs-ping program that was used to capture several latency matrices between Seattle test nodes. Furthermore, we use the obtained results to present a very simple but efficient method to predict the distance of an Internet path by its latency. Finally, we conclude the paper in Section V.

II. Seattle

Cappos *et al.* [3] introduced Seattle in 2009 as an educational research platform for cloud computing. Seattle runs on a wide variety of different operating systems and architectures, e.g. Windows, Mac, Linux and BSD platforms; mobile devices are also supported. To write programs for Seattle, an API is provided that consists of a subset of the Python programming language. The Seattle programming API provides low-level operations and maintains program portability using an abstraction layer. The API consists of five categories (file, network, timer, locking and miscellaneous) to provide the limited functionality that a Seattle program can use. It is a very important fact to stress that every Internet user can install Seattle onto his computer and, thereby, participate and contribute a part of the computer's resources to the Seattle network.

The foundation of Seattle's architecture is a sandbox that guarantees security and resource control for an individual program, i.e. the sandbox limits resource usage and prevents actions that could compromise the safety of the host system. Sandboxed, isolated programs are controlled by the node manager's state about the program; both, the sandboxed program and the relevant node manager's state, are called in the terminology of Cappos *et al. vessel*. Consequentially, each vessel consists of the program along with its sandbox and its corresponding state controlled by the node manager (see Figure 1). For every running installation of Seattle the user receives resources, i.e. vessels, on ten other computers. For example, if one installs Seattle on ten running computers, one receives control over 100 vessels. Inside each remote vessel the user can deploy and execute his code. It is important to realize that all user code deployed in a Seattle vessel is executed inside a sandbox.

Seattle has certain characteristics, which distinguish it from other distributed testbeds like PlanetLab. All programs for Seattle must be written with the reduced Python subset of the Seattle API. This fact limits the usability of the platform such that it is not possible to just plug in and test any given application. Furthermore, every vessel has certain limitations. As already mentioned, each program runs in a resource restricted sandbox and one of the limitations is that the maximum bandwidth for every standard vessel is limited to 10 KBytes per second. In contrast to the PlanetLab, where each node runs a single operating system with dedicated hardware, Seattle vessels run as a normal user process on the host operating system (see Figure 1). Thereby, as each vessel is resource limited, timing delays encountered by scheduling are increasing, since the process scheduler of the host operating system could interrupt the process at any time and put it asleep to keep the process under CPU quota. Another drawback of the framework regarding Internet measurements is the strong encapsulation of the socket API due to security considerations. In practice that means, one can neither manipulate protocol headers nor send ICMP messages, since the socket API is pretty much transparent to the user. As a result there is no possibility to perform traceroute measurements. Seattle would be much more attractive for Internet measurement practitioners if the strict socket encapsulation would be relaxed, or if, at least, the emission of ICMP messages would be allowed.

However, the ease of access and use means that it is a framework that enables almost every technology savvy Internet user to participate and contribute. As of May 2011 a total number of 4123 distinct installations have accessed the Seattle software updater from January 2010 to January 2011, among these at least 1455 are located in a customer premises network. Since the access base of the host computers running the Seattle network is more heterogeneous than in PlanetLab, one can assume that the connectivity characteristics of the average Internet user are represented more realistically. In addition, as there are up to 1000 distinct Seattle nodes concurrently online in contrast to roughly 400 PlanetLab nodes, it is possible to scale measurements beyond previous studies using Seattle. The important question, which types of measurements can be safely conducted by Seattle, will be answered in Section III-C.

III. Seattle as Measurement Testbed

Measuring a period of time is a key element of many metrics and measurement techniques, e.g. bandwidth measurements or, of course, latency measurements. Regarding sound Internet measurement it is vital to have a high temporal resolution and accuracy. But the limited resources and restrictions of a Seattle vessel have an adverse impact on the precision by which time measurements can be performed. For example, when one of the scheduling interrupts occurs while receiving probing packets, the time of the receipt cannot be measured accurately and the measurement could become useless. Apart from delays caused by process scheduling as already elaborated, many other factors can additionally interfere with the accuracy of time measurements. Basically, the timing precision is dependent

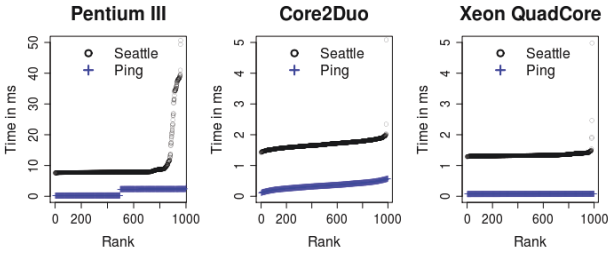

Figure 2. Seattle vs. Ping RTTs inside Emulab

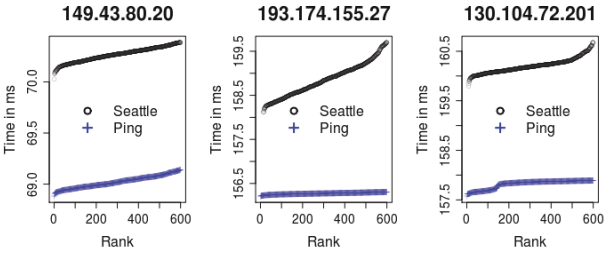

Figure 3. Seattle vs. Ping RTTs outside Emulab

Table I
RTT ESTIMATES BETWEEN TWO HOSTS IN EMULAB.

Hardware	Min.	Med.	Mean	Max.	Var.
Pentium III					
Seattle	7.585	7.805	10.46	169.3	107.6511
Ping	0.199	2.340	1.322	3.13	1.116533
Core 2 Duo					
Seattle	1.417	1.680	1.847	31.38	3.5568
Ping	0.102	0.344	0.344	0.580	0.010697
Xeon Quad					
Seattle	1.279	1.324	1.514	31.14	3.660652
Ping	0.073	0.076	0.076	0.083	3.938e-06

Table II
FILTERED RTT ESTIMATES BETWEEN ONE HOST IN EMULAB TO OUTSIDE SEATTLE NODES.

Host	Min.	Med.	Mean	Max.	Var.
149.43.80.20					
Seattle	70.03	70.30	70.31	70.54	0.01088
Ping	68.88	69.05	69.07	69.26	0.00846
193.174.155.27					
Seattle	158.1	158.8	158.9	159.7	0.14586
Ping	156.2	156.3	156.3	156.4	0.00087
130.104.72.201					
Seattle	159.8	160.2	160.2	160.7	0.02309
Ping	157.6	157.8	157.8	157.9	0.00741

on the clock resolution of the host system; some Windows systems have a resolution with a coarse granularity of 10 ms. This is one of the reasons why we perform all measurements on Linux as operating system, since it provides a much higher timing resolution. Additionally, as the system stack of Seattle is based on multiple layers (compare Figure 1), each layer adds more delays and increases the variance of the measurement noise. To yield an accurate image of the timing precision in Seattle and to be able to estimate the error distribution, we perform measurements inside the controlled test environment of Emulab. These results are compared with measurements taken on a reference system.

A. Measurements inside Emulab

In the first experiment we want to exclude interfering network effects like high and varying cross traffic to omit all causes that could potentially influence our measurement results. Therefore, we use the simplest topology just two computers connected by one link. One computer measures the round-trip-time (RTT) between both hosts 1,000 times as illustrated in Figure 4. The RTT is the two-way delay between two machines on a network as illustrated in Figure 4. One simple method to obtain this value is the ping program. In one measurement scenario we use a modified version of the standard ping program and in the other setup we measure the RTT with a customized version of Seattle on both hosts. The normal ping program is written in plain C code. Therefore, it is reasonably fast and accurate. Yet, we modified the standard ping program to increase the timing granularity of the RTT estimates. Since there is no ping program provided by Seattle, the RTT measurement is done by sending an UDP packet at time t_{R_1} to the receiver, which listens to a pre-defined port and sends immediately an answer back to the sender. When the sender receives the answer at time t_{S_1}, the RTT is then determined by $RTT_1 = t_{R_1} - t_{S_1}$. All RTT measurements are performed sequentially, such that $t_{S_2} = t_{S_1} + t_{wait}$.

Since we are especially interested in the influence of different hardware on the time lags encountered in Seattle, we perform the same experiment with different systems. We switch from a Pentium III on both hosts, to a Core 2 Duo, to a Xeon Quad Core basis, always with the same operating system and the same programs. The results of this experiment are shown in Figure 2. To visualize the distribution of the estimates, we use the rank ordering technique, i.e. sorting the measurements according to the resulting RTT values. Obviously, all estimates yielded by Seattle are above the ping estimates. We omit to plot the biggest outliers in Seattle to retain an expressive graph. One can realize that the difference between the Seattle and the ping estimates shrinks with the computing power of the host

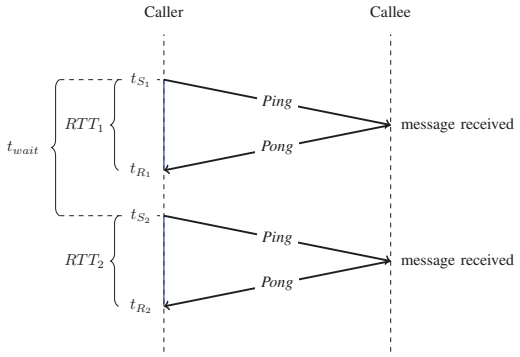

Figure 4. RTT measurement

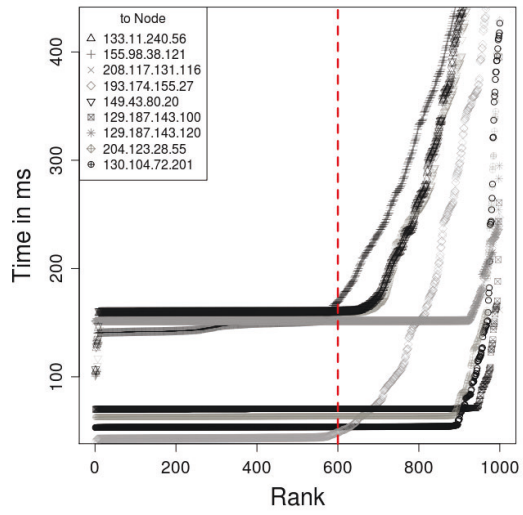

Figure 5. RTT measurements from one Emulab host to nodes of the Seattle network.

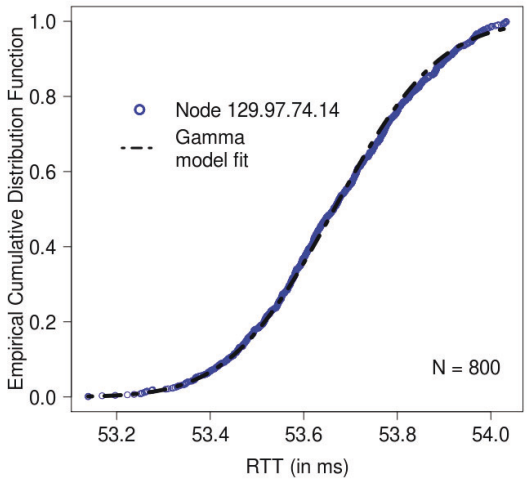

Figure 6. RTT values fitted to a Gamma distribution.

system. However, the large variance is really significant in all Seattle measurements. To provide a better overview over the variability of the RTT estimates, we included Table I, which shows the minimum and maximum results, the mean value, the median and the variance of the estimates. The extreme case can be observed in the experiment with the Pentium III. The variance is as large as 107 ms with a mean value of 10.4 ms, compared to ping estimates with a variance of 1.1 ms and an average value of 1.3 ms. In the other two setups the variance differs by a factor of more than 100 in the Core 2 Duo scenario, respectively a factor of more than 1,000,000 with Xeon Quad Cores as host systems. This effect is mainly due to the resource restrictions of the sandbox and the process scheduling on the host systems. To summarize, with common off-the-shelf hardware the mean value of the RTT differs by more than 1 ms without any network effects. However, the large variability of the estimates obtained by Seattle could be really problematic for sound measurement campaigns.

B. Measurements from Emulab to the Outside World

To investigate the variability of the RTT estimates in depth, we performed the experiment again, but this time with one node located inside the Emulab and others located outside as normal nodes of the Seattle network. Figure 5 shows the 1,000 ranked RTT values to nine other nodes in the Seattle network. In this setup the percentage of the results with a high variation is even more increasing, since there is additional load on the machines causing larger virtualization delays and additionally, the normal link fluctuations of the Internet paths interfere with the measurements.

For a short time interval one can reasonably assume that the average propagation delay of a stationary Internet path is stable. Furthermore, if one only considers the values below the 60 % quantile of the ranked values (the dotted line in Figure 5), the large variability diminishes and the values approach the determined lower bound, provided that we discard all values greater than the median and use the mean of the remaining values to get an approximation of the path RTT. By this simple heuristic we are able to get reasonable accurate approximations of the "true" RTT. We applied our heuristic to the results of the last experiment and we present the modified results of three Seattle nodes in Figure 3. The effects of the heuristic on the variance reduction can very well be observed in Table II. In all other runs of the experiment we were able to obtain RTT estimates with our heuristic in Seattle. They are by 1 to 3 ms above the ping estimates. For metrics, which do not depend on a high accuracy of the estimates, this range of divergence can be acceptable. If one needs on-line results of live measurements, another possibility could be the use of filter functions to reduce the variance caused by the outliers.

As discovered by Bolot [2], the distribution of RTTs in the Internet is best fitted by a constant plus a gamma distribution. The cumulative distribution function of the Gamma distribution is given by

$$F(x;k,\theta) = \frac{1}{\Gamma(k)}\gamma\left(k,\ \frac{x}{\theta}\right) \tag{1}$$

for $k > 0$ and $\theta > 0$, where $\Gamma(k)$ is the gamma function and $\gamma(k,\ x/\theta)$ is the lower incomplete gamma function.

To evaluate the rationality behind our heuristic, we have assessed the fit of the remaining RTT values to a Gamma distribution after removing the outliers. One example is depicted in Figure 6. In this case the Kolmogorov-Smirnov test reports a P-Value of 0.6053 for the fit, and therefore we accept H_0. Other examples also confirm that our heuristic is able to retain a realistic distribution of RTT estimates.

C. Which types of measurements can be safely conducted in Seattle?

Generally speaking, all measurements that do not rely on highly precise time measurements, like network tomogra-

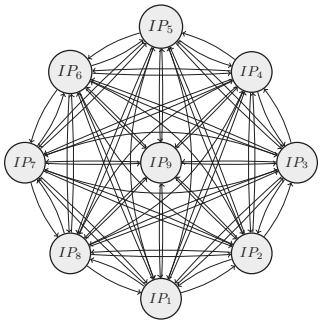

Figure 7. All-Pairs-Ping

phy systems (compare [5]), i.e. packet loss measurements, or geolocation, can be safely performed by Seattle without any objections. Other metrics, like latency or bandwidth measurements, need special attention: The available bandwidth measurement literature distinguishes more or less between two paradigms, *probe rate* and *probe gap* models. Roughly speaking, probe rate techniques try to fill the capacity of the bottleneck link. Since each standard vessel is bandwidth restricted, the probe rate model is not an option here. Tools that use the probe gap model, send a packet pair or a packet train with certain time gaps in between. By the variations of the time gaps between the packets they try to estimate the bandwidth. As a result this technique heavily relies on very accurate time measurements. We conclude that available bandwidth measurements in Seattle will not yield satisfying results at all. Additionally, all measurement studies relying on ICMP measurements are not feasible in Seattle. This is a large drawback and Seattle would be much more useful for Internet measurement practitioners if this limitation would be relaxed.

The problem regarding latency measurements concerns the large variability of the results. One encounters in Seattle a larger variance of the RTT estimates than with the ping program, which is mainly caused by the interruptions and sleep phases of the scheduling and not by the changing conditions along the path. To address this problem, we present a simple heuristic to compensate the large variability regarding the RTT estimates. Using this heuristic, we are able to reduce the variance of the estimates so far that future work in Seattle could include the comparison of network coordinate systems like that one presented in [9] or the validation of delay-based geolocation studies, as shown in [6] or [7].

IV. Exploring the Internet Delay Space

We have discovered that latency measurements are feasible in Seattle by applying our heuristic, provided that one can tolerate overestimations in the range between 1 and 3 ms. Thus, we use Seattle to investigate the delay space between its nodes in the Internet. The delay space or latency measurements can be useful for a wide variety of applications. As example, network coordinate systems like Vivaldi use latency measurements; geolocation, content distribution or traffic shaping techniques can be based upon the delay space too.

A. All-Pairs-Ping Algorithm

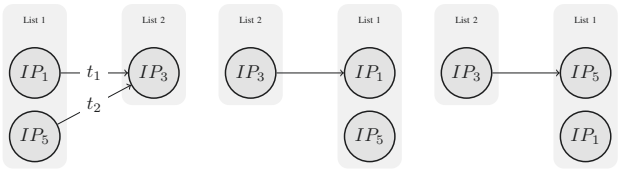

Figure 8. Conquer Phase

The technique to measure the two-way delay between a set of nodes is known as all-pairs-ping. This procedure constructs a latency matrix of the RTT estimates between all nodes of a network (see Figure 7). Cappos *et al.* presented in [3] an all-pairs-ping implementation as an example for the simplicity by which distributed programs can be implemented in Seattle. The presented implementation consists only of 30 lines of code. However, this simplicity comes at a price, as there is no coordination between the measurement nodes; thus, it happens quite often that measurements interfere with each other, and thereby, distort the measurement results. We seek to enhance the provided algorithm by a central coordination instance to avoid such interferences. Additionally, as the amount of measurements grows exponentially with the number of nodes, we implement a round based measurement tournament inspired by the divide-and-conquer algorithm to increase the measurement speed. To obtain latency matrices by this approach, we implement two programs, one that monitors and controls the steps of the divide-and-conquer algorithm, which is only installed on one node, and another one that is responsible for performing the measurements on each tested Seattle vessel. We present both programs in the following:

Node. The corresponding program is responsible for the execution of the measurements. Its operations are governed by receiving messages from the controller. Each node will produce a list of n measurements with a specific waiting time t_{wait} (compare Figure 4) between each measurement. To capture the three data sets, we use $n = 50$ trials for each node with a waiting time of $t_{wait} = 0.4$ s. However, in our implementation all parameters are configurable. The node program is also responsible for the collection of the measurement results and the first preprocessing of the results, i.e. it applies the proposed heuristic to yield a more accurate estimation of the RTT.

Controller. The controller node is responsible for the execution of the divide-and-conquer algorithm and the coordination of the measurement rounds. For this purpose it obtains the list S with the IPs of all Seattle nodes to be measured. At first, in the divide step, the list is split into two sublists S_1 and S_2. If the number of nodes is odd, S_1 receives the last remaining node. Then, in the conquer phase, the controller sends a message to each node of the sublist S_1 to indicate, which node of S_2 should be measured. As shown in Figure 8, the controller will ensure that the starting times of the measurements are not

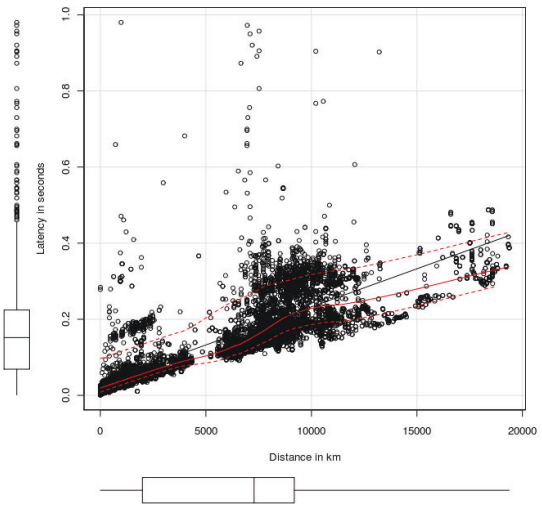

Figure 9. Latency vs. Distance

overlapping. In this case, IP_1 will measure the RTT to IP_3 at time t_1 and IP_5 will conduct the measurement at time t_2. Each node of sublist S_1 tries to obtain all RTTs to the nodes of S_2; this is done by permuting the measured list in each measurement step such that all possible pairs are measured. After the complete permutation of one list the measurement direction is changed and the conquer phase begins again (see the next two steps in Figure 8). When the measurements have been performed bidirectionally and the length of one of the remaining sublists is greater than one, the divide-and-conquer algorithm starts recursively again.

After the termination of the algorithm all nodes send their results to the controller, which subsequently constructs the latency matrix. Furthermore, the controller computes the following values for the measurement results of each IP tuple: mean, median, quantiles (25 %, 75 %), and an estimation of the standard deviation respectively the variance. When the processing of the data is finished, the measurement results can be obtained in the following formats: CSV, HTML and LaTeX. Additionally, the controller provides a web server functionality to display the results and it visualizes each conquer step graphically with the TikZ framework. For instance, Figure 8 has been automatically generated in this way. Our implementation is publicly available on the official Seattle wiki. [1]

B. Measuring the Delay Space

To gather the latency matrices we use more than 120 distinct vessels, i.e. nodes on separated networks. Due to the usual Internet measurement problems, like triangle inequality violations or churn, we obtain in each of our three experiments complete data sets of roughly 100 vessels. Every latency matrix consists of the consolidation of more than 700,000 measurement probes. The captured data sets are also available at the Seattle wiki.[1]

[1]https://seattle.cs.washington.edu/wiki/CollectedNodeData/DelaySpace

C. Case Study: Distance Prediction

To illustrate the usefulness of our implementation we conducted a simple exemplary case study that investigates the relation between the delay and distance. Such delay-based geolocation relies upon measurements of end-to-end network delays to locate a target IP. Using the distance-to-delay function, one can transform the observed delay to the target into a predicted distance to the target. Figure 9 shows a scatter plot of the relation between the latency and the geographic distance in one of the experiments. Each data point represents the approximated RTT between two IPs. The value is obtained by using the heuristic proposed in Section III-B on the captured RTTs. One can observe in Figure 9 that the relation between latency and distance can be fitted by a linear regression function (the lines in Figure 9).

To determine the location of a given IP, we use an on-line GeoDB. By this means we can get the geographic information for each IP address, i.e. the country, the longitude and the latitude. Provided this information, we use the spherical law of cosines to calculate an approximation of the air-line distance between the IP tuples (compare with [11]). :

$$dist(SP, EP) = \arccos\{\sin(La_{EP}) \cdot \sin(La_{SP}) + \cos(La_{EP}) \cdot \cos(La_{SP}) \cdot \cos(Lo_{EP} - Lo_{SP})\} \cdot r \quad (2)$$

In Equation 2 the starting point is denoted by SP and the end point by EP. The geo-coordinates (Lo represents the longitude and La the latitude) must be given as radian values. We assume that the radius of the earth is $r = 6378.137$ km. Of course, this is just a rough estimation, but without knowing the length of the cables that carry the probes there is no need for more precise distance formulas.

V. Conclusion

In this paper we investigate the feasibility of Internet measurement campaigns by the educational research framework Seattle. This study presents the first investigation of the prevalent conditions in Seattle. In particular, we characterize the usability of Seattle for distributed measurement campaigns and we discuss which kind of metrics can be obtained by the help of this platform. In addition, we elaborate the kind of measures needed to yield reasonably accurate latency measurements on Seattle. By providing a publicly available implementation of a coordinated all-pairs-ping implementation, we lay the foundation for further investigations regarding the Internet delay space by Seattle. Our software and the obtained data sets are publicly available at the official Seattle wiki.[1]

Acknowledgments

The authors would like to thank Justin Cappos for his assistance and support with the Seattle framework. The authors acknowledge the partial financial support by the COST project IC0703.

REFERENCES

[1] Planetlab. http://www.planet-lab.org.

[2] J.-C. Bolot. End-to-end packet delay and loss behavior in the internet. In *Conference proceedings on Communications architectures, protocols and applications*, SIGCOMM '93, pages 289–298, 1993.

[3] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson. Seattle: a platform for educational cloud computing. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 111–115, 2009.

[4] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: a decentralized network coordinate system. In *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '04, pages 15–26, 2004.

[5] D. Ghita, H. Nguyen, M. Kurant, K. Argyraki, and P. Thiran. Netscope: practical network loss tomography. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 1262–1270, 2010.

[6] B. Gueye, A. Ziviani, M. Crovella, and S. Fdida. Constraint-based geolocation of internet hosts. *IEEE/ACM Transactions on Networking*, 14:1219–1232, 2006.

[7] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe. Towards ip geolocation using delay and topology measurements. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, IMC '06, pages 71–84, 2006.

[8] L. Leonini, E. Rivière, and P. Felber. Splay: distributed systems evaluation made simple (or how to turn ideas into live systems in a breeze). In *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, NSDI '09, pages 185–198, 2009.

[9] P. Pietzuch, J. Ledlie, and M. Seltzer. Supporting network coordinates on planetlab. In *Proceedings of the 2nd conference on Real, Large Distributed Systems - Volume 2*, WORLDS'05, pages 19–24, 2005.

[10] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine. Survivable key compromise in software update systems. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 61–72, 2010.

[11] W. M. Smart and R. M. Green. *Textbook on Spherical Astronomy*. 1977.

[12] N. Spring, L. Peterson, A. Bavier, and V. Pai. Using planetlab for network research: myths, realities, and best practices. *ACM SIGOPS Operating Systems Review*, 40:17–24, 2006.

[13] M. Steiner and E. W. Biersack. Where is my peer? Evaluation of the vivaldi network coordinate system in azureus. In *Proceedings of the 8th International IFIP-TC 6 Networking Conference*, NETWORKING '09, pages 145–156, 2009.

[14] K. Tutschku, A. Rafetseder, J. Eisl, and W. Wiedermann. Towards sustained multi media experience in the future mobile internet. In *14th International Conference on Intelligence in Next Generation Networks*, ICIN '10, pages 1–6, 2010.

[15] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, OSDI '02, pages 255–270, 2002.