Int J Parallel Prog
DOI 10.1007/s10766-016-0469-7

Pathfinder: Application-Aware Distributed Path Computation in Clouds

Hai Jin[1] · **Aaqif Afzaal Abbasi**[1] · **Song Wu**[1]

Received: 30 September 2016 / Accepted: 26 October 2016

Abstract Path computation in a network is dependent on the network's processes and resource usage pattern. While distributed traffic control methods improve the scalability of a system, their topology and link state conditions may influence the sub-optimal path computation. Herein, we present Pathfinder, an application-aware distributed path computation model. The proposed model framework can improve path computation functions through software-defined network controls. In the paper, we first analyse the key issues in distributed path computation functions and then present Pathfinder's system architecture, followed by its design principles and orchestration environment. Furthermore, we evaluate our system's performance by comparing it with FreeFlow and Prune-Dijk techniques. Our results demonstrate that Pathfinder outperforms these two techniques and delivers significant improvement in the system's resource utilisation behaviour.

Keywords Cloud computing · Virtualization · SDN · Network · Topology inference · Path computation

1 Introduction

Cloud computing, which represents the long-held dream of computing as a utility, has the potential to transform a large part of the IT industry, including making the software even more attractive as a service and shaping the way IT hardware is designed

✉ Aaqif Afzaal Abbasi
aaqif@hust.edu.cn

[1] Services Computing Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China

and purchased [1]. The success of any cloud management software thus critically depends on the flexibility, scale, and efficiency with which it can utilise the underlying hardware resources while providing necessary performance isolation [2–5]. Thus, resource management on the cloud scale requires the management platform to provide a rich set of resource controls that can balance the QoS requirements of tenants with the overall resource efficiencies of datacenters (DCs). In order to clearly understand the current rigid DC architectures, it is necessary to also understand the characteristics of the network links connecting them.

In general, there are two serious issues in topology inference and distributed path computation: (1) The use of inconsistent information across nodes, resulting in the formation of transient routing loops and forwarding tables [6]; (2) The lack of opaqueness to link states and traffic information [7], which contributes much to the cloud's flexibility but hinders the efficient execution of distributed applications that require the exchange of large amounts of data among the virtualized infrastructure and resources.

In light of the above, an interesting solution is required that can integrate both conventional and state-of-the-art path computation schemes and reduce the complexity related to configuring and operating the cloud infrastructure. A solution providing comprehensive information on the status of the network's nodes could be expected to assist in achieving higher efficiency while incurring only a minimal penalty on the network resource utilisation.

1.1 Overview of Our Approach

In this paper, we utilise network topology inference (sometimes called network tomography) in our development of Pathfinder, a model framework that enhances the efficiency of topology inference and discovery in virtualized environments. Previous contributions and surveys on network tomography [8] principally suggested developing estimations on given network statistics, whereas Pathfinder exploits SDN functionalities to bring real-time network-wide abstraction under topology control and to relate it to path computation and service level agreement (SLA) requirements. SDN [9] is a technology that separates the control plane of a network from its data plane and enables programmable network behaviour. This assists in reducing the complexities of understanding complex virtualised environments that work beyond static code visualisations.

Instead of implementing a range of policies (to manage information-flow) or using static analysis, we develop an application program interface (API) based solution similar to [10] to administer topology functions. Although we describe our prototype system in a fair amount of detail, it should be noted, that only limited parameters are considered in developing Pathfinder. In addition, the overheads incurred by Pathfinder are ignored during the experimentation. This presents a limitation of the current development phase.

1.2 Contributions of the Paper

In summary, the paper makes the following contributions:

- We discuss the importance of network topology inference.
- We highlight the dependencies that influence topology inference behaviour.
- We present Pathfinder model and elaborate its features.
- We implement and evaluate a prototype of the Pathfinder model.

1.3 Outline

The rest of the paper is organised as follows. Section 2 outlines the related work on topology inference and path-computation schemes. In Sect. 3, we give an overview of dependencies in distributed path optimisation features and functions. In Sect. 4, we present Pathfinder's design, including its orchestration environment, topology inference model and path allocation scheme. Section 5 provides comprehensive details about Pathfinder's implementation. Finally, Sect. 6 concludes the paper.

2 Related Work

2.1 Design Considerations in Topology Inference

End-to-end path monitoring and restoration techniques are commonly used in data networks to ensure better performance and recovery from network link failures. The following points need to be considered during the path allocation procedure.

- Ensure that the recovery path for a network can be computed alongside its working path (for an easy and quick restoration of services).
- Ascertain that the recovery path has enough available resources.
- Identify the roles of ingress and egress nodes beforehand (in the case of node failure). Previous studies [8,11–14] addressed network path computation and topology administration challenges by using end-to-end estimations and network characterization approaches.

2.2 Network Characterization

In network characterization, a network's topology-related information is collected through its internal resources. This is a widely-adopted strategy and is explained briefly in literature [15–17]. An interesting aspect of network characterization is highlighted in [18], where the authors propose a general framework for the construction of a metric induced model based on network characterization. The framework was metric-induced network topologies (MINT). MINT employs network characterization concepts by integrating different topologies obtained at different points in time and from different network vantage points.

With complete link resource information available in a centralised path computation module (PCM), the computation for a shared mesh restored path is a NP-complete problem if minimization of the total capacity usage (working and restoration) is sought [8]. It is therefore desirable that the path computation algorithms use the input information as little as possible to compute the paths, with no penalty (or just a small penalty) in terms of capacity efficiency [19]. We, therefore, in this paper, use a combination (our own developed API and a link state database dependent API) approach to communicate and modify the topology state behaviour, as illustrated in Fig. 2. In summary, the discussed end-to-end estimation and network characterization techniques offer different trade-offs in term of network topology administration, topology inference and restoration guarantee under single and/or multiple network failures. This enables network operators to choose their desired protection and restoration mechanism based on the performance requirements of the end user applications.

3 Dependencies in Distributed Path Optimization Functions

Service dependencies in a virtualized environment are very rigid and complicated. From a development point-of-view, each service component of traffic function depends on a bunch of service components in logic. While on the other hand, from a deployment point-of-view, the dependencies must overcome the limitations to satisfy the desired QoS levels. Based on [8], and our experience, we consider the system dependencies of topology inference in the following four major areas.

- *Enforcing optimisation for large production networks* Operations performed through the path computation element (PCE) provide information not only related to the link/nodes states, but also provide information on the active network connections. Carefully engineering these statistics can improve the best (i.e. optimised) path selection in networks.
- *Path maintenance* When a connection is lost due to a link failure, path optimisation takes places using distributed intermediate variables (DIVs) based mechanisms. This helps the system to tolerate data traffic (packets) reordering and losses.
- *Overlapping updates* In cases where multiple overlapping updates occur, cost-to-destination schemes can be applied for result selection. It is therefore necessary to ensure a suitable mechanism whereby, the traffic nodes only respond to the latest or best update.
- *Topology inference* Topology identification takes into account the global information related to network resources and paths and works by probing a network's internal characteristics, using information derived from each node. It is therefore necessary to improve the topology inference models to reduce network's link loss rates.

The techniques discussed in [18] employ time-dependent link costs for routing but are not applicable to our experimental settings. Moreover, such approaches are not applicable to very-large scale data-centric environments. Pathfinder improves topology inference and control in the following two ways

- By exploiting software-defined principles in data centres.
- By adopting a SLA-aware approach to administer topology functions.

4 System Design

Pathfinder incorporates SDN features in distributed path computation process, which brings two major benefits:

– It brings flexibility to introduce new ideas into the network through a software program, as it is easier to change and manipulate a software program than to use a fixed set of commands with proprietary network devices.
– It simplifies network configuration, as opposed to more complex conventional distributed management systems. In SDN-enabled networks, operators do not have to configure all the network devices individually but instead can make network-wide traffic-forwarding decisions in a logical single location, i.e. the controller, with global knowledge of the network state.

4.1 Orchestration Environment

We develop a set of programmable APIs to manage an already existing cloud infrastructure. Pathfinder implements an integrated orchestration of inter-DC/ cloud connectivity by allowing operators to re-configure network management control functions. In Fig. 1, we highlight the contribution of Pathfinder's role (in red colour) in handling the path computation process, where it updates the access link state information database to implement the desired switching and routing functions on SDN-enabled hardware equipment.

Pathfinder's orchestration process revolves around four major actors, i.e. PCE, API, the link state database and (SDN-enabled) switching elements. The process takes place in following steps

– *Step 1* A topology request is sent for PCE computation. This request is mirrored and forwarded into the link state database—which is a collection of node links and their states—through an API plug-in. The link state database evaluates the request for any inconsistency.

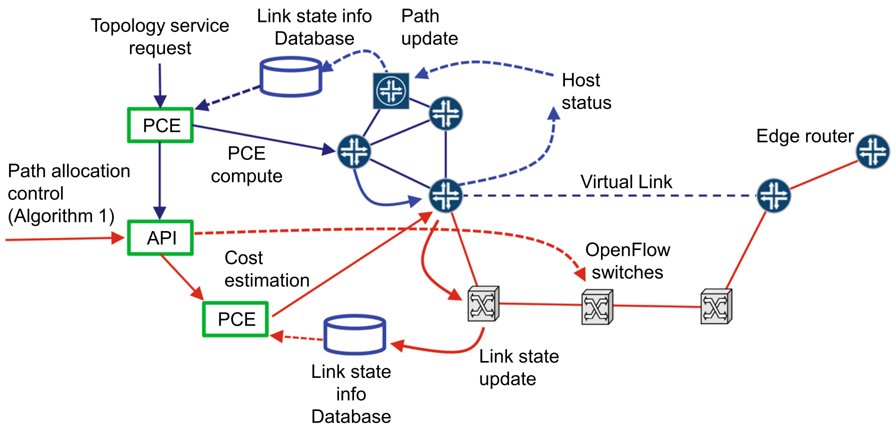

Fig. 1 Implementation scenario (Color figure online)

Algorithm 1 Resource-aware path allocation scheme

```
1: Input: $PQ$ : $PrecedenceQueue$ (the queue of pending topology requests); $R[i]$: the exit time of $i^{th}$
       topology request; $REQ$: a topology request; $REQ_c$, $REQ_m$ and $REQ_{bw}$: requested CPU, memory
       and bandwidth resources; Available CPU, memory and bandwidth resources denoted by $C$, $M$ and
       $BW$ respectively.
2: Output: Path allocation decision
3: /* The $PQ$ queue accumulate a domain's pending and rejected requests */
4: while $PQ_i$ != $NULL$ do
5:   sort the requests of $PQ_i$ in ascending order of $R_t$
6:   for $REQ \in PQ_i$ do
7: /* To ensure that available resources are enough to handle upcoming requests */
8:     if $REQ_c \leq C \&\& REQ_m \leq M \&\& REQ_{bw} \leq BW$ then
9:       Allocate desired resources and activate request ($REQ$)
10:    else
11:      return computation failure (resources insufficient)
12:    end if
13:    Repeat
14:  end for
15:  switch to the next $PQ$
16: end while
```

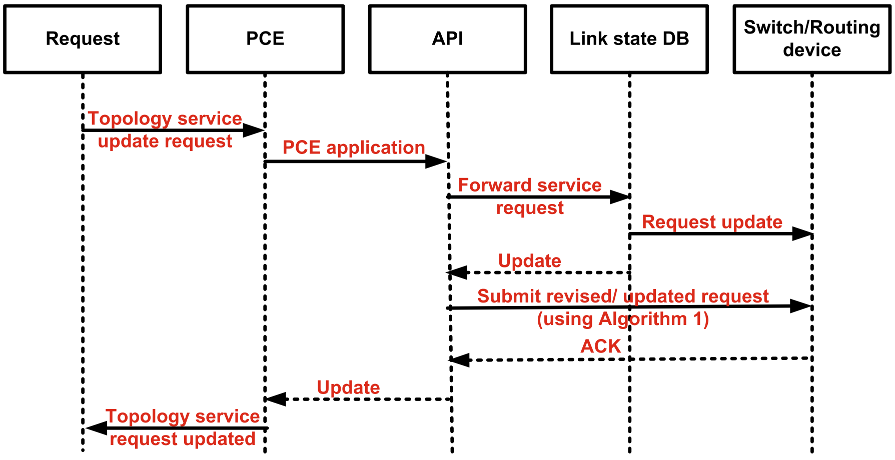
Request
PCE
API
Link state DB
Switch/Routing device
Topology service update request
PCE application
Forward service request
Request update
Update
Submit revised/ updated request (using Algorithm 1)
ACK
Update
Topology service request updated

Fig. 2 Pathfinder—workflow process

- *Step 2* The accepted request is replied with an acknowledgement.
- *Step 3* The API plug-in reads the latest switch update information from the link state database and assigns the best available route to the request. This function is performed through Algorithm 1.
- *Step 4* The processed request is sent back to the receiver and an acknowledgement of the same is updated in the API plug-in's $successful - request$ vault. This workflow process is also illustrated in Fig. 2. Next, we discuss Pathfinder's topology monitoring and resource-aware path allocation schemes.

Table 1 Topology control functions and their short description

Component	Function	Description
Application handler	Invoke services	By mapping dependencies, the application handler selects topology requests and sends them onward for SLA compatibility check
Service handler	SLA compliance	The service handler performs SLA compatibility check. Each topology request is controlled by the service handler through its handler function. All services must accept and process the control code
Path computation	Path identification	First, a check is made that the system has sufficient resources for executing a process. Then PCE is used for path computation so that the appropriate network devices can take part in executing the process

4.2 Monitoring Approach

A topology manager provides an open choice to administrators for fulfilling their monitoring needs in a combination of ways, including the use of cloud platform monitoring features, third party monitoring services, or customised monitoring components. Topology manager also addresses application deployment on the basis of SLA requirements and resource availability.

In Pathfinder, the monitoring process is initiated following a process or application request for the granting of resources. The Application handler checks the process for SLA violations. If SLA violations are found, the process is rejected. If there are no SLA violations, the request is forwarded to the Service handler, which ensures that the process has enough resources for execution. Process requests with insufficient resources are dropped out (rejected), while those with sufficient resources are reserved for the PCE process. The PCE defines a suitable route for conveying data between the request source and its destination for the accepted requests. By using PCE, the overall processing overhead is reduced as PCE reuses the already calculated paths (using path caching) to improve routing performance. This monitoring approach therefore attempts to identify the bottlenecks and investigate SLA violation issues efficiently. A short description of inference model functions is presented in Table 1.

4.3 Resource-Aware Path Allocation Scheme

Let us consider a case where an SDN controller sits atop a domain of programmable switching elements to oversee their topology patterns. The path allocation function can be developed by forming a precedence queue (PQ). The PQ contains a list of all (i.e. pending and rejected) topology service requests of an individual network domain. This allows model descriptions to be developed for different kinds of workloads, such as topology patterns and management plans. The term *pattern* here refers to a description of components of a service, i.e. an application, its required resources and their relationship. The proposed resource-aware path allocation scheme performs two major tasks.

- It sorts the received topology requests from a domain in increasing order of their burst time.
- It ensures that the network has enough resource to entertain the topology requests.

In Pathfinder, resource requirements of a topology request are defined as a composite of the CPU, memory and bandwidth resources. Algorithm 1 presents our proposed resource-aware path allocation scheme. The developed scheme also corresponds to the case of virtualized infrastructure managers providing an integrated orchestration of IT resources.

5 Implementation and Evaluation

5.1 Scenarios and Compared Schemes

We take into account the following two considerations in our implementation case scenarios.

- *Path isolation* For path isolation in virtualized network environments, it is assumed that enterprise traffic is handled at "global table" of a network.
- *Overlay design* Virtualized infrastructures constitute an overlay design (rather than employing a "rip-and-replace" approach), which means that network deployment of network virtualization should take place without impacting (or just a limited impact on) the network design that tenants already have in place.

We perform path computation on a five node topology mesh and compare Pathfinder's performance with the Freeflow [20] and Prune-Dijk [21] techniques using following cases.

- *Case 1* Measure optimised traffic flow behaviour on a pre-computed route.
- *Case 2* Measure traffic flow behaviour on a resilient path.

5.2 Experimental Settings

We consider a controlled environment for Pathfinder and use a limited number of packet forwarding elements mapped to transmit data traffic over a network. The data traffic in our experiments is controlled by a SDN-enabled EPC Gateway [22], which is itself controlled by an API. The inter-node link distance is pre-computed and the shortest path calculated is measured as the path-cost between the source and destination nodes. We develop the API using RealCloudSim (v9.8). The console prototype is deployed on an Intel Xeon CPU E5-2650 v3, with a clock speed of 2.30 GHz, 25 MB Smart cache and 10 cores (20 threads). The test bed comprises SDN-enabled Huawei SN640 switches (OpenFlow v.1.3) and EPC Gateway agent. The agent is implemented to administer the routes. If required, it could also selectively break out the traffic streams (Fig. 3).

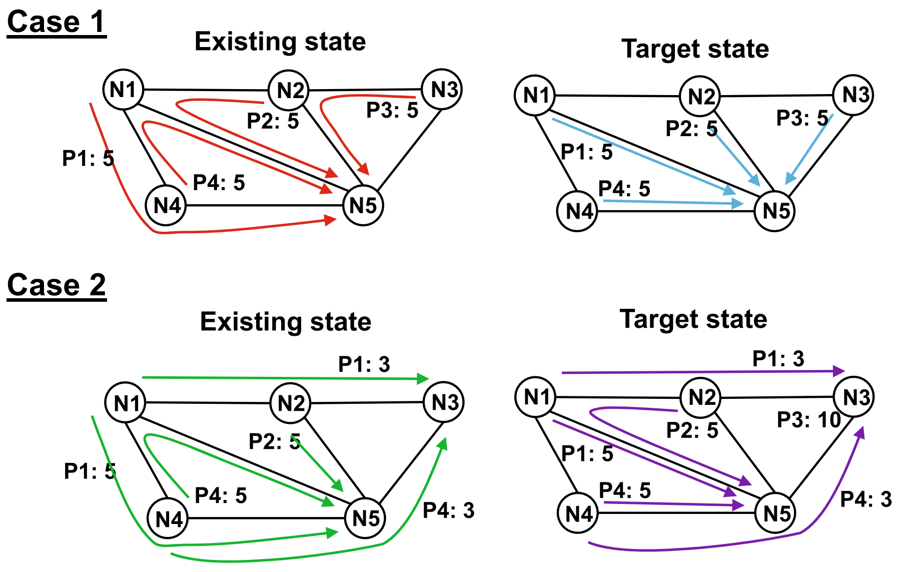

Fig. 3 Topology map cases

Table 2 Average preprocessing time (in seconds)—Case 1

Batch no.	Freeflow	Prune-Dijk	Pathfinder
1st	17.45	16.92	16.40
2nd	17.21	16.67	16.93
3rd	17.62	16.83	16.42
4th	17.03	16.76	16.20
5th	17.43	16.89	16.34

Table 3 Average preprocessing time (in seconds)—Case 2

Batch no.	Freeflow	Prune-Dijk	Pathfinder
1st	14.36	11.61	10.83
2nd	14.21	11.46	10.92
3rd	14.06	11.33	10.57
4th	14.11	11.72	11.03
5th	14.18	11.41	10.87

5.3 Results and Discussion

These experiments are aimed at evaluating the following:

– Average pre-processing time for each case (Tables 2, 3).
– Number of node visited for each path computation technique (Table 4).
– Service response time for each case (Fig. 4a, b).
– CPU time (Fig. 5a).
– Memory utilisation (Fig. 5b).

Table 4 Number of node visits

Methodology	No. of node visits	Path cost (in seconds)
Freeflow	43	16.21
Prune-Dijk	32	13.46
Pathfinder	28	10.33

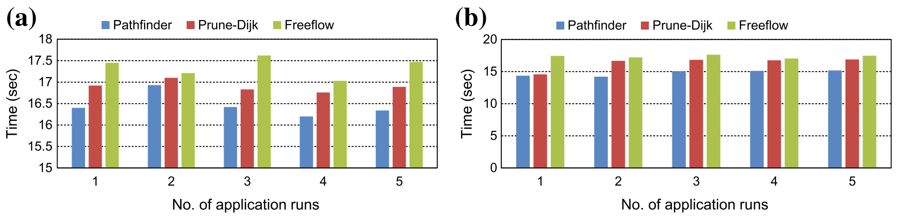

Fig. 4 Response time analysis. **a** Response time (Case 1). **b** Response time (Case 2)

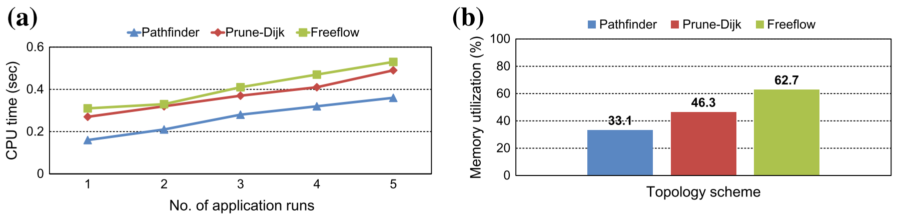

Fig. 5 Resource utilisation analysis. **a** CPU time comparison. **b** Memory utilisation

We perform simulation by first launching five batches of 30 requests under both test case scenarios, Case 1 and Case 2 (Fig. 3). We begin by first calculating the pre-processing time of requests. The pre-processing time of all the three compared path computation strategies (i.e. Pathfinder, Freeflow and Prune-Dijk) keeps on fluctuating. However, we observe a considerable improvement in Pathfinder's time. We believe this improvement is a result of the pre-computed paths, which is only possible in our technique due to the flexibility brought in by the programmable API plug-in feature.

We calculate and compare Pathfinder's pre-processing time with Freeflow and Prune-Dijk using both Case 1 (Table 2) and Case 2 (Table 3). We also evaluate and compare the number of node visited per successful execution (Table 4). For service response time, we perform a set of experiments using trace-driven simulation. In our experimentation process, service response time of a query is calculated as the time period from the query is issued until the source node receives a response result from the first responder (node). Finally, we perform CPU time comparisons among the three compared techniques and find Pathfinder has better CPU and memory usage statistics. We believe that the application-awareness brought in by the SDN concept and the introduction of a separate PQ for pending requests creates room for incoming services to be treated on the basis of resource availability.

Results demonstrate that adopting an application-aware approach can not only lessen the node visited and checked but can also help in reducing the CPU (Fig. 5a) and memory utilisation of the system (Fig. 5b). Contrariwise, CPU resources can be overburdened if visits are made strictly on the basis of pre-determined paths. Basically, by adopting a continuous look and update policy by Pathfinder through resource-aware path allocation improves path computation constraints in a simple yet effective way, which resulted in reduced system CPU utilisation and memory utilisation.

6 Conclusion and Future Work

In this paper, we present Pathfinder, an application-aware distributed path computation model. By fine tuning the data traffic structure between the traffic flow source and SDN-enabled destination switches, Pathfinder attempts to minimise the network resource utilisation. We start the paper by briefly discussing the varying behaviour and importance of network topology inference in virtualized environments. By highlighting the role and importance of application-awareness (using SDN as the enabling technology) in dealing with path computational dependencies, we present our system's design and briefly explain its topology inference model and resource-aware path allocation control policy. We compare Pathfinder's performance with the Freeflow and Prune-Dijk techniques. The obtained results demonstrate that Pathfinder outperforms these two techniques when comparing them in terms of the pre-processing time and system resource utilisation (CPU and memory utilisation) performance.

In future, we plan to simulate Pathfinder's performance on real-time workload and by comparing it with several other state of the art topology inference schemes. We also plan to explore a dynamic solution to cope with the changing demands of topology requests.

Acknowledgements We thank Xinyu Zhao and Hao Fan for their feedback on data loads. This research is supported by National Science Foundation of China under Grant No. 61232008, National 863 Hi-Tech Research and Development Program under Grant No. 2015AA01A203 and the Fundamental Research Funds for the Central Universities under Grant HUST: 2016YXZD016.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Drepper, U.: The cost of virtualization. ACM Queue **6**(1), 28–35 (2008)
3. Greenberg, A., Hamilton, J.R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D.A., Patel, P., Sengupta, S.: Vl2: a scalable and flexible data center network. ACM SIGCOMM Comput. Commun. Rev. **39**(4), 51–62 (2009)
4. Shue, D., Freedman, M.J., Shaikh, A.: Performance isolation and fairness for multi-tenant cloud storage. In: Proceedings of 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12), pp. 349–362 (2012)
5. Battré, D., Frejnik, N., Goel, S., Kao, O., Warneke, D.: Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In: Proceedings of 2011 IEEE International Conference on Cloud Computing (CLOUD'11), pp. 17–24 (2011)
6. Ray, S., Guérin, R., Sofia, R.: Distributed path computation without transient loops: an intermediate variables approach. In: Lorne, M., Tadeusz, D., James, Y. (eds.) Managing Traffic Performance in Converged Networks, pp. 104–116. Springer, Berlin, Heidelberg (2007)
7. Leonardi, E., Mellia, M., Marsan, M.A., Neri, F.: Optimal scheduling and routing for maximum network throughput. IEEE/ACM Trans. Netwo. **15**(6), 1541–1554 (2007)

8. Qin, P., Dai, B., Huang, B., Xu, G., Wu, K.: A survey on network tomography with network coding. IEEE Commun. Surv. Tutor. **16**(4), 1981–1995 (2014)
9. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Commun. Mag. **51**(2), 114–119 (2013)
10. Abbasi, A.A., Jin, H., Wu, S.: A software-defined cloud resource management framework. In: Lina, Y., Xia, X., Qingchen, Z., Laurence, T.Y., Albert, Y.Z., Hai, J. (eds.) Advances in Services Computing, pp. 61–75. Springer International Publishing AG, Switzerland (2015)
11. Ni, J., Xie, H., Tatikonda, S., Yang, Y.R.: Efficient and dynamic routing topology inference from end-to-end measurements. IEEE/ACM Trans. Netw. **18**(1), 123–135 (2010)
12. Wei, W., Wang, B., Towsley, D., Kurose, J.: Model-based identification of dominant congested links. IEEE/ACM Trans. Netw. **19**(2), 456–469 (2011)
13. Ni, J., Tatikonda, S.: Network tomography based on additive metrics. IEEE Trans. Inf. Theory **57**(12), 7798–7809 (2011)
14. Zhao, Y., Chen, Y., Bindel, D.: Towards unbiased end-to-end network diagnosis. IEEE/ACM Trans. Netw. **17**(6), 1724–1737 (2009)
15. Jesus, V., Aguiar, R.L., Steenkiste, P.: Topological implications of cascading interdomain bilateral traffic agreements. IEEE J. Sel. Areas Commun. **29**(9), 1848–1862 (2011)
16. Snoeren, A.C., Raghavan, B.: Decoupling policy from mechanism in internet routing. ACM SIGCOMM Comput. Commun. Rev. **34**(1), 81–86 (2004)
17. Krishnamurthy, B., Willinger, W., Gill, P., Arlitt, M.: A socratic method for validation of measurement-based networking research. Comput. Commun. **34**(1), 43–53 (2011)
18. Bestavros, A., Byers, J.W., Harfoush, K.A.: Inference and labeling of metric-induced network topologies. IEEE Trans. Parallel Distrib. Syst. **16**(11), 1053–1065 (2005)
19. Liu, H., Bouillet, E., Pendarakis, D., Komaee, N., Labourdette, J.-F., Chaudhuri, S.: Distributed route computation and provisioning in shared mesh optical networks. IEEE J. Sel. Areas Commun. **22**(9), 1626–1639 (2004)
20. Zhao, L., Lai, Y.-C., Park, K., Ye, N.: Onset of traffic congestion in complex networks. Phys. Rev. E **71**(2), 026125 (2005)
21. Liu, G., Ramakrishnan, K.: A* prune: an algorithm for finding k shortest paths subject to multiple constraints. In: Proceedings of the Twentieth IEEE Annual Joint Conference of the Computer and Communications Societies (INFOCOM'01), vol. 2, pp. 743–749 (2001)
22. Sama, M.R., Contreras, L.M., Kaippallimalil, J., Akiyoshi, I., Qian, H., Ni, H.: Software-defined control of the virtualized mobile packet core. IEEE Commun. Mag. **53**(2), 107–115 (2015)