How Network Reverse Engineering Differs from Software Reverse Engineering

Hui Zhou and Wencai Du

College of Information Science & Technology, Hainan University
Renmin Ave No.58, Haikou, China, 570228
h.zhou.china@gmail.com, wencai@hainu.edu.cn

Abstract. Software reverse engineering has undergone many milestones and stepped from research to industry quickly in recent ten years. By analogy, researchers have found that it is also possible to apply reverse engineering to computer networks. The goal of network reverse engineering is to annotate a living map of the networks, which exhibits node role, link connectivity, topology dynamics, and bandwidth usage. It is necessary, but also challenging, to employ reverse engineering to computer network. We present an comparatively analysis on the reverse engineering of both software and network from five fundamental perspectives: source, analysis, presentation, validation, and prediction. The comparison indicates that both software and network communities would benefit from the collaborative effort on reverse engineering.

Keywords: Network reverse engineering, software reverse engineering.

1 Introduction

Software reverse engineering is, in practice, one of the most important endeavors in software engineering. This stems from the fact that software systems are complex and often poorly specified and documented. As a result, software practitioners need to spend a substantial amount of time understanding the source code from a structural and behavioral perspective, before carrying out any maintenance task. In this context, most reverse engineering processes follow the same pattern: a program is analyzed through static or dynamic analysis and the collected low-level program information is transformed into a higher level, more abstract presentation. The presentation helps engineers better understand the rationale of the code and thus facilitate future refactoring.

From networking standpoint, reverse engineering is the process of analyzing a target network so as to identify the design of network and create presentations. Specifically, we take "design" to mean how its components, e.g. node, link and internal networks, are assembled and configured, as well as runtime properties including link available bandwidth, node congestion status, and end-to-end packet transmission delay.

It has become obviously necessary to employ reverse engineering to computer networks, and there have been a few experimental studies on this [1]. However,

P. Sénac, M. Ott, and A. Seneviratne (Eds.): ICWCA 2011, LNICST 72, pp. 33–43, 2012.

network reverse engineering is a challenging task. The key reason is that the design of the Internet can't provide explicit support for end nodes to obtain information about the network internals. A network typically consists of many small networks; such networks are under different administrative control, so there is no single place from which one can obtain a complete picture of the specified target network. Furthermore, the Internet is so heterogeneous that an approach found to be useful in a certain networks may not be effective elsewhere [2].

To reverse engineer the computer networks, we not only need to study network technology, but also need to understand software engineering. We argue that a collaborative effort of software and network domains would achieve significantly more than the isolated efforts of individuals. This paper tries to answer questions like "how does the reverse engineering of software and network differ?" and "can they benefit from each other?" To do so, we analyze both software reverse engineering and network reverse engineering from five basic perspectives: source, data analysis, presentation, validation, and prediction.

This paper is organized as follows. Section 2 first summarizes the recent progress on computer network reverse engineering. Section 3 analyzes the reverse engineering techniques of both software and networking, and then Section 4 concludes the paper with a short discusses.

2 Related Works

The field of software reverse engineering and its closely related fields, such as program comprehension or software analysis, have undergone many successes over the past 20 years. In addition, software reverse engineering environment has been equipped with various intelligent tools: extractors, analyzers, and repositories [3]. During the same time, along another thread, network community has introduced quite a few measurement systems to gathering and presenting the information of network properties [4]. The theories, protocols, techniques, tools, overlay framework, and the released data archives have initially make up the main body of network reverse engineering.

The reverse engineering of computer network mainly starts from measurement. Specifically, a router can be configured to passively record the information about its own performance, e.g. the number of packets received/sent by each of its network interface cards (NICs). A typical example is network traffic monitoring. Fig. 1 illustrates the bytes sent through the USENET bulletin board system, averaged over two-week intervals [5].

Furthermore, the measurement literature can further be classified according to different targets: node, link, topology, and packet pattern. Learning the role that a node plays is the first step to understand the network. Basically, each node has one of the following roles: client host; access router that aggregates the traffic from clients; and backbone router that transmits a large volume of traffic. The role problem has been frequently addressed, e.g. Rocketfuel [6] uses IP prefixes, DNS information, and topological ordering to identify role. In addition, many tools search for the bottleneck node with diverse heuristics [7].

Besides node, link is another important component. Generally, a link is the IP connection between two nodes that are only one IP-hop away from each other. Much research has been done to capture the usability, delay, and bandwidth capacity of a single link. Recently, the research community extends the study of link to end-to-end path, which can be regarded as a line of connected links. Measuring the properties of a path is very meaningful since it enables us to better understand how packets flow between nodes. For example, variation in the transmission delay of path is both a problem for time-critical traffic and a key indication of network congestion. Typically, tools use Internet control message protocol (ICMP) [8] timestamps to estimate the delay variation.

Finally, topology auto-discovery has strongly driven the study of active probing measurement. Network community has examined five categories of topologies: the graphs of connections between autonomous systems (ASs) [6], the point-of-presence (POP) topologies that interpret the structure of backbone using geography information, the IP-level topologies whose nodes are IP addresses and whose links are connections between the IP addresses, the router-level topologies that resolve IP aliases and group the IP addresses in the unit of router, and the connectivity of physical components, including routers, switches, and bridges. In particular, the router-level topology has attracted more interest than the others because it establishes the basis of AS and POP topologies, gives a more operational picture than the IP topology. As an example, Fig. 2 gives the result of a topology discovery work; the target network is Abilene backbone, i.e. an educational IPv6 network in America [9].

3 Comparative Analysis

We comparatively analyzed the reverse engineering of software and network from five basic perspectives: source, data analysis, presentation, validation, and prediction. The following analysis isn't exhaustive since we can't cover every aspect, but it offers a skeletal picture on the differences of software reverse engineering and network reverse engineering, and highlights the challenges faced by the network community.

3.1 Source

The source of software reverse engineering is code and code-related files such as log. Generally, software reverse engineering depends on performing some analysis of the source code in order to produce one or more models of the system under analysis. Generally, source code is written by software engineers according to the well-designed specification of programming languages, e.g. ASM, Pascal, C/C++, and Java. A language often comes with a specification, to which compiler developer and software engineer must conform. Furthermore, the coding process is supported by various integrated development environments. As a result, no matter how well (or bad) the code is organized, software reverse engineering tools is built on a solid basis, i.e. the tools do understand the exact meaning of each line of code.

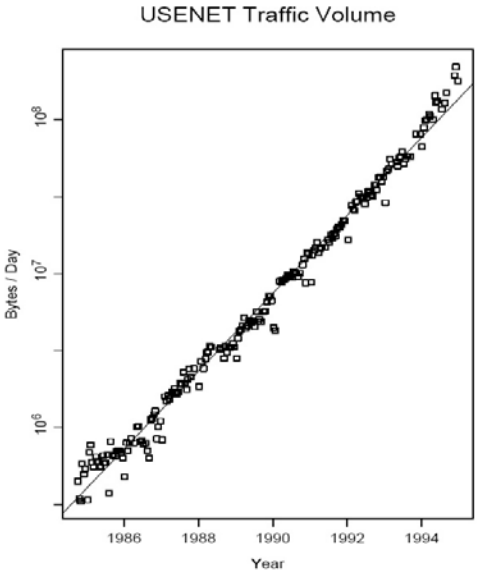

Fig. 1. USENET traffic monitoring information [4]

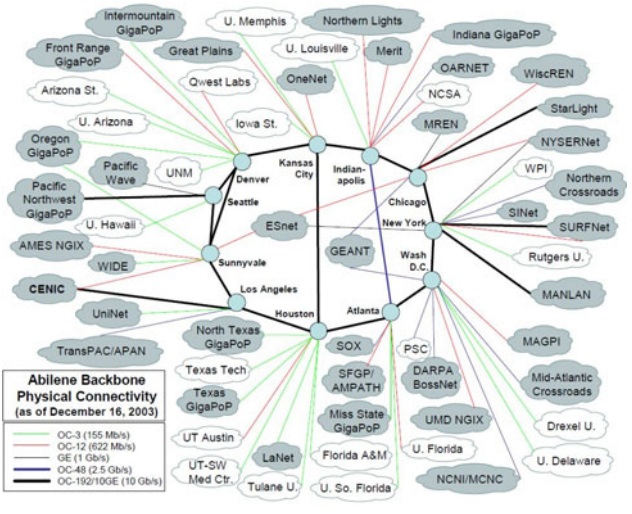

Fig. 2. The discovered topology of Abilene backbone [9]

Unlike the source of software reverse engineering, the one of network reverse engineering mainly comes from measurement, and it is highly volatile. The volatility can be perceived in almost every parameter that we attempt to measure. For example, the round-trip time (RTT) of a pair of nodes is an important metric of network performance. Generally, RTT can be used as an indicator of end-to-end transmission quality. Here we attempt to measure the RTT of a short path, i.e. two directly

connected computers C_1 and C_2. First, C_1 sends an ICMP echo-request packet to C_2. When C_2 receives the packet, it immediately sends an ICMP echo-reply packet back to C_1. In each active probe, the time from sending out an ICMP echo-request to receiving the corresponding echo-reply is regarded as a candidate of RTT. As shown in Fig. 3, the RTT is ever-changing with network traffic and time.

3.2 Data Analysis

To analyze the source code, a software reverse engineering tool will first scan the source code. In most cases, reverse engineering tool assumes that the target source files won't undergo any change during the scan, which is done once and for all. In a very limited time interval, the source of software is safe to be regarded as static, while network is always a moving target. As a result, network tools must continuously collect the information about the designated network, in a never-ending style.

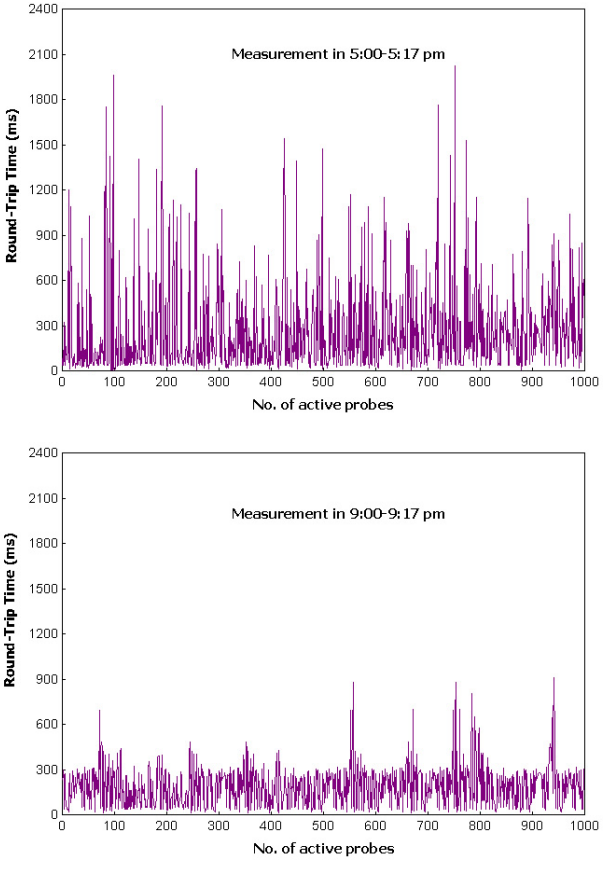

Fig. 3. Round-trip time of two directly connected computers

Moreover, as to network reverse engineering, analyzing the data source is challenging since it generally contains too much noises. But the analysis is valuable since it often provide insight into the network. For example, Faloutsos et al. discover some surprisingly simple power-laws of the network topologies [10]. These power-laws hold for three topologies between November 1997 and December 1998, despite a 45% growth of its size during that period. As shown in Fig. 4, log-log plot of the out-degree d_v versus the rank r_v in the sequence of decreasing out-degree.

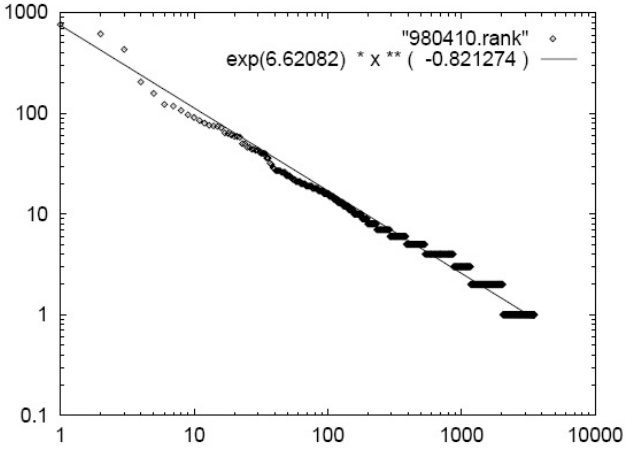

Fig. 4. The rank plots on dataset Intel-98 [10]

3.3 Presentation

After analyzing the source, software reverse engineering tools generally use UML diagrams to visualize the result. Historically, a rich set of diagrams have been introduced by the software research community, and then been applied by the industry. The diagrams, especially the class, activity and sequence diagrams, do provide an abstract and easy-to-understand picture of design, and lead to the prosperity of software modeling.

Software reverse engineering presentation tools support two significant features: design-code linkage and reuse. Specifically, in some reverse engineering tools, source code can be modified by changing the UML diagrams in the presentation. In this way, not only the code can be reorganized, but also some coding functions can be generated by changing the presentation. Besides design-code linkage, the presentation outputted by one tool generally can be reused by many others. For example, the UML diagrams of Java code from Eclipse plug-in can easily be loaded into Rational Rose since the presentations of both systems are UML-compatible.

Suppose that the presentation of software reverse engineering is a snapshot, the one of network reverse engineering can be regarded as a video. The parameters of target network can undergo changes as time passes, and thus lead to high dynamics. As shown in Fig. 5, the IP conversations of LAN captured by Sniffer Pro, which is a

network packet sniffing tool installed in one node [11]. Since the target network is ever-changing, the presentation must trace the changes and output pictures that match.

Compared with software reverse engineering, the network reverse engineering tools can't support large-scale reuse since there isn't a universal accepted presentation standard. It is also hard to establish such a standard because each reverse engineering tool is built to study a specific question and work in a specific network environment.

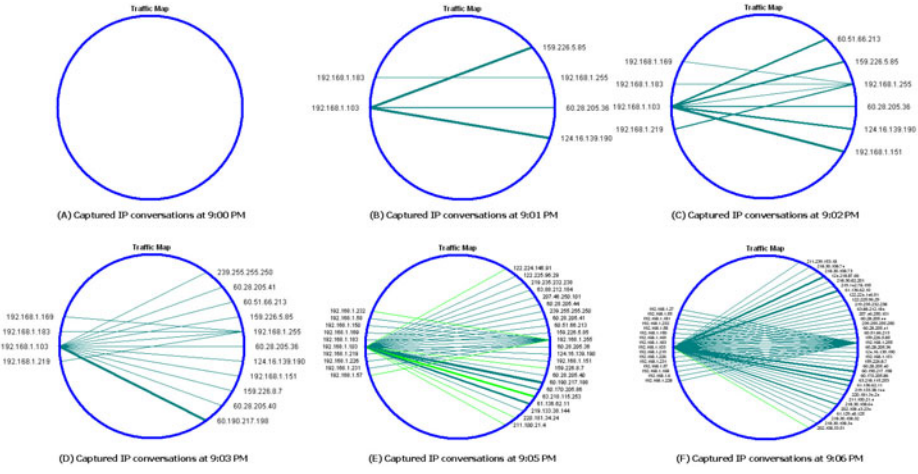

Fig. 5. IP conversations captured by Sniffer Pro in 9:00 – 9:06 PM

3.4 Validation

Usually, every reverse engineering tool needs to validate the correction of its result. Validation is necessary since it enables us to make sure that the data analysis and presentation is correct. However, validating the result of network reverse engineering is a challenging task. Almost all network reverse engineering tools use measurement, instead of the exact value, as data source to perform analysis and presentation. Thus, different reverse engineering algorithms and tools may obtain different result even if the target is the same. A step further, the key problem becomes: does the measurement exactly match the real network situation?

Now suppose that we are validating the available bandwidth of a path, which is defined as the minimum link available-bandwidth of all links along the path [7]. Therefore, the validation requires the information of all link available-bandwidth of target paths. But if the path crosses multiple administrative areas, or it is long enough (e.g. consist of more than six links), it becomes hard to obtain all link available-bandwidth at a specific time [1].

To validate the available bandwidth of a path, researchers have introduced many inspiring techniques. It seems that comparing the estimation result with closely estimated bulk TCP throughput over the same path is a good idea [12]. However, available-bandwidth and bulk TCP throughput are indeed different. The former gives the total spare capacity in the path, independent of which transport protocol attempts

to capture it. While the latter depends on TCP's congestion control. Fig. 6 typically shows the measurement result of the available bandwidth of an end-to-end path, which starts from Hainan University and ends at Chinese Academy of Sciences. In particular, Cprobe [13] and BNeck [7] are installed on hosts inside Hainan, Pathload [14] is installed in both end points, while TCP throughput is tested by maximized the parallel TCP connections of Iperf [15]. It is apparent that there isn't a curve that can exactly match the other. Furthermore, since this end-to-end path traverses the confidential networks of several ISPs, we can't even validate which curve matches the exact situation.

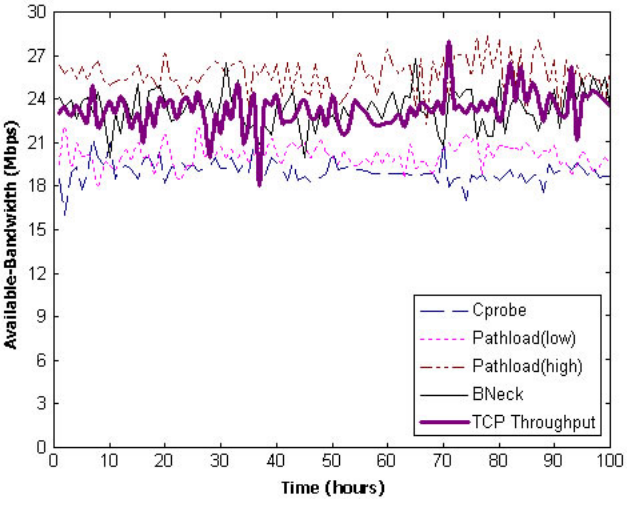

Fig. 6. Available-bandwidth measured by different tools

As a result, we are not able to completely validate end-to-end available bandwidth. Furthermore, it is very hard to make sure the data we collect reflects the exact network status, even if we have success experience on a limited number of networks. The same problem is faced by almost all measurement techniques that rely on active probing. And this thus makes the network reverse engineering more challenging than its software counterpart.

3.5 Prediction

Recently, there is a growing need of reverse engineering tools to support the prediction of changes in source. For example, through analyzing the history of the lines of code, managers can predict the code scale of a Java program in the next development iteration [3]. Surprisingly, though network contains much more noise than stationary software source code, many useful rules have been extracted, and used to predict the macro-behavior of networks.

Diurnal Patterns of Activity: It has been recognized for more than thirty years that network activity patterns follow daily patterns, with human-related activity beginning

to rise around 8-9AM local time, peaking around 11AM, showing a lunch-related noontime dip, picking back up again around 1PM, peaking around 3-4PM, and then declining as the business day ends around 5PM. The pattern often shows activity in the early evening hours, rising around say 8PM and peaking at 10-11PM, diminishing sharply after midnight. Originally, this second rise in activity was presumably due to the "late night hacker" effect, in which users took advantage of better response times during periods of otherwise light load.

Self-Similarity: Longer-term correlations in the packet arrivals seen in aggregated Internet traffic are well described in terms of self-similar processes [16]. "Longer-term" here means, roughly, time scales from hundreds of milliseconds to tens of minutes. The traditional Poisson or Markovian modeling predicts that longer-term correlations should rapidly die out, and consequently that traffic observed on large time scales should appear quite smooth. Nevertheless, a wide body of empirical data argues strongly that these correlations remain non-negligible over a large range of time scales. While on longer time scales, non-stationary effects such as diurnal traffic load patterns (see previous item) become significant. On shorter time scales, effects due to the network transport protocols—which impart a great deal of structure on the timing of consecutive packets—appear to dominate traffic correlations [17].

Heavy-Tailed Distributions: When characterizing distributions associated with network activity, expect to find heavy tails. By a heavy tail, we mean a Pareto distribution with shape parameter $a<2$. These tails are surprising because for $a<2$ the Pareto distribution has infinite variance [17].

An important example is the application of best path selection. The availability of multiple paths between sources and receivers enabled by content distribution, multi-homing, and overlay or virtual networks suggests the need for the ability to select the "best" path for a particular data transfer. A common starting point for this problem is to define "best" in terms of the throughput that can be achieved over a particular path between two end hosts for a given sized TCP transfer [12].

4 Conclusions

Can we reverse engineer the computer networks? Network reverse engineering is the process of annotating a map of the designated network with properties such as: client populations, features and workloads; network ownership, capacity, connectivity, geography and routing policies; patterns of loss, congestion, failure and growth; and so forth. Naturally, the urgent need of exploring network internals gives birth to the network reverse engineering. The key challenge of network reverse engineering goes with the birth of Internet: the design of network doesn't provide explicit support for end nodes to gain the information of network internals. The measurement approach has led to many techniques, tools, and data, but it can't achieve the goal alone.

Is the reverse engineering of software and network the same? From a general perspective, the reverse engineering of both fields are the same in that it analyzes a subject system to (1) identify the system's components and their interrelationships and (2) create representations of the system in another form or a higher level of abstraction. Both software reverse engineering and network go through a roughly

common process: reading source, carrying out analysis, creating presentation, validating result, predicting changes, and mining knowledge.

Is the reverse engineering of software and network different? Indeed, software reverse engineering and its network counterpart are not the same things. They come from different background, aiming at solving different problems, and have been supported by two lines of methods, techniques, tools, applications, and so forth. Specifically, their data sources come in two totally different forms, their analysis require different knowledge and techniques, their presentation are based on sharply different abstract models, their validation meets two domains of challenges, and finally their prediction and data mining focuses on different emphases.

Can we benefit from the collaborative effort? Both software and network fields have made an incredible number of contributions, and have learnt from each other at various points. For example, RichMap uses snapshot concept to optimize its models and fasten its presentation. Another example is that the patch dissemination system can also optimize its delivery strategies according to the traffic load measured by networking system [18]. We believe that both software and network communities can strongly benefit from the collaborative effort on reverse engineering computer networks, more than the hints illustrated by RichMap.

Acknowledgment. We gratefully acknowledge the financial support of the Project 211 supported coordinately by the State Planning Commission, Ministry of Education and Ministry of Finance, China.

References

1. Zhou, H., Wang, Y.: RichMap: Combining the Techniques of Bandwidth Estimation and Topology Discovery. Journal of Internet Engineering 1(2), 102–113 (2008)
2. Zhou, H., Wang, Y., Wang, X., Huai, X.: Difficulties in Estimating Available-bandwidth. In: Proceedings of IEEE International Conference on Communications, pp. 704–709 (2006)
3. Kienle, H.: Building Reverse Engineering Tools with Components. Ph.D. Thesis, Department of Computer Science, University of Victoria, Canada, 325 p (2006)
4. Labovitz, C., et al.: Internet inter-domain traffic. In: Proc. ACM SIGCOMM (2010)
5. Thompson, K., Miller, G., Wilder, R.: Wide-area Internet Traffic Patterns and Characteristics. IEEE Network,10–23 (1997)
6. Spring, N., Mahajan, R., Wetherall, D., Anderson, T.: Measuring ISP Topologies with Rocketfuel. IEEE/ACM Trans. Networking 12(1), 2–16 (2004)
7. Zhou, H., Wang, Q., Wang, Y.: Measuring Internet Bottlenecks: Location, Capacity, and Available Bandwidth. In: Proceedings of International Conference on Computer Network and Mobile Computing, pp. 1052–1062 (2005)
8. Postel, J.: Internet Control Message Protocol. IETF RFC 792 (September 1981)
9. Abilene Network, `http://www.internet2.edu/abilene`
10. Faloutsos, M., Faloutsos, P., Faloutsos, C.: On Power-law Relationships of the Internet Topology. In: Proceedings of ACM SIGCOMM, Cambridge, USA (1999)
11. Sniffer Pro., `http://www.netscout.com/`
12. He, Q., Dovrolis, C., Ammar, M.: On the Predictability of Large Transfer TCP Throughput. Computer Networks 51(14), 3959–3977 (2007)

13. Carter, R., Crovella, M.: Measuring Bottleneck Link Speed in Packet-switched Networks. Performance Evaluation 27(28), 297–318 (1996)
14. Jain, M., Dovrolis, C.: End-to-end Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. IEEE/ACM Trans. Networking 11(4), 537–549 (2003)
15. Tirumala, A., Qin, F., Dugan, J., Ferguson, J., Gibbs, K.: Iperf - The TCP/UDP Bandwidth Measurement Tool
16. http://dast.nlanr.net/Projects/Iperf/
17. Zhang, Y., Duffield, N., Paxson, V., Shenker, S.: On the Constancy of Internet Path Properties. In: Proceedings of ACM SIGCOMM conference on Internet measurement, pp. 197–211 (2001)
18. Paxson, V.: End-to-end Internet Packet Dynamics. In: Proceedings of ACM SIGCOMM (1997)
19. Gkantsidis, C., Karagiannis, T., Vojnovi, M.: Planet Scale Software Updates. ACM SIGCOMM Computer Communication Review 36(4), 423–434 (2006)