HELM: Conflict-Free Active Measurement Scheduling for Shared Network Resource Management

Miao Zhang, Martin Swany, Adithya Yavanamanda, and Ezra Kissel
School of Informatics and Computing
Indiana University, Bloomington, IN 47405
Email: {miaozhang, swany, adiyavan, ezkissel}@indiana.edu

***Abstract*—Network resource measurement is a key functionality for large scale network management. Intelligent, network-aware applications may benefit from access to detailed representations of network resources, including multi-layer topologies and real-time traffic measurement, and shared resources may obtain better overall utilization by identifying performance bottlenecks. In this study, we describe a network measurement framework, which includes a network topology analysis mechanism as well as agent tools for running active probes and collecting data from end hosts. The system includes a centralized coordinator, which abstracts network elements into annotated network graphs and applies scheduling algorithms to calculate conflict free measurement probes over shared links. Our evaluation integrated perfSONAR services into our framework and included deployment scenarios on research and education networks such as Internet2 and ESnet. The data presented in this study offers compelling evidence that supports a method by which to measure the performance of real world networks.**

I. Introduction

Network performance is critical to the overall performance of cloud computing and service-oriented systems, and to the quality of experience of users accessing content. Monitoring and measurement are infrastructure services that quantify the operational status and performance of networks. Building and maintaining large scale networks, in conjunction with provisioning services that provide QoS or bandwidth services to users [11], requires accurate topology and performance metrics that are necessary to represent the current state of available resources. In recent years, projects like StarBED [24], GENI [3], together with other high performance research and education (R&E) networks, allow research projects to reserve their resources, configure network topologies as they wish and bring up and down services dynamically. At the same time, the physical hardware supporting these rapidly changing environments is finite, shared, and often multi-tenant through the utilization of virtualization technologies. Consequently, active resource monitoring is necessary for managing fair resource use across different user projects, and to ensure consistent user experiences.

This paper describes a network scheduling system called HELM, which is targeted at scheduling active network measurements. We focus on a particular monitoring task, namely the scheduling and execution of active measurement probes without conflict on the measured resources. In network measurements, probes have very different tolerances on resource contention. A user may not be concerned about the average latency of "ping" probes meant to simply verify connectivity, while other types of measurement may be very sensitive to interference [7]. For example, one-way delay measurement probes [30] can be affected by high-throughput performance probes running concurrently on the same network interface card. This problem of interference necessitates the need for scheduled probes that can reduce or eliminate conflicting resource usage. In general, scheduling problems are well-known as a computationally intensive task [19]. Studies have been done with various constraints on different types of shared resources, including CPU time allocation, register assignment [26], and processor scheduling [4]. However, the identification and resolution of conflicts in a multi-layer, complex network with virtualized and multi-tenant resources remains a unique and difficult challenge. Our approach must model the network topology in as much detail as possible and use this representation as a basis to reason about the problem.

In "The Sciences of the Artificial", Herbert Simon describes problem solving in design as transformation [32]. He observes that solving a problem can often be couched as transforming the problem representation so as to make the solution transparent. Even if this view is exaggerated, he suggests that we must consider how the problem representation contributes to a solution. In this work, we leverage our representation of the network topology, *UNIS* (described in Section III-A), to unambiguously describe the relevant topology. Then, based on our knowledge of the semantics of the network graph, we transform this representation so as to apply well-understood graph solutions. While we are not breaking new ground in graph processing, we note that our approach is based on a realistic, detailed model, and thus our solution is solving the "real" problem, rather than a simplified version.

In this article, we describe our approach for probe scheduling, examine the correctness and scalability of our approach, and perform some experiments on real network topologies. The main contribution of this study includes analyzing and constructing a topology across different traditional network layers to reveal the resource conflicts in the network; an algorithm mapping such a multi-layer, heterogeneous topology into an intersection graph and resolving the measurement schedule. Finally, we explore experiments on some real example networks. The remainder of the paper is organized as follows: Sections II and III describe some background and discuss the tools and techniques used in this study. Section IV is dedicated to the design and implementation of the HELM framework. The experimental data is presented in Section V and Section VI compares several related efforts and concludes.

II. Background

Depending on the technique, monitoring can be categorized in different ways. Here we focus on active monitoring [25], [2] in which traffic is generated to probe network behavior, as opposed to passive observations of network metrics. Active monitoring can monitor particular aspects within networks such as link performance or reachability of a specific device [35], [5]. Simple monitoring of a single network device is relatively straightforward, but the design and implementation of a complete network monitoring framework involves a number of design considerations and deployment challenges. The forwarding behavior, correctness, and expected performance of a network is a function of the interaction of many interconnected components, involving various site-specific policies and configurations, and many real-time network conditions that can affect overall operation.

Metcalfe's Law describes the "network effect" of increasing value as more devices or users become connected to a network [31]. A natural consequence of the network effect is an increase in the complexity and difficulty of monitoring and managing networks. The network effect applies to network monitoring systems as well. The more measurement vantage points that exist, the greater the value of the information. At the same time, more measurement points introduces more complexity in scheduling active measurements.

The situation for conflict-free measurement is becoming more complicated with the widespread use of dynamic virtual resource allocation. Cloud computing environments allow users to dynamically instantiate virtual machines across geographically diverse locations. With uncoordinated measurements between resources in these environments, it is possible that measurements conflict. For example, virtual machines measuring across the same physical interfaces concurrently and thus skewing the results. A similar situation arises within evolving Software Defined Networking (SDN) environments where multiple "flowspaces" are created and destroyed on-demand. Seemingly independent virtual links may traverse the same physical paths.

In the Global Environment for Network Innovations (GENI), and in similar experimental infrastructures worldwide, this problem is quite obvious. For empirical computer science research activities, users must be able to measure the infrastructure aggressively, but as many resources are virtualized and shared, the potential for perturbation of resources is extremely high. Our approach is largely motivated by these experimental use cases. In addition, large-scale R&E networks provide WAN connectivity for testbeds like GENI, and offer interfaces for customers to use dedicated bandwidth services directly. In our evaluation, we consider the Energy Sciences Network[1] (ESnet) and the Internet2 Network[2], the latter of which provides WAN connectivity for GENI. Our approach thus takes into account the use of such networks that "stitch" together resources within testbed experiments while also considering the scheduling of measurements over those same multi-layer networks as a common problem.

In the environments described above, the topology and connectivity of the networks being measured is known to a relatively high degree. The GENI, Internet2, and ESnet networks all advertise topology and resource information via global services within their communities. We note that this is an important distinction from commodity networks, or the general Internet, where detailed network connectivity is protected information and not often publicized. In either case, the network operators would have access to their own network topologies. Given the topology of the network, and the state of provisioned resources, our approach gives experimenters and operators the ability to understand the performance of the substrate and collect measurements for their virtual networks within a common framework.

[1]ESnet:http://www.es.net

[2]Internet2: http://www.internet2.edu

III. Periscope

Periscope is our implementation of the perfSONAR [14] network monitoring architecture. It is semantically compatible with the perfSONAR protocols and schemata, and backward compatible with the perfSONAR-PS implementation, while implementing additional functionality. The perfSONAR architecture implements components common to many measurement systems, including Measurement Points (MPs) and Measurement Archives (MAs) and a Lookup Service (LS). The perfSONAR data model includes a representation of the network topology [6] and this is stored in the Topology Service (TS).

A. *UNIS*

The *UNIS* component of Periscope is the Unified Network Information Service [12]. *UNIS* is a combination of the LS and TS functionality in perfSONAR and stores service information and configuration for discovery, measurement metadata that describes available measurement data and network topology. The perfSONAR topology information was initially used to describe the measured network entities, but was extended to support path finding for dynamic network systems like ESnet OSCARS [11]. In *UNIS*, everything is related to a common network topology graph, from services to the paths that flows take through the network.

1) UNIS Abstraction: As described in [12], [36], [6], a key aspect of *UNIS* is the use of the same basic abstract elements for network resources including `Node`, `Port`, `Link`, `Path`, and `Service`. These elements are "subclassed" with layer or domain specific attributes. These entities at different layers of the network stack are described along with their relationships both *horizontally* — L2 `Port` connected to L2 `Port` via L2 `Link`, and *vertically* — L3 `Port` atop L2 `Port`, forming a multigraph [27]. *UNIS* can then describe the different network elements traversed as data flows through a network with all potential points of conflict, and of observation, represented. This path through the topology graph can be referred to directly via another *UNIS* element, `Path`, abstracting the details when appropriate.

Via various encoders, *UNIS* is able to take other standard network configurations and parameters as inputs, e.g., GENI resource specification (RSpec) files and router configuration files from Internet2 Network.

In other cases, little or no network configuration is known. In these cases we can derive network topology from `traceroute`, using expiring time-to-live counters to enumerate IP hops, and from this infer the L3 `Node`, `Port`, `Link` and `Path` elements.

2) Representation of Multi-layer Network Resource Conflicts: In the vast majority of deployments, networks are shared resources, and shared resources are subject to competing use. In the case of measurements, concurrent use over time may cause conflicts, and several technologies manipulating the state of the network configuration across layers may easily give rise to previously unforeseen issues.

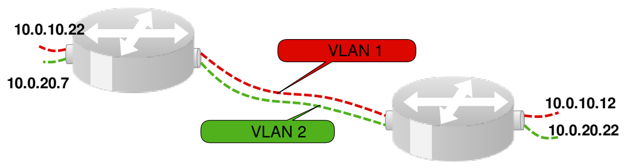

Fig. 1: Layer 2 Conflicts

With virtualization at many levels, this problem is even more significant. On edge hosts and routers alike, multiple IP layer `Port` may utilize the same Ethernet layer `Port`. Multiple VLAN `Port` endpoints may use the same physical `Port` and a virtual machine Ethernet may be bridged onto a shared physical `Port`. Virtual `Link` elements may share bandwidth in a single physical `Link` as well.

Taking the VLAN configuration in Figure 1 as an example, either via the results of `traceroute` or querying it from perfSONAR, we observe the IP hops 10.0.10.22 and 10.0.20.7. They show no conflicts at the IP layer; however, a common link lies underneath at Layer 2, and in fact the path is realized by a dedicated VLAN. If the active probes running over these resources attempt to measure the bandwidth, they should be scheduled independently.

The *UNIS* approach to deal with such situation is to represent these layered relationships insofar as they are known. *UNIS* is meant to be a "library" or a "record keeper". It should unveil multi-layer conflicts via its structured data. In other words, relations of network elements are implied by *UNIS* internal data entries. To discover the aforementioned conflicts, the computation tasks are assigned to the HELM subsystem. *UNIS* is a neutral platform that keeps network abstractions separated and additional functionalities, like HELM can be plugged in to minimize data redundancy.

B. *BLiPP*

The Periscope MP component is called *BLiPP* (Basic Lightweight Periscope Probe.) As suggested by the name, *BLiPP* acts as a collector agent, running arbitrary probes and storing their results in the measurement store. *BLiPP* also assumes its role as a scheduled agent, running local commands and formatting their results based on the associated measurement schema and regular expressions. *BLiPP* is configured with directives from the Periscope Lookup/Topology Service, *UNIS*. This configuration provides a measurement metadata description, and *BLiPP* in turn produces sets of measurement data and metadata.

HELM, described in more detail in Section IV acts as a coordinator. It controls monitoring tasks by coordinating *BLiPP* probes with a schedule generated based on an aggregate measurement configuration from the user. *BLiPP* and HELM cooperate and exchange data via *UNIS*. They both solely extend the Periscope framework, yet taken together, they form a feedback loop carrying out the active probe scheduling and executing task.

IV. HELM: Scheduling Active Network Measurements

This chapter discusses the design and implementation of HELM. In our framework, HELM takes an "aggregate" measurement specification from a Periscope user. These specifications include a set of hosts and a set of measurements, along with characteristics of each measurement, including parameters and periodicity. The goal of HELM is to construct a conflict-free schedule for these measurements and configure the *BLiPP* instances appropriately. HELM has access to *UNIS*, in which other measurement activities' configuration is stored. Note this scenario also allows HELM to elide duplicate measurements and reuse exiting measurement activities.

Many of the active network measurements we employ in Periscope and perfSONAR are parameterized by a duration. For purposes of this work, we assume that we know the measurement duration for each schedulable entity. Due to the nature of scheduling and the sensitivity of some types of measurements, timelines on probe hosts and on HELM need to be synchronized. HELM (and *BLiPP*) can either work in pulling mode or subscribe a listener at certain *UNIS* element(s). It uses RESTful APIs to pull and its publish-subscribe mechanism is implemented via WebSocket channels. We enable the Network Time Protocol (NTP) [23] on all participants. Normally, NTP is supposed to keep the hosts within a few milliseconds away from each other, and we consider it sufficient for current scheduling purposes. When subscribed to *UNIS*, HELM and *BLiPP* will always get notified instantaneously on event updates. This can help eliminate possible missing or redundant data transfers.

Elements in a network form a graph, and in our model this is a multigraph representing entities at different layers. We refer to this graph as the resource topology in order to distinguish from the graph we construct for scheduling. This resource topology represents the network in our *UNIS* model, with which we can programmatically identify paths between sources and sinks of network flows, discover related resources residing at different network layers and contract or expand representations to leverage the best position in the network hierarchy.

As described in Section III-A, this resource representation is a cross-layer topology. This implies that adjacent nodes in this topology may belong to different network layers. The connection between nodes reflects a relationship rather than a necessarily physical connection. Network monitoring systems have employed various strategies to approximate topology when the details are not known [10]. Our layered topology model can always be correct, even if it is less than complete. For instance, a TCP connection is simply modeled as a Layer 4 link regardless of the availability of transport-specific knowledge. We may be able to expand downward with information about lower layers, but we can always use high-level information to construct a overall resource topology.

Discovering the underlying topology is a key problem in general network measurement. Note however that in our two main areas of emphasis, R&E networks and experimental distributed and cloud computing testbeds, the details of the topology are often discoverable. In the case of Internet2, we get the published router configurations and use those to create a detailed *UNIS* model. ESnet publishes its topology in a perfSONAR Topology Service, which is an ancestor of the *UNIS* model and which we can import directly. In GENI, the resource specification descriptions (RSpecs) are advertised; we encode the underlying resource description and the dynamic "slice" of resources created by a user into the *UNIS* model.

After describing the cross-layer topological nature of these networks, we determine what resources cause contentions. For measurement tasks that conflict with each other on common resources, they form an intersection graph. We map the scheduling problem onto a graph coloring algorithm to utilize well-studied solutions. The graph coloring problem, e.g. discovering the optimal number of colors such that no two adjacent vertices have the same color, is an NP-complete problem for general graphs. We not only want to populate the schedule, but also want to reduce the total amount of waiting time for each probe test by using the minimal number of colors.

In essence, HELM needs to construct a given topology, analyze the tasks assigned to this topology to identify paths based on the measurement tasks and construct the intersection graph out of these conflict paths.

A. Constructing Topologies and Paths

After we get the topology information from Internet2, ESnet and GENI and encode them into the *UNIS* model (or simply reference previously encoded information), we can construct the resource topology.

Getting the `Path` information is a bit more involved. HELM needs to know the network elements a particular measurement will use. In some cases, that `Path` will be explicitly configured by the user with SDN. In other cases, we need to learn the end-to-end paths from an already configured network. HELM can directly assign tests to *BLiPP* agents running at the end hosts, and request them to launch `traceroute` and discover the IP hops. After this data is collected and uploaded to *UNIS*, HELM will store it. Alternatively, HELM can query it from other information sources like perfSONAR or Periscope measurement archives with the client API. The latter case can be useful when access to the end hosts is not possible. Both mechanisms can provide the hop information.

Our end-to-end path starts from extending a pure Layer 3 path. HELM is able to map an IP hop to a port located at Layer 2. To achieve this, static routing configuration files may be uploaded and parsed into *UNIS*, forwarding table discovery services (a looking glass) for public R&E networks backbone routers may also be used. HELM can identify such mappings and resolve the Layer 3 address to a unique lower layer address in its cross-layer topology.

To fill the single or multiple missing links between two newly mapped ports, we make use of our knowledge about this Layer 2 network segment. HELM will calculate the path based on a standard algorithm, Dijkstra's Single-Source Shortest Path in the link layer to determine a Layer 2 path through that segment.

Another example is to calculate VM related resources, especially in our GENI experiments. In the case of GENI, users allocate compute resources in the form of virtual machines on demand as Figure 2. Network measurements on such resources need to be aware of the underlying hardware layer in order to schedule conflict-free measurements. As shown in Figure 3, contentions can arise if the measurement service is not aware of the underlying host resource distribution. The GENI manifest RSpec, which describes the hardware details of the allocated resources, e.g. raw host IDs, NIC's UUIDs is updated to *UNIS*, and helps in detecting hardware conflicts during measurement scheduling.

The eventual working topology is the result of many inputs: `traceroute`, topology archives, configuration data and path calculations. By consulting it, HELM attempts to resolve any shared resource conflicts with the observable knowledge about a given network. A functional implementation of the HELM system is a cornerstone of this study. By tolerating incomplete knowledge of layers, which is common in heterogeneous networks, our solution maximizes the benefits while remaining

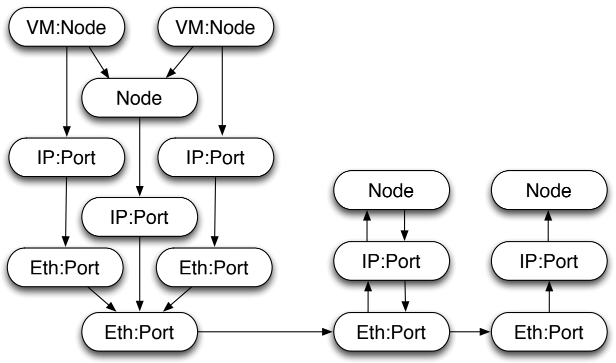

Fig. 2: Virtual Tenants

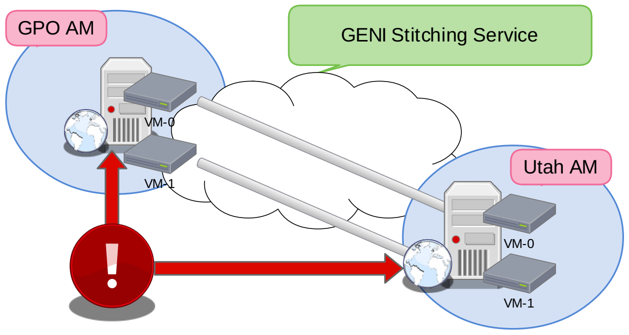

Fig. 3: GENI VM

practical over a wider set of deployment scenarios. We note that the constructive appraoch taken with HELM differentiates this work from a purely theoretical emulation.

B. *Constructing the Intersection Graph*

There are numerous approaches possible for constructing the intersection graph that allows HELM to compute a schedule of measurements free of shared resource contention. In this study, we reduce our scheduling task to a graph coloring problem. It is a canonical methodology for computing scheduling problems and we investigated classes of Perfect Graphs such as Chordal [13] and Trapezoid [9] Graphs to find an appropriate model that fit our particular scheduling problem. In the end, we concluded that our approach required a more general representation, due in part because any two measurements can conflict in complex ways. A Trapezoid Graph, for example, has constraints requiring its trapezoids to be placed between two parallel lines, which rules out many cases that require a general intersection graph [22]. This fact makes the elimination-order optimization [29] not possible. Instead, we chose a heuristic, namely smallest-last ordering [21], combined with the greedy coloring algorithm as our approach.

The maximum outdegree of a node in the graph is an upper bound on k for a k-coloring approach. We construct our graph for the coloring phase so that the edges reflect conflicts, and thus the number of conflicts (i.e., colors) correspond to the maximum number of non-contending measurement slots.

Our problem can be considered as:

$$G_{UNIS} = V, E \models Node, Port, Link \in V \tag{1}$$

HELM is given a measurement specification at a particular network layer (e.g., TCP throughput tests), which is reified into a set of endpoints, V_{meas}. From V_{meas} we construct a set of paths,

$$Path_{meas} = (P_{meas1} ... P_{measN}) \tag{2}$$

$$P_{meas1} = V_1 ... V_M \tag{3}$$

For each $P \in Paths$ we walk the path, copying the traversed vertices from G_{UNIS} into G_{meas}, including a reference count for each node in G_{meas}, which is incremented when referenced, and storing a reference to each Path P that traverses it.

We elide any nodes from G_{meas} that have only one reference. We create a distinct vertex in the intersection graph for each time a node in G_{meas} with a reference to the Path P that created it. The resultant G_{meas} is the intersection graph that represents the conflicts.

The graph coloring problem is closely related with the notion of a clique. For example, for all the Chordal Graphs, the largest clique determines the chromatic number. In general, a full mesh test running on a set of network resources will not necessarily form a Chordal Graph; however, we can make use of a heuristic approach to transform the graph into a form that has the desired property.

Note that, typical graph coloring problems take $G(E, V)$ as an input, which does not include the clique information about the graph unless additional computations are done. The complexity of discovering clique information from a graph is as computationally difficult as many other problems, e.g., the minimum coloring problem since the approach essentially groups vertices into smaller sets of unique cliques. Thus a "pre-processing" step is required that makes use of the smallest-last ordering heuristic to prevent the greedy coloring algorithm from using a poor ordering. A key advantage of HELM is that we are not given a graph directly, but rather we form our graph based on the known relationships between topology and measurement path objects. Therefore, we may compute and transform the graph as necessary during the contruction of the intersection graph.

C. *Scheduling Policy*

We note that the algorithm as presented realizes a strict conflict-free policy. That is certainly the desired situation for certain cases, but not all. A large router or switch would certainly be able to handle multiple measurement flows in many situations.

Our algorithm generalizes to deal with any of these cases. If administrative policy is that a resource can manage a certain number of concurrent measurement flows, we can effectively clone that resource in the topology, creating N clones provides N concurrent slots. Even if a device should take an arbitrary number of measurement flows, we can elide it from the consideration of intersection graph. In measurements that are sensitive to queuing, we might not constrain the node itself but only the egress interface.

V. Experiments

In this section, we list some experimental results from running HELM and discuss two major factors: 1) the correctness of HELM measurement schedules in real-world use cases, 2) the scalability of our scheduling algorithm. We perform an empirical analysis that demonstrates how HELM works in a realistic scenario.

As the basis of our experimental environment, we made use of both the Internet2 and ESnet backbone networks. These two networks cover a significant portion of the available R&E networks deployed within the US. Their topologies are stored in the *UNIS* topology service and converted into in-memory local objects for HELM to manipulate. We randomly select 300 pair-wise measurement points in this mixed topology, and configure them as *iperf* network bandwidth tests. If deployed, each *iperf* measurement should be scheduled at a different time slot than any other measurement that traverse common resources. We ran HELM on a low end commodity device, with 2.6GHz Intel dual core CPUs and 4GB of RAM to demonstrate that HELM can operate with modest computational resources.

Note that both the communication to a remote *UNIS* service and local calculation contribute to the total processing time. The network communication to *UNIS* services is a typical RESTful service invocation, and the communication time is highly dependent the latency to the *UNIS* server. An evaluation of *UNIS* query response time has been presented in our previous work [12] and does not factor into our current evaluation of HELM. Therefore, we only present the actually scheduled calculation time cost in the charts. The HELM implementation makes use of the graph coloring algorithms in the Boost Graph Library [3].

[3]http://www.boost.org/doc/libs/1_49_0/libs/graph/doc/index.html

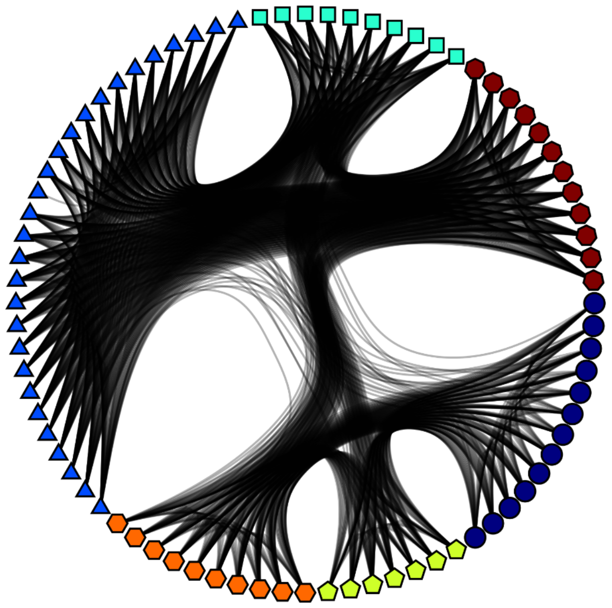

Fig. 4: Internet2 and ESnet end hosts measurements

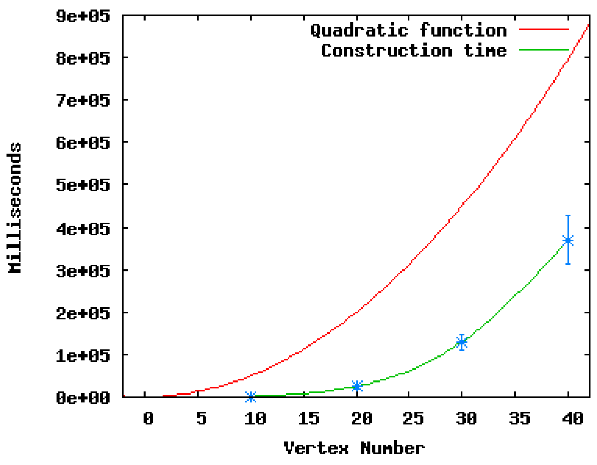

Fig. 5: construction time compare to O(n^2)

Figure 4 shows the actual cross-layer connectivity HELM discovered about all the requested end hosts. Their paths are discovered by proactively running `traceroute` between sources and destinations. By querying the forwarding information stored in HELM, IP addresses are translated into topology resource identifiers. All of the resolved resources form an intersection graph where each edge denotes a resource conflict at each network layer. This solution is computed over less than 300ms in repeated experiments in the aforementioned testbed setup. As described below, a relatively large number of available network paths reduced the routing conflicts, and fewer overall conflicts were identified. For example, in Internet2, end hosts located in New York city area may take a route through Chicago to LA, whereas the opposite route may choose a southern path through the Atlanta area. This fact dramatically reduces the density of the generated graph and thus reduces the amount of necessary computation. The routing diversity in the use case allows concurrent traffics, and HELM shows that it can produce the schedule for this real world scenario.

Next, we evaluate how HELM scales with increasing measurement demands, where we expect the computation time to follow the complexity of the smallest-last ordering heuristic and the greedy coloring algorithm.

We vary our end host number from 10 up to 40 composing of up to 40×39 full mesh measurement pairs. Further, we assume the worst case, in which each single measurement pair will conflict with other sibling measurements. This generates on the order of n^2 edges in the resulting intersection graph. In the following figures, the *x* axes show the size of the problem set, and the *y* axes show the processing time. We averaged the execution time over 10 runs for each case. A smooth line connecting the data points is used as the expected processing time curve. We measured three time sections, namely the time used to construct an intersection graph for the problem set, the time used to re -order the vertices as a heuristic input to coloring algorithm and the actual coloring time. Note that, the construction and ordering step do not need to be re-executed multiple times. Once the topology is inputed and initial measurement requests are submitted to HELM, HELM should build its intersection graph and order them in the graph according to their degree. From then on, new requests can be inserted additively.

The construction phase is O(n^2), n denotes the number of the total amount of measurements. Figure 5 plots a line to show the expected quadratic curve with a constant factor.

Similarly, in figure 6 the quadratic curve is the upper bound of the increasing of ordering time consumption. It is consistent with the complexity claimed [21]. Note that, we have set up the scenarios such that all measurement events conflict with each other, so the number of edges are on the same order as the square of the number of vertices. O($|V|+|E|$) is effectively O(n^2), where $n = V$.

Figure 7 shows the coloring time complexity with the ordered vertex input. Again, the computation time did not exceed the expected upper bound.

As experiments show, the complexity is as predicted. In terms of scalability, the discussion can be subjective, depending on how much computational power one may have, how diverse (for a given network size) the net-

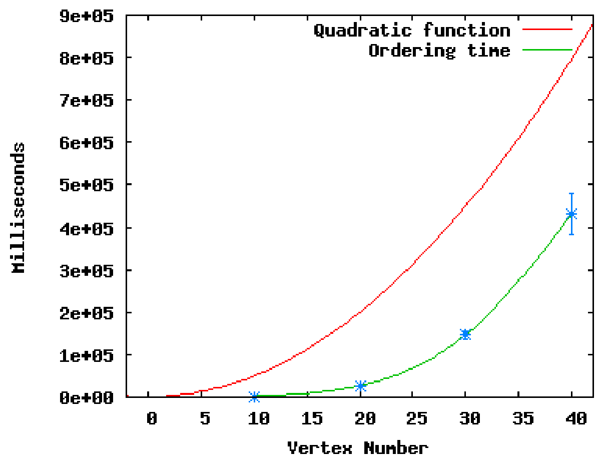

Fig. 6: *ordering time compare to O(n^2)*

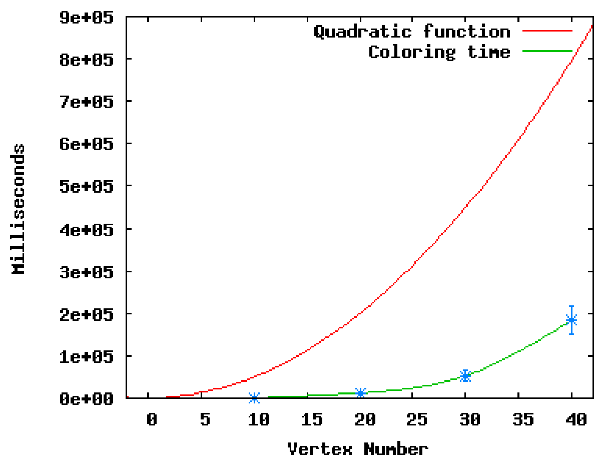

Fig. 7: *coloring time compare to O(n^2)*

work paths are etc. Our experiment is conducted on a modestly equipped Linux host, with representative real-world networks as input, and yielded results suitable for a centralized deployment scenario.

VI. Related Work and Conclusions

We refer to and contrast with several previous active measurement scheduling studies. A number of approaches have been proposed for mitigating scheduling computation overhead. Some try to balance the intrusive nature of active probing [20] while others, including this study, look for a way to find non-conflicting schedules. For example, in Daniel et al [17], active probe timing is focused on traffic peak hour avoidance. In contrast to our work, it is a macro scheduling approach, as a specific probe conflict is not under consideration. Instead, the strategy focuses on deriving an optimal schedule for long period runs. Some approaches lean toward reserving resources rather than schedule them [33]. Not only may the methodologies vary, but scheduling may also target specific shared resource types. For instance, [15] leverages the property of optical network environments. Even on a similar test environment such as ESnet, scheduling may show different perspectives, like in [16], instead of maximizing a single test probe, their study focuses on adopting multiple circuits at the same time as long as the infrastructure has adequate capacity. For HELM, we are interested in a solution that uses a generalized model to accommodate the varieties of networks, in our case supported by *UNIS*.

The second half of our solution is related to the large body of work in graph algorithms. Indeed, graph algorithms have been applied in a lot of research about resource scheduling [28] [1], as they share common combinatorial principles. Related with Qin's study, Network Weather Service [34] also scheduled with a token passed around a user-defined clique, where shared network resources form cliques in their equivalent intersection graphs. A heavily shared single point resource causes a large clique in the graph. While connecting with other resource nodes, the group of shared points effectively generate smaller cliques nested within the large "parent" clique.

Since the family of general scheduling optimization problems is NP-complete [18], some heuristic methods have been used [8]. In these studies, heuristics can help guide solution finding. However, a concern about these methods is that they may introduce dependences on external information sources, e.g. Earliest Deadline First (EDF) heuristics are applied to deadline constrained problems. Periodical tasks have deadlines commonly, but not for all cases. On-demand tasks are handled by a separated scheduling schema in Calyam's work, etc. In contrast, our Smallest Last Ordering heuristic [21] is for pure graph algorithms as all it needs is vertices and their connectivities. Its graph abstraction makes it fit broader use cases. Also, it gives a definitive time complexity.

To conclude, HELM is an integrated tool that collects and analyzes network topology information and transforms it into graph representations and calculates solutions for schedule optimization via graph heuristics and algorithms. A key question we could further investigate as future work is the possibility to better use the knowledge we have already collected from certain networks. The resource topology should contain more information than what we used to transform our problem. Such information might inspire the development of new heuristics leading to an improved algorithm.

References

[1] Amotz Bar-Noy, Mihir Bellare, Hadas Shachnai, Tami Tamir, and et al. On chromatic sums and distributed resource allocation, Feb 1998.

[2] P. Barford, N. Duffield, A. Ron, and J. Sommers. Network performance anomaly detection and localization. In *INFOCOM 2009, IEEE*, pages 1377–1385, April 2009.

[3] Mark Berman, Jeffrey S. Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61(0):5 – 23, 2014. Special issue on Future Internet Testbeds Part I.

[4] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. *Computers, IEEE Transactions on*, C-35(5):389–393, May 1986.

[5] K.A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R.A. Olsson. Detecting disruptive routers: a distributed network monitoring approach. *Network, IEEE*, 12(5):50–60, Sep 1998.

[6] A Brown, M Swany, and J Zurawski. A general encoding framework for representing network measurement and topology data. *Concurrency and Computation: Practice and Experience*, 21(8):1069–1086, June 2009.

[7] Mathijs Den Burger, Thilo Kielmann, and Henri E. Bal. Topomon: A monitoring tool for grid network topology. In *In International Conference on Computational Science (2*, pages 558–567. Springer, 2002.

[8] P. Calyam, Chang-Gun Lee, Phani Kumar Arava, and D. Krymskiy. Enhanced edf scheduling algorithms for orchestrating network-wide active measurements. In *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 10 pp.–132, Dec 2005.

[9] F. Cheah and D.G. Corneil. On the structure of trapezoid graphs. *Discrete Applied Mathematics*, 66(2):109 – 133, 1996.

[10] Yan Chen, David Bindel, Hanhee Song, and Randy H. Katz. An algebraic approach to practical and scalable overlay network monitoring. *SIGCOMM Comput. Commun. Rev.*, 34(4):55–66, August 2004.

[11] ESnet On-demand Secure Circuits. Advance reservation system (oscars), 2007.

[12] A. El-Hassany, E. Kissel, D. Gunter, and M. Swany. Design and implementation of a unified network information service. In *10th IEEE International Conference on Services Computing (SCC 2013)*, June 2013.

[13] Fnic Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974.

[14] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, and J. Zurawski. PerfSONAR: A service oriented architecture for multi-domain network monitoring. In *In Proceedings of ICSOC 2005*, December 2005.

[15] Yaohui Jin, Yan Wang, Wei Guo, Weiqiang Sun, and Weisheng Hu. Joint scheduling of computation and network resource in optical grid. In *Information, Communications Signal Processing, 2007 6th International Conference on*, pages 1–5, Dec 2007.

[16] Dimitrios Katramatos, Xin Liu, Kunal Shroff, Dantong Yu, Shawn McKee, and Thomas Robertazzi. TeraPaths: End-to-End Network Resource Scheduling in High-Impact Network Domains. *International Journal On Advances in Internet Technology*, 3(1 and 2):104–117, sep 2010.

[17] N. Daniel Kumar, Fabian Monrose, and Michael K. Reiter. Towards optimized probe scheduling for active measurement studies. pages 26–31, March 2011.

[18] Frank Thomson Leighton. A graph coloring algorithm for large scheduling problems. *Journal of research of the national bureau of standards*, 84(6):489–506, 1979.

[19] J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. *Complexity of Machine Scheduling Problems*, volume 1 of *Annals of Discrete Mathematics*, pages 343–362. Elsevier, 1977.

[20] Bruce B. Lowekamp. Combining active and passive network measurements to build scalable monitoring systems on the grid. *SIGMETRICS Perform. Eval. Rev.*, 30(4):19–26, March 2003.

[21] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *J. ACM*, 30(3):417–427, July 1983.

[22] Terry A McKee and Fred R McMorris. *Topics in intersection graph theory*, volume 2. Siam, 1999.

[23] D. Mills. Network time protocol (version 3) specification, implementation, 1992.

[24] Toshiyuki Miyachi, Ken-ichi Chinen, and Yoichi Shinoda. Starbed and springos: Large-scale general purpose network testbed and supporting software. In *Proceedings of the 1st International Conference on Performance Evaluation Methodolgies and Tools*, valuetools '06, New York, NY, USA, 2006. ACM.

[25] Jeffrey C. Mogul. Efficient use of workstations for passive monitoring of local area networks. In *In Proceedings of ACM SIGCOMM 90*, pages 253–263, 1990.

[26] Jens Palsberg. Register allocation via coloring of chordal graphs. In *Proceedings of the Thirteenth Australasian Symposium on Theory of Computing - Volume 65*, CATS '07, pages 3–3, Darlinghurst, Australia, Australia, 2007. Australian Computer Society, Inc.

[27] Sriram Pemmaraju and Steven Skiena. *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, page 89. Addison-Wesley, 1990.

[28] Zhen Qin, Roberto Rojas-Cessa, and Nirwan Ansari. Task-execution scheduling schemes for network measurement and monitoring. *Computer Communications*, 33(2):124–135, February 2010.

[29] Donald J Rose, R Endre Tarjan, and George S Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

[30] S Shanlunov, B Teitelbaum, A Karp, JW Boote, and M Zekauskas. A one-way delay measurement protocol (owamp). Technical report, Internet Draft, May, 2003.

[31] C. Shapiro and H.R. Varian. *Information rules: a strategic guide to the network economy*. Strategy/Technology / Harvard Business School Press. Harvard Business School Publishing, 1999.

[32] Herbert A. Simon. The sciences of the artificial, September 1996.

[33] B. Urgaonkar and P. Shenoy. Sharc: managing cpu and network bandwidth in shared clusters. *Parallel and Distributed Systems, IEEE Transactions on*, 15(1):2–17, Jan 2004.

[34] Rich Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.

[35] Rich Wolski, Neil T. Spring, and Jim Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Journal of Future Generation Computing Systems*, 15:757–768, 1999.

[36] J. Zurawski, M. Swany, and D. Gunter. A scalable framework for representation and exchange of network measurements. In *TRIDENTCOM*, 2006.