Leveraging Router Programmability for Traffic Matrix Computation

Giacomo Balestra[‡] Salvatore Luciano[*]

Maurizio Pizzonia[*] Stefano Vissicchio[*]

[*] Dept. of Computer Science and Automation
Roma Tre University, Rome, Italy
{salv.luciano}@gmail.com
{pizzonia,vissicch}@dia.uniroma3.it

[‡] CASPUR
Rome, Italy
{giacomo.balestra}@gmail.com

ABSTRACT

Traffic matrices are used by Internet Service Providers as an input for many business-critical activities like capacity planning and traffic engineering. Despite their importance, current approaches for computing traffic matrices are either unpractical or not arbitrarily accurate.

In this paper, we propose a novel distributed architecture in which programmable routers autonomously compute parts of the whole traffic matrix. Our proposal conjugates router programmability with some ideas firstly proposed by Varghese and Estan [17] and avoids the need for packet sampling exploiting highly optimized counting mechanisms, already available on commercial routers.

We describe our architecture and analyze main implementative issues. Also, we show the viability of our solution by describing a prototypical implementation and some experimental results.

1. INTRODUCTION

A traffic matrix (TM) quantifies the amount of traffic traversing a network from every ingress point to every egress point in a given time interval. TMs are practically used by Internet Service Providers (ISPs) as an input for several business-critical activities related to the network design, e.g., capacity planning, provisioning and traffic engineering [11]. Computing TMs in real-time would also enable anomaly detection and on-line diagnosis of routing events and device failures [20].

Computing TMs is a challenging task, since direct measurement is unpractical [20], packet sampling forces a trade-off between the overhead imposed on network devices and the accuracy of the TM, and estimation techniques rely on statistical assumptions and cannot be arbitrarily accurate [10].

In [17], Varghese and Estan envision a new measurement approach, which can lead to a superior trade-off, by carefully evaluating implementation costs, understanding real needs, and leveraging other system parts. We believe that this challenge can now be tackled exploiting router programmability features recently made available in commercial products.

Following this idea, in this paper we propose a new architecture for directly measuring TMs, founding on the possibilities opened by router programmability. Because of their practical importance for ISPs [3], we focus on PoP-to-PoP TMs, that account of traffic between each pair of Point of Presence of an ISP. However, our solution can be easily extended to other types of TMs. Contrary to existing approaches, we develop an actually distributed solution for the TM computation. Indeed, our approach is based on the possibility to program routers to autonomously compute parts of TM and a central component is only used to trivially combine precomputed data and display the TM. This allows us to limit the total overhead [11]. Our architecture can be implemented using current technologies and avoids the need for packet sampling since it leverages highly optimized packet counting features already available on commercial routers.

In order to show the feasibility of our solution, we prototypically realized our idea on a Juniper router and experimentally evaluated our prototype in a testbed. Preliminary experiments show that our architecture has the potential to enable accurate computation of TMs with a tunable time granularity and limited router load.

The rest of this paper is organized as follows. In Section 2, we describe existing solutions for computing TMs along with their major limitations, further motivating our effort for finding a more efficient solution. In Section 3 and 4, we describe our proposed architecture and our prototypical implementation, respectively. In Section 5, we report the results of our experimental evaluation. Finally, we conclude in Section 6.

2. RELATED WORK

Direct measurement of traffic data on network devices is probably the most straightforward approach to compute TMs. Unfortunately, this approach is generally considered unpractical, or even unfeasible, because of the huge amount of data to be managed and the lack of an appropriate measurement infrastructure [20].

To mitigate these problems, sampled flow data are typically collected on routers and combined with routing information. In particular, Cisco NetFlow v9 [1] allows collection of flow records that also include some routing information (e.g., the BGP `next-hop` address). Techniques based on sampled flow statistics are presented in [6, 11]. However, packet sampling forces a trade-off between the overhead imposed on network devices and the accuracy of the TM.

Estimation techniques based on indirect measures, e.g., aggregated link loads that can be collected using SNMP [4], are proposed in literature as an alternative approach. Most of these techniques are based on mathematical and statistical methods and can combine multiple data sources (e.g., SNMP and NetFlow data, BGP routing information, etc.) [10, 19, 20]. However, this approach generally relies on statistical assumptions and periodic snapshots of routing information, resulting in a final estimation whose accuracy cannot be arbitrarily high (error is typically about 10%) and also depends on the absence of routing changes [10, 20, 17].

A simulation approach is proposed in [16].

All the solutions listed above are centralized in that they require a central component that gathers all necessary data and computes, or estimates, the TM. A distributed solution is advocated in [11] in order to improve efficiency and lower total overhead. In this paper, we propose an architecture that enables the TM computation to be distributed, relying on current technologies.

An approach to directly collect statistics on routers is proposed in [18], but the technique can only be applied to MPLS networks. More recently, a new approach to directly measure TMs, based on accounting capabilities of switches, is proposed in [15]. However, its applicability is limited to OpenFlow networks and the usage of per flow counters would not scale to large ISP networks.

Our solution philosophically conforms to the general proposal described in [17], where authors suggest to map prefixes into equivalence classes (according to the BGP `next-hop`) and set up per-class counters. However, leveraging router programmability, we do not require any support from routing protocols. This makes our solution readily deployable.

3. COMPUTING THE TRAFFIC MATRIX INSIDE PROGRAMMABLE ROUTERS

Nowadays, the vast majority of commercial devices implements a *counting mechanism* (*CM*) that can be configured to selectively count traversing traffic. Such a counting mechanism allows operators to define a set of *counting rules*, each responsible for incrementing a specific *counter* when traffic satisfies a given *matching condition*. Being involved in packet forwarding, this mechanism is tightly integrated within the data plane and is highly optimized [14]. CMs usually track both number of packets and bytes.

A CM can be activated on real-world devices through simple configurations. E.g., it can be enabled on Juniper routers using the `count` statement within an appropriate `firewall filter`. On Cisco routers, statements `policy-map` and `class-map` can be used for the same purpose. Current router architectures do not allow any matching condition to be specified. Indeed, CMs can match traffic only based on the information that is known by the data plane, e.g., the IP `next-hop`, the IP `destination address` and other fields of the IP packet. Unfortunately, computing a PoP-to-PoP TM requires to correlate, at each ingress point, the `destination address` inside IP packets with the BGP `next-hop` attribute [12] as specified in the BGP Routing Information Base (RIB) of traversed routers.

Since collecting per-prefix data does not scale up in common scenarios [17], plain usage of counting mechanisms is not viable for computing TMs. We propose to exploit router programmability to conveniently configure the CM on routers located at each PoP, grouping IP flows on the basis of the BGP `next-hop` and dynamically adapting counting rules to routing changes.

3.1 Architecture

Fig. 1 shows our proposed architecture and the interaction between components. Router programmability allows us to add *new* modules to the *standard* modules of a router.

Among standard modules, the *BGP routing daemon* computes the association between each IP prefix and the BGP `next-hop` to which traffic destined to that prefix has to be forwarded. Also, the *counting mechanism* (*CM*) is used as a primitive in order to install appropriate counting rules on the router.

Counting rules to be installed are defined by the *counting rules computing module* (*CRCM*), which is new with

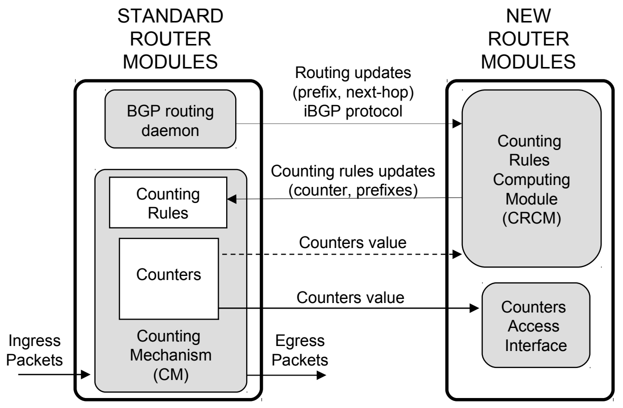

Figure 1: Proposed architecture.

respect to the standard router modules. The CRCM configures the CM as follows. One counter is set for each BGP `next-hop`. Each counting rule R matches the IP `destination address` of traversing packets against the IP prefixes associated to a specific BGP `next-hop` $H(R)$, and increments the counter associated to $H(R)$.

The most important task of the CRCM is to guarantee the consistency between counting rules and BGP routing information. For this purpose, it interacts with both the CM and the BGP routing daemon. Interaction with the BGP routing daemon is performed using standard protocols, since currently available API does not provide special hooks for BGP events. In our architecture CRCM maintains an iBGP session with the BGP routing daemon in which the daemon is configured as route reflector [2]. Over that iBGP session, the CRCM receives all the BGP updates destined to the router. After the reception of a BGP update, the CRCM modifies the set of the counting rules accordingly and communicates to the CM the changes that must be performed on the counting rules installed on the router.

A *counters access interface* is also added to standard router modules. It enables external systems to easily access the values of counters. In practice, the counters access interface can be either designed from scratch or based on other standard protocols like SNMP.

3.2 Practical Issues

A few issues must be handled when implementing the architecture depicted in Fig. 1.

Firstly, the CM might take some time to activate a new counting rule or to modify an existing one, since compilation and optimization steps may be required. Moreover, routing events, like routing instabilities or BGP path explorations, can generate many short-lived routing updates that can force the system to update rules at an unnecessarily high rate. Depending on implementative aspects, the continuous modification of single counting rules can be time consuming and can introduce errors in the TM computation. For these reasons, the CRCM can gather BGP updates and require the CM to update the counting rules at tunable time intervals, e.g., periodically. This allows the CM to use a single transaction for updating many rules and helps limiting the impact of transient routing events. Time granularity at which counting rules are updated on the device can be conveniently tuned according to specific needs and device performance. Clearly, postponing rule updates can, in turn, introduce errors, since the CM is not always aligned with the routing control plane. To minimize such errors, the CRCM can assign a priority value to each counting rule according to the amount of traffic handled by that rule, making the system more responsive to updates that are more likely to introduce a larger error, if postponed. Effectiveness of the priority mechanism is increased by the fact that traffic distribution across the Internet is becoming more and more skewed [9]. To prioritize rule updates, the CRCM could take as input the values of counters stored by the CM (dotted line in Fig. 1).

Also, the CRCM can be implemented either totally inside routers or, at least partially, outside them. Both options allow a distributed solution to be deployed, mandating each router to autonomously compute a part of the TM, thus significantly reducing the total overhead [11]. The first option minimizes the communication overhead while forwarding capabilities of routers should not be affected, since the CRCM only needs to use the control plane CPU and memory (see Section 4).

On the other hand, the latter option would make the architecture suitable for more flexible measurement operations. For instance, moving the CRCM outside routers allows the logic for rule computation to be centralized and easily modified, e.g., providing support for what-if analyses. For example, an operator could compute the traffic matrix resulting from applying a different BGP configuration to routers, using a fictitious BGP RIB to feed the CRCM. Such a flexibility comes at the cost of re-introducing some communication overhead.

3.3 Key Benefits and Major Limitations

In order to efficiently manage huge volumes of traffic, the counting mechanisms already available on routers are typically implemented in a highly optimized way (e.g., using ASICs) [14]. One of the major benefits of our architecture is that it exploits these optimized mechanisms, making packet sampling unnecessary. It consumes a limited amount of memory, since no data on IP traffic flows is stored but only a counter per BGP `next-hop` and a set of counting rules. Also, the way counting rules are actually stored on devices depends

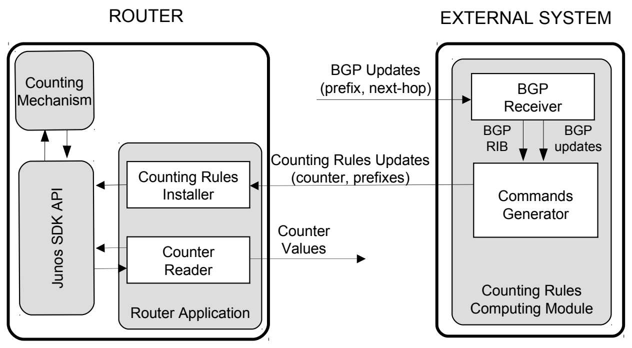

Figure 2: High-level view of the prototype.

on the implementation of the CM, that is designed to be scalable since packet forwarding is concerned.

A complete performance study is subject for future work. However, founding on the results of our preliminary experiments (see Section 5), we argue that our architecture can potentially enable network operators to accurately measure TMs in near real-time.

Even if we implemented a prototype on a Juniper device, our architecture can, in principle, be deployed at any router that allows to configure the CM using programmability features. In this sense, our approach is vendor independent.

Nevertheless, our architecture relies on few reasonable assumptions concerning the routing control plane. First of all, routing information is assumed to be correct, i.e., network configuration is assumed not to be subject to routing or forwarding anomalies [7]. If such anomalies can arise in the network, the computation of the TM cannot rely only on local BGP routing information owned by single routers. We also assume that all destination network prefixes are advertised in BGP. Finally, the association between each BGP `next-hop` and the corresponding PoP is assumed to be known or to be easily derived from known topological information.

4. IMPLEMENTATION

We developed a prototypical implementation of the proposed architecture. A high-level diagram of our prototype is depicted in Fig. 2. In order to facilitate the installation of counting rules and the access to the values of counters, we deployed a *router application* inside a programmable router. The router application communicates with an *external system* that implements the core functionalities of the CRCM. The choice of implementing the CRCM outside the router is motivated by the possibility of rapidly prototyping the module and easily experimenting different strategies. Of course, deploying the CRCM inside the router is also possible and it is part of our future work. All the operations performed by the external system are CPU-bound and communications between submodules happen over pipes. Thus, implementing the whole architecture inside the router should only affect the control plane CPU and memory usage, that is, device capability to route packets should not be concerned, being forwarding and control plane typically separated in commercial devices.

To realize the router application, we used the Junos SDK, which allows third party developers to install custom applications inside Juniper routers [8]. To define and modify counting rules, our prototype exploits the firewall filters manipulation facilities provided by the Junos SDK API.

In our implementation, the *counting rules installer* component allows the external system to specify simple *commands* for adding and removing counting rules, using an ad-hoc protocol based on TCP. Right after receiving a command, the counting rules installer uses the Junos SDK API to instruct Junos to perform the operation specified by the command. Access to counters is provided by the *counter reader* component, which can print current values of counters on a plain text file. We plan to provide support for SNMP in future work.

The external system realizing the core of CRCM consists of two submodules: a BGP receiver and a commands generator.

The *BGP receiver* is realized as a customly modified version of the Python Routeing Toolkit (PyRT) open-source software. It maintains a BGP session with a given BGP peer and dumps the association between each IP prefix and the corresponding BGP `next-hop` as communicated by its peer. The BGP receiver separately dumps data concerning the BGP RIB and successive BGP updates.

The *commands generator* processes data dumped by the BGP receiver in order to generate the commands to be sent to the router application. We use a single firewall filter containing several terms to implement the counting rules.

In order to avoid redundancy and lower the number of terms as much as possible, the commands generator reduces the size of the BGP RIB using a Patricia Trie data structure and an algorithm similar to that described in [5]. Among optimizations performed, only one counting rule, matching the 0/0 IP prefix, is added for the BGP `next-hop` associated to the highest number of IP prefixes. Since that rule is installed on the router as the last-evaluated counting rule, it matches only the traffic that is not destined to any other BGP `next-hop`. Notice that the correctness of the traffic count is not affected by the usage of the 0/0 IP prefix since we apply counting rules on output traffic, i.e., traffic that is not routed by the device does not increment any counter. The Patricia Trie is also exploited to elaborate successive BGP updates, avoiding the generation of unnecessary commands. For instance, a command is not gen-

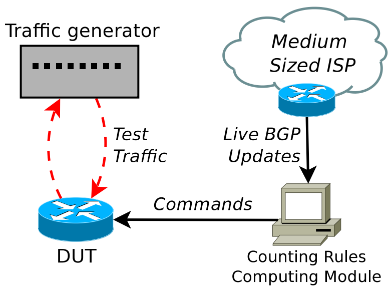

Figure 3: Testbed topology.

erated for a BGP update that specifies an association between an IP prefix and a BGP `next-hop` if the association between a less specific IP prefix and the same BGP `next-hop` is already known. Generated commands are finally collected for a tunable time interval, in order to send them to the router application in groups and to eliminate redundant ones (e.g., sequences of add and delete operations for the same term).

To implement the commands generator, we developed few Perl and Bash scripts that run in pipeline.

5. EVALUATION

In this section, we experimentally evaluate the viability of our approach. We performed a preliminary study on the load our solution imposes on routers and we estimated the accuracy our prototype is able to achieve. We repeated each experiment several times and we present average results in the following.

The topology of the testbed we used in all the experiments is depicted in Fig. 3. We deployed the router application presented in Section 4 on an M7i Juniper router. The device is equipped with an old generation of Forwarding Engine Board (FEB) which is not designed to support a high rate of rule updates. In the following we refer to the M7i router as the device-under-test (DUT). We connected the DUT to a traffic generator, a Smartbits 600B. The traffic generator generated and collected IP traffic traversing the DUT, which was configured to route all packets back to the traffic generator. Since the DUT is equipped with only two FastEthernet interfaces, we were not able to measure performance (e.g., switching throughput) degradation when the device is stressed with huge amounts of traffic. However, since the CM is implemented in hardware in the DUT and the majority of the operations performed by the router application are CPU-bound, we expect the throughput of the DUT to be marginally affected by our solution. Our prototypical CRCM was installed on commodity hardware (a desktop PC equipped with a dual-core 2.4 GHz CPU and 4G of RAM). It was configured to receive the full BGP RIB and a stream of real-world BGP updates from the border router of a medium-sized Italian ISP and to send the corresponding commands to the router application inside the DUT.

In the first experiment, the CRCM generated the commands associated to the full BGP RIB (consisting of more than $310,000$ IP prefixes) of the medium-sized ISP and the corresponding counting rules were installed inside the DUT, sending all the commands in a single group. Using our optimizations, we needed only 408 counting rules, each matching 2000 IP prefixes at most, to represent the entire BGP RIB. The total number of counters is 370. The generation of all the commands took about 8 seconds and the installation of the whole set of counting rules on the device took less than 40 seconds, during which the control plane CPU usage at the DUT was about 90%. Observe that, the BGP RIB must be processed only once (for each reboot), this computation is independent from the number of peerings (depending only on RIB entries), and forwarding capability of the router should not be significantly affected.

We then checked that counters were correctly installed and properly worked, by configuring the traffic generator to send traffic (at top speed for a FastEthernet interface) destined to several IP prefixes and comparing collected data with the values of the counters on the DUT. In all the tests we run, we found that every packet sent by the traffic generator incremented the right counter, i.e., the counter associated to the BGP `next-hop` matching the IP `destination address` of the packet (as specified in the RIB).

As a second experiment, we configured the CRCM to run for 5 consecutive minutes, sending commands to the DUT every 10 seconds. Such commands corresponded to the processing of both the initial RIB and the stream of BGP updates received from the ISP's border router. The CRCM always spent few millisecond to process BGP updates. The DUT control plane used no more than 3% additional memory and the control plane CPU usage was always less than 45%, with spikes of $35\% - 40\%$ typically for about 7 seconds.

Finally, we measured the responsiveness of our prototype to rule updates. We configured the traffic generator to generate only traffic destined to specific IP prefixes. While traffic traversed the DUT, at a certain time we sent a group of 17 ad-hoc commands, which represents one of the bigger groups practically generated by the CRCM in the previous experiment. Each command in the group modified a term containing several IP prefixes. Among those modifications, commands replaced the counters associated to the IP prefixes traffic is destined to, simulating changes in the BGP `next-hop` associated to such prefixes. By comparing actual values of the counters with their expected values, we found that the DUT took about 8 seconds for replacing counters. Indeed, old counters, i.e., counters associated to monitored prefixes before changes required by ad-hoc commands, are incremented for about 5 seconds more than expected, while a number of packets corresponding to about 3 seconds is not accounted by any counter. In our tests, the time taken by the DUT for replacing counters was independent both on the number of commands in the group and on the amount of traffic the DUT was loaded with.

We then estimated the accuracy our prototype can achieve, computing the maximum error made on single counters $E_{max} = (t_{inst} + t_{group})/t_{inter}$, where t_{inst} is the time took for installing rules at the DUT, t_{group} is the time interval configured at the CRCM, and t_{inter} is the inter-arrival time between consecutive BGP updates that changed the BGP `next-hop` associated to the counter. Exploiting the BGP peering with the medium-sized ISP, we collected data in different days between July 20th and August 11th. We found that for the 98% of IP prefixes, the 90th percentile of BGP updates that contain a change of the BGP `next-hop` has an inter-arrival time always higher than 15 minutes, resulting in a maximum error of 0.89% if commands are sent to the DUT as soon as they are generated and of 2% if commands are grouped for 10 seconds. Since it is shown [13] that routing changes rarely affect large amounts of traffic, we expect average errors on TM to be much smaller.

We consider experimental results on our prototype promising. Also, we expect the responsiveness of our solution to be improved by usage of a new generation FEB and optimization of our prototypical software.

6. CONCLUSIONS

We believe that router programmability will provide fundamental support for network management in the near future, paving the way for a new generation of innovative monitoring solutions.

In this paper, we showed the feasibility of a distributed architecture for the accurate computation of traffic matrices. The architecture is based on programmable routers which autonomously compute different parts of the traffic matrix. We realized a prototypical implementation of our architecture using current technologies and preliminary experiments we performed are promising.

We plan to improve our prototype and better evaluate our solution through deeper performance tests on up-to-date hardware, with the purpose of assessing practical deployability of our architecture in production environments.

7. ACKNOWLEDGMENTS

We are grateful to anonymous reviewers for their feedback. Juniper provided part of the hardware for the experiments and technical support for the development of the prototype. We thank Consortium GARR for providing routing data.

This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, FIRB project "Advanced tracking system in intermodal freight transportation" and under Project "ALGODEEP: Sfide algoritmiche per elaborazioni data-intensive su piattaforme di calcolo emergenti", MIUR PRIN.

8. REFERENCES

[1] Cisco IOS NetFlow Version 9 Flow-Record Format. Official Cisco Documentation, Cisco Systems, Inc. `http://www.cisco.com`.

[2] Bates, T., Chen, E., and Chandra, R. BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP). RFC 4456, 2006.

[3] Bhattacharyya, S., Diot, C., Jetcheva, J., and Taft, N. POP-level and Access-link-level Traffic Dynamics in a Tier-1 PoP. In *Proc. IMW '01*.

[4] Case, J. D. and Fedor, M. and Schoffstall, M. L. and Davin, J. Simple Network Management Protocol (SNMP). RFC 1157, 1990.

[5] Draves, R. P., King, C., Venkatachary, S., and Zill, B. D. Constructing Optimal IP Routing Tables. In *In Proc. INFOCOM '99*.

[6] Feldmann, A., Greenberg, A., Lund, C., Reingold, N., Rexford, J., and True, F. Deriving Traffic Demands for Operational IP Networks: Methodology and Experience. *IEEE/ACM Trans. on Networking 9* (2000), 265–279.

[7] Griffin, T., and Wilfong, G. T. On the Correctness of IBGP Configuration. In *Proc. SIGCOMM '02*.

[8] Kelly, J., Araujo, W., and Banerjee, K. Rapid Service Creation using the JUNOS SDK. In *Proc. PRESTO '09*.

[9] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J., and Jahanian, F. Internet Inter-Domain Traffic. In *Proc. SIGCOMM '10* (2010).

[10] Medina, A., Taft, N., Salamatian, K., Bhattacharyya, S., and Diot, C. Traffic Matrix Estimation: Existing Techniques and New Directions. In *Proc. SIGCOMM '02*.

[11] Papagiannaki, K., Taft, N., and Lakhina, A. A Distributed Approach to Measure IP Traffic Matrices. In *Proc. IMC '04*.

[12] Rekhter, Y., Li, T., and Hares, S. A Border Gateway Protocol 4 (BGP-4). RFC 4271, 2006.

[13] Rexford, J., Wang, J., Xiao, Z., and Zhang, Y. BGP Routing Stability of Popular Destinations. In *Proc. IMW '02*.

[14] Sardella, A., Devarasetty, M., Eccli, J., and Kharitonov, D. Efficient Scaling for Multiservice Networks. White Paper, Juniper Network, Inc. `http://www.juniper.net`.

[15] Tootoonchian, A., Ghobadi, M., and Ganjali, Y. OpenTM: Traffic Matrix Estimator for OpenFlow Networks. In *Proc. PAM '10*.

[16] Uhlig, S., Quoitin, B., Lepropre, J., and Balon, S. Providing Public Intradomain Traffic Matrices to the Research Community. *SIGCOMM Comput. Commun. Rev. 36*, 1 (2006), 83–86.

[17] Varghese, G., and Estan, C. The Measurement Manifesto. *SIGCOMM Comput. Commun. Rev. 34*, 1 (2004), 9–14.

[18] Xiao, X., Hannan, A., and Bailey, B. Traffic Engineering with MPLS in the Internet. *IEEE Network Magazine 14* (2000), 28–33.

[19] Zhang, Y., Roughan, M., Duffield, N., and Greenberg, A. Fast Accurate Computation of Large-Scale IP Traffic Matrices from Link Loads. *SIGMETRICS Perform. Eval. Rev. 31*, 1 (2003), 206–217.

[20] Zhao, Q., Ge, Z., Wang, J., and Xu, J. Robust Traffic Matrix Estimation with Imperfect Information: Making Use of Multiple Data Sources. *SIGMETRICS Perform. Eval. Rev. 34*, 1 (2006), 133–144.