Neo-NSH: Towards Scalable and Efficient Dynamic Service Function Chaining of Elastic Network Functions

Sameer Kulkarni
University of Göttingen

Mayutan Arumaithurai
University of Göttingen

K.K. Ramakrishnan
University of California Riverside

Xiaoming Fu
University of Göttingen

***Abstract*—Middleboxes (Service Functions) have become indispensable part of Enterprise, Mobile and Data Center networks. Network operators rely on middleboxes to enforce network policies and to provide performance optimization, security and other value added services. With the increasing scale of network services and use cases demanding a specific sequence of service functions, the complexity and scale requirements of enforcing the network policies and orchestrating the service functions have significantly increased. Hence, elastic scaling and dynamic service chaining of network functions is fundamental for the performance of Data Center and Enterprise networks. To this end, we propose *Neo-NSH*: an amendment to Network Service Headers (NSH) proposal. We re-purpose the 24 bit Service Path Identifier (SPI) to express the service-chain ID instead of representing the service paths. Our proposal extends on the key benefits of NSH and makes it more efficient and scalable in facilitating the dynamic service function chaining.**

I. Introduction

Middlebox functions such as firewall, intrusion detection, cache optimization, load balancing, *etc.* have become an integral part of large scale enterprise and data center networks. Typically, flows are subject to network-resident services that require one or more functions to be processed in sequence. Service Function Chaining (SFC) is the construct to describe the execution order of service functions [1].

Consider Figure 1 that illustrates two high level policies *i.e.,* i) for video traffic, the service function chain demands the traffic to traverse through Firewall and Video-optimizer service functions and ii) for the regular web traffic, Firewall and the value added services functions like anti-virus and parental control services are desired. In such cases, once the flows are classified, the flows need to be steered along two different paths to go through the respective policy desired service functions. Software Defined Networking (SDN) provides the flexibility in setting different flow rules across the switches and enables to steer flows accordingly. With Network Function Virtualization (NFV), the service functions can be provisioned to meet the traffic demands and thus many instances of network functions can be instantiated at will to ensure performance even at varying network loads.

SDN and NFV provide richer support for more fine grained traffic steering and demand instantiation of network functions. However, there are still several challenges in realizing the dynamic SFC with traditional routing constructs. Problem statement for SFC [2] lists out the key issues in realizing dynamic SFC with the traditional networks and illustrates the desired characteristics of dynamic SFC with elastic network

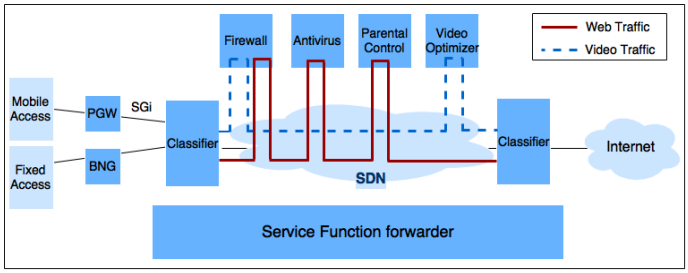


Fig. 1: SFC Use case for two different traffic classes

functions. Herein, we present the key issues and characteristics most pertinent to the scope of our work.

- Foremost is the topological independence, which aims at defining SFC in a way that mapping of policies to the desired service functions is independent of the topology and underlying network routing mechanism. This decoupling ensures the service policies to be more generic and easily deployable across different networks.
- Second, the capability to re-classify and update the SFC, which enables to change the course of traffic to traverse through different service types. This feature is highly beneficial for providing value added services like intrusion detection, deep packet inspection, and URL filtering.

Network Service Header (NSH) [1] with a dedicated service plane fills in these gaps and offers several other benefits like i) the ability to exchange meta-data across service functions in the chain through the context headers and ii) provides end-to-end visibility of the service path However, the cost and complexity in managing the path identifiers and updating the path information to the Classifiers and Service Function Forwarders (SFFs) that forward the traffic towards the service function instances also needs to be accounted. In order to scale and adapt at fine time scales, it is necessary to address and manage service paths more efficiently. We account for the control plane aspects pertaining to how the policies are mapped on to the network topology and service function instance specific Service Path Identifier (SPI) management and the role of control plane in managing and updating the SFFs whenever a new service function instance is added or an existing instance is taken down.

The key contributions of our work include:

- Illustration of the control plane functionality needed to support NSH. We argue that the NSH construct of SPI results in significant cost and complexity in control plane,

and we seek to simplify it.

- Our proposal Neo-NSH, that provides flexibility, ease of configuration and adaptability to instantiate/relocate the service functions with minimal control plane overhead.
- Preliminary evaluation that justifies the benefit of our proposal in terms of achieving significant reduction in the number of SPIs.

II. Related work

Over the past few years, several works [3]–[7] have proposed solutions for SFC. We briefly discuss a few that employ network overlay (MPLS, VLAN, VxLAN), underlay (overloading the existing L2/L3/L4 header fields), and alternate header based approaches.

A. SFC with Network Overlay and Underlay

Shadow-MACs [6] and OpenSCaaS [7] emphasize on utilizing the L2 address fields —media access control (MAC) address to represent the path identifiers. [6] utilizes the destination MAC addresses as opaque forwarding labels while [7] employs the source MAC addresses as a forwarding label to setup the service chain ID (SC-ID). In both cases, the SDN controller has the responsibility of managing (defining and mapping) the SC-ID and setting up the appropriate L2 address to steer traffic to the desired service instance. StEERING [3] also overloads the L2 address fields, to steer packets to inline service functions. It relies on multiple forwarding tables that require additional extension to the Openflow API. In addition, it requires all the service function instances in the network to be aware of the Ethernet address of all the service functions in the network topology, which limit the flexibility and scalability of elastic network functions.

These approaches provide efficient mechanism to steer the packets through the service function chain by re-purposing the existing packet headers fields, however they cannot support the exchange of meta-data and lack the ability to re-classify or alter the course of service functions after initial classification and assignment of the service function chain, thus inhibit the support for elastic network functions.

B. SFC with explicit tag and other alternatives

FlowTags [8] enable SFC by defining tag enhanced network functions, where the network functions generate and consume the service tags, while switches forward the packets based on the tags. SIMPLE [5] addresses efficient routing by constraining the number of switch forwarding rules and load balancing the traffic across middle-box instances and relies on the tag-based approach to tunnel packets across service functions. The computation and optimization of service paths is done through the mix of offline and online mixed integer linear programming, which results in considerable amount of computation time complexity. These approaches can facilitate re-classification and enable to alter the route through every service function, but as with earlier approaches, information exchange and sharing of meta-data is not possible. In addition these approaches need to account for additional complexity in tag management and distribution.

[9] and [10] propose Information Centric Networking (ICN) based approach of named services and named service instances for service chaining *i.e.,* the routing based on service function names and service instance names respectively. The network elements (routers and switches) take the responsibility of steering traffic to the desired service function instance. Both rely on network overlay/underlay mechanisms to tunnel flows to service instances. Though, the approach in [9] results in enormous reduction of switch rules, due to lack of fine grained control, it fails to provide visibility and control over appropriate service instance selection for a class of traffic, which is generally required for multi-tenancy, multi-subscriber policy matching and cannot cater to optimal instance utilization.

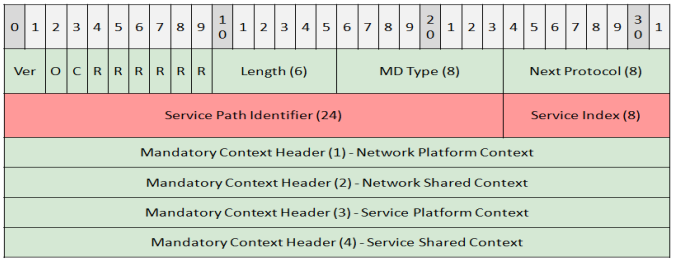


Fig. 2: Packet Structure of Network Service Header

III. NSH - Dedicated Service Plane for SFC

Network Service Header (NSH) [1] is an IETF draft proposed to address the Service Function Chaining (SFC) based on the SFC encapsulation to support the SFC architecture as outlined in the RFC7665 [11]. NSH defines the data plane header format that is used to create a dedicated service plane for realizing SFC. The NSH as shown in Figure 2 consists of the following fields:

- A 4-byte Base header consisting of Version, Flags, MD Type and Next Protocol fields. The next protocol field indicates the protocol type of the encapsulated data.
- A 4-byte Service Path Header consisting of a 24 bit Service Path ID (SPI) and 8 bit Service Index (SI), is used to define the service path that interconnects needed service functions.
- Meta-data context headers. The value of MD Type determines the context headers. If the value is 0x1, it consists of four mandatory 32-bit context headers as shown in Figure 2 or if the value is 0x2, this field is optional, consisting of variable length context headers.

SPI defines one of the possible instantiations —a logical path to sequence of specific service function instances of a SFC, while the SI indicates the location within the service path. In addition, NSH defines optional header fields that can carry meta-data information. The format of meta data is determined by the MD type field in the base header. However, it must be noted that NSH needs to be inserted onto encapsulated packets, *i.e.,* the actual transport/steering of packets in the network is based on the outer encapsulation.

A. Why NSH?

The NSH approach of specifying a dedicated service plane for service function chaining offers several benefits:

SFC Approach	Number of Unique Identifiers
Service ID	$\alpha\ Num.\ of\ Function\ Types\ (SFT)$
Service Instance ID	$\alpha\ Num.\ of\ Function\ Instances\ (SFI)$
Service Chain ID	$\alpha\ Factorial(Chain\ Length)$
Service Path ID	$\alpha\ Factorial(Chain\ Length)^{SFI}$

TABLE I: Identifier requirements for different SFC approaches

- NSH provides a transport and topology independent service forwarding framework. This decoupling enables the service plane to be realized as overlay service over the existing data plane without requiring any additional complexity and protocols at the data plane.
- NSH enables the ability to classify and re-classify the flows at each service functions. This enables to dynamically steer same flows across different service paths and enables to have more richer and finer policy control.
- NSH enables to exchange meta-data across service functions in chain through the context header fields. This aspect is beneficial for Gi-Lan/mobile use cases that can carry the subscriber ID and Tenant IDs across the chain to realize per-user/per-subscriber based policies.
- NSH also provides end-to-end service path visibility. This enables to monitor and troubleshoot service functions, which is critical for Operations and Management (OAMs) to support high availability and resiliency.

B. Where NSH falls short?

Despite the key benefits of NSH, it doesn't account for control plane overheads in terms of required orchestration and overlay management *i.e.,* label distribution as in Multiprotocol Label Switching (MPLS) and Label Distribution Protocol (LDP). We show that the SPI management with NSH is complex and the burden on the control plane affects the efficiency and scalability. Foremost, NSH is a SFC encapsulation, that is transport agnostic and requires an outer transport specific encapsulation to forward the NSH packet across the network. Control plane is responsible to manage this encapsulation. This however is customary function of a control plane even in the absence of NSH.

SFC control plane is responsible for constructing Service Function Paths (SFPs), translating SFCs to forwarding paths, and propagating path information to participating nodes to achieve requisite forwarding behavior to construct the service overlay. *i.e.,* It is up to the control plane to map the high level policies based on the network topology and service function instances in the network to specific Service Path Identifiers (management of SFPs) and in updating the SFFs about the SFP mapping, that can change over time with addition of new service function or deletion of existing services and instances.

In Section IV, we highlight the key challenges with NSH in terms of control plane responsibilities to manage and update the Service Path Identifiers and pitch towards an approach to mitigate the challenges in handling the SPIs.

IV. Problem Description

The key to realizing agile and elastic network functions is the ability to dynamically instantiate, remove and relocate the network functions. Any such activity would result in having either a new set of service paths or the invalidation of existing service paths. The current NSH draft defines a 24-bit Service Path Identifier (SPI) and 8-bit Service Index (SI). SPI defines one of the possible instantiations (a logical path to sequence of service functions that includes one of the several instances of each service function) for a given SFC, while the SI indicates the location within the service path. Typically the order of relation between service chains and service paths is $1:n$ and it grows exponentially [12]. Although 24 bits is large enough to accommodate any sets of possible service paths the complexity is in managing the SPI labels and updating labels to the Classifiers and different SFFs.

A. Control plane Functionality

In conjunction to the role of control plane listed in the SFC architecture [11], in order for NSH to realize a service plane, the control plane needs to perform at least the following tasks:

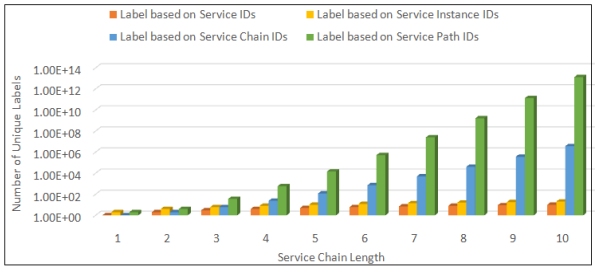
- For all the SFCs, construct the map of SPI (labels) needed for each valid logical path to the SF instance in the chain.
- Disseminate (communicate and update) the SPI information to the Service Function Forwarders (SFFs) and Service Functions (SFs).

Note that the SPI labels would change every time there is either an addition or deletion of as service function instance or changes to the physical topology.

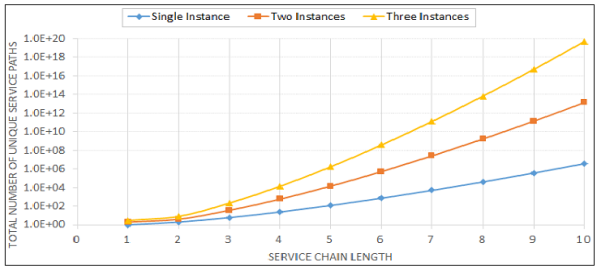
Though, it can be argued that topological changes are rather rare, and addition or removal of service functions too are rather infrequent, it must be noted that key benefit of NFV is in that it enables to realize elastic service function instances that can be dynamically instantiated or de-commissioned to better adapt to the traffic requirements and meet the SLA requirements from customer perspective and also improve on the overall network utilization. Hence the second aspect cannot be ignored for providing a truly elastic and dynamic service function chaining.

B. Control plane Overhead Analysis

From section II, we observe that the identification of service paths and the classification of different service paths and service chains can be broadly categorized into four different categories. i) that assign label for each service type and let the SDN controller to steer traffic in a service by service fashion to different service function instances in the network [9]. ii) that assign label for each service instance and let the SDN controller and SFFs to determine and steer traffic to different service function instances in the network [10]. iii) that assign the label for each service chain - a high level policy of desired service functions and let the network to employ some overlay/underlay to steer traffic across service functions [8]. iv) that assign the identifier to each of the logical service paths to the service function instances in the network [1]. The scale of required identifiers to define an end-to-end service function chain and the involvement of SDN controller vary for each of the approaches.



(a) Labels for different approaches with varying SFC length



(b) Service paths for varying SFC length and service instances

Fig. 3: Comparison of scale of unique labels and SPIs for different SFC approaches

We first present the scale of the number of unique identifiers (labels) required for the different approaches discussed earlier. From Figure 3a, we see that approaches that utilize the service or service-instance[1] based labels like FCSC [9], NSN [10], and source-routing [12] require a minimum number of labels, while the service-chain ID[2] and service path ID based approaches require a far larger number of labels. Table I shows the way in which the number of unique labels for each of the different approaches are determined.

We observe that the number of active service function instances (SFIs) affect on the number of identifiers required for the approaches relying on the service instance id and service path ids. In the latter case, we can see that number of identifiers scale almost exponentially, as each instance addition results in multiplier of factorial of new possible paths. Analytically, we can show that for addition of every service function instance to each of the service functions types the label requirements for a given chain length grows as shown in Table I. Also, note that from Figure 3b, with the increasing length of service chains and scale of service instances, the number of SPIs scales exponentially. This implies that burden on the control plane for performing the aforementioned tasks for any addition or deletion of service instances is significant. *i.e.,* with the increase in number of services in a chain and number of instances, the control plane overheads in managing and adapting to the increased number of paths, and dissemination of path information to each of the instances grows exponentially.

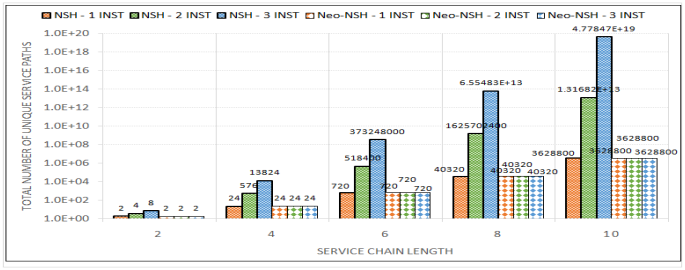


Fig. 4: Total Paths for varying chain length & instances/svc

V. Neo-NSH Proposal

We make the following fundamental observations:

- The service-chain IDs directly reflect the high-level specification of the policy intent.
- The number of labels required to support service IDs, and service-chain IDs is minimal compared to path identifiers.
- service ID base approaches are not affected by the service instance dynamics (i.e., addition, deletion and relocation).

Therefore, augmenting such a feature in NSH, naturally makes it a more robust candidate for implementing a dynamically scalable SFC. Also, we note that the real benefit of SPI (as purposed in NSH) is in providing the traceability and end-to-end visibility of the service function chains logical path.

We propose to relax the usage of SPI in its true sense, *i.e.,* to refer to the service paths and instead we propose to re-purpose the usage of 24-bit SPI field to denote the service-chain ID. This way SPI more closely reflects the intent of the service chain policy and represents the list of service functions and not the logical path to the service instances in SFC.

In Neo-NSH approach, the Service Forwarders (SFs) use the SPI and SI fields to represent the service function type rather than the service function instance, and let the network to dynamically choose the best instance based on the service function type(or service name) and the context data information. Service Function Forwarders (SFFs) that select the service path have to rely on either the control plane or the intelligent data-plane to choose the appropriate service instance. And, the role of service classification functions that update the service header is changed to inserting the service-chain ID, while the role of service functions and SFF is unchanged compared to the NSH.

The only down-side of our proposal Neo-NSH is the loss of end-to-end path visibility. The actual path *i.e.,* the list of physical SFs chosen for given SPI cannot be determined statically, as the same SPI could map to different logical paths (path to different service function instance) and physical paths. As the SPI is not static but determined at the runtime (meaning it can change dynamically), it provides an additional benefit of providing the ability to adapt to the network requirements dynamically.

Table II shows the feature comparison between NSH and Neo-NSH. We can observe that Neo-NSH retains all the key features of NSH.

A. Dynamic Service Function Instance selection

Typical to the SDN, whenever the flows are classified and service chain ID that determines the policy intent is derived,

[1]Service IDs correspond to number of unique services in the network, and service instance IDs to the number of instances for each service type. Figure 3a considers two instances per service function type.

[2]We acknowledge that SFC-IDs are mapped based on the required policy intents and are often much limited than actual possible combinations of service chains.

the controller must determine the appropriate service function instances and then ensure to set the forwarding rules at each of the forwarding elements so that the flow or class of flows traverse through the identified set of service instances. In-order to avoid the path setup latency, forwarding rules can be proactively setup at the forwarders and controller can modify the rules based on the network load in order to re-route and distribute the flows evenly across different service function instances. The key benefits of this approach over static path identification are: i) capability to load-balance the traffic across service function instances. ii) the addition/removal of service function instances does not affect the SPI. Hence the network becomes more agile and as well there is no need to compute and communicate the SPI labels to the SFFs.

TABLE II: Salient features of NSH and Neo-NSH

Features	**NSH**	**Neo-NSH**
Topological independence	✓	✓
Transport agnostic	✓	✓
Meta-data sharing and re-classification	✓	✓
End-to-End path visibility	✓	×
Flexible service instance selection	×	✓
SPI management overhead	High	Low
Communication overhead	High	Low

In Neo-NSH, by separating the logical service chain from actual service path, we can achieve significant improvements to NSH in terms of:

- Adaptability and efficiency: by reducing the control overheads in managing the path IDs and disseminating them to all the service instances.
- Flexibility: classifiers, service functions and proxies only need to care about logical service chain and not the service paths.
- Scalability: can easily accommodate more instances of service functions without impacting the SFFs.

VI. Evaluation

We performed preliminary evaluation of the proposal using the mathematical model to demonstrate the benefits of Neo-NSH. We compare Neo-NSH with base NSH approach. In this evaluation, we primarily focus on the demonstrating the benefit achievable in terms of the reduction in the number of service path identifiers in the case of increasing service chain length and increasing number of instances.

From Figure 4, we can see that in case of base NSH, the number of Service Path Identifiers scale exponentially with the increasing service chain lengths and also for a given service chain length NSH exhibits exponential growth with the increasing number of service function instances of the chain. With our proposal Neo-NSH, the number of SPIs increase only with the increasing service chain lengths Thus Neo-NSH in-comparison to NSH, not only results in significantly lowering the number of path identifier labels, but can support the elastic network functions wherein the network functions can be dynamically instantiated without any additional overhead of updating the SPIs to all the participating NSH aware components of the network.

VII. Conclusion

We have characterized and analyzed the benefits and challenges with the current NSH. We have proposed ***Neo-NSH***, an enhancement to NSH and demonstrated the benefits of our proposal which enables for a more agile and flexible network for service function chaining with elastic network functions. We seek to discuss and incorporate improvements on our proposal from the community. Next, we plan to prototype our solution using ONOS controller to evaluate and quantify the benefits of our proposal.

Acknowledgement

This research was funded by the EU FP7 Marie Curie Actions CleanSky ITN project Grant No. 607584 and the joint EU Horizon2020/NICT ICN2020 Project, Contract No. 723014 and NICT No. 184.

References

[1] P. Quinn and U. Elzur, "Network Service Header," Internet Engineering Task Force, Internet-Draft draft-ietf-sfc-nsh-10, Sep. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-sfc-nsh-10

[2] T. Nadeau and P. Quinn, "Problem Statement for Service Function Chaining," RFC 7498, Apr. 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7498.txt

[3] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patneyt, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula, "Steering: A software-defined networking for inline service chaining," in *2013 21st IEEE International Conference on Network Protocols (ICNP)*, Oct 2013, pp. 1–10.

[4] Z. Cao, M. Kodialam, and T. V. Lakshman, "Traffic steering in software defined networks: Planning and online routing," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. –, Aug. 2014. [Online]. Available: http://doi.acm.org/10.1145/2627566.2627574

[5] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *ACM SIGCOMM*, 2013.

[6] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow macs: Scalable label-switching for commodity ethernet," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '14. New York, NY, USA: ACM, 2014, pp. 157–162. [Online]. Available: http://doi.acm.org/10.1145/2620728.2620758

[7] W. Ding, W. Qi, J. Wang, and B. Chen, "Openscaas: an open service chain as a service platform toward the integration of sdn and nfv," *IEEE Network*, vol. 29, no. 3, pp. 30–35, May 2015.

[8] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing Network-Wide Policies in the Presence of Dynamic Middlebox Actions using FlowTags," in *NSDI' 14*, pp. 543–546. [Online]. Available: https://www.usenix.org/conference/nsdi14/technical-sessions/presentation/fayazbakhsh

[9] M. Arumaithurai, J. Chen, E. Monticelli, X. Fu, and K. K. Ramakrishnan, "Exploiting icn for flexible management of software-defined networks," in *ACM ICN*, 2014.

[10] S. G. Kulkarni, M. Arumaithurai, A. Tasiopoulos, Y. Psaras, K. K. Ramakrishnan, X. Fu, and G. Pavlou, "Name enhanced sdn framework for service function chaining of elastic network functions," in *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2016, pp. 45–46.

[11] J. Halpern and C. Pignataro, "Service function chaining architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015.

[12] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a flexible data center fabric with source routing," in *SOSR '15*, pp. 1–8. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2774993.2775005