G2: A Network Optimization Framework for High-Precision Analysis of Bottleneck and Flow Performance

Jordi Ros-Giralt[1], Sruthi Yellamraju[1], Atul Bohara[1], Richard Lethin[1]
Josie Li[2], Ying Lin[2], Yuanlong Tan[2], Malathi Veeraraghavan[2]
Yuang Jiang[3], Leandros Tassiulas[3]
[1] *Reservoir Labs, USA*
[2] *University of Virginia, USA*
[3] *Yale Institute of Network Science, USA*
{giralt, yellamraju, bohara, lethin}@reservoir.com
{jl9gf, yl5cp, yt4xb, mv5g}@virginia.edu
{yuang.jiang, leandros.tassiulas}@yale.edu

***Abstract*—[1] Congestion control algorithms for data networks have been the subject of intense research for the last three decades. While most of the work has focused around the characterization of a flow's bottleneck link, understanding the interactions amongst links and the ripple effects that perturbations in a link can cause on the rest of the network has remained much less understood. The Theory of Bottleneck Ordering is a recently developed mathematical framework that reveals the bottleneck structure of a network and provides a model to understand such effects. In this paper we present G2, the first operational network optimization framework that utilizes this new theoretical framework to characterize with high-precision the performance of bottlenecks and flows. G2 generates an interactive graph structure that describes how perturbations in links and flows propagate, providing operators new optimization insights and traffic engineering recommendations to help improve network performance. We provide a description of the G2 implementation and a set of experiments using production TCP/IP code to demonstrate its operational efficacy.**

***Index Terms*—Traffic engineering, congestion control, bottleneck structure, max-min**

I. Introduction

Modern computer networks are constructed using a variety of components with the system-level objective to achieve maximum overall flow-performance at the minimum operational cost. For example, the congestion control algorithm part of the TCP protocol tries to contribute to the system-level goal by maximizing network utilization while achieving some form of fairness among competing flows. In this context, it is well-known that the performance of a flow is uniquely determined by the capacity of its bottleneck link and its end-to-end round trip time (RTT) [1] [2] [3]. As observed in [4], it is however challenging to identify the complex interactions among the bottleneck links. For instance, if a link's effective capacity is altered—e.g., either logically by reallocating circuits, physically by performing link upgrades, or simply through the action of new flows passing through the link or the termination of existing ones—how does such perturbation propagate through the network and what effective impact it has on the rest of the links? Because such *hidden* link interactions can significantly impact the overall performance of a network, existing systems that lack a precise understanding of their cause and effects fall short in helping operators optimize network performance.

The Theory of Bottleneck Ordering, as recently proposed by the authors in [5], has shown that bottlenecks in a distributed network interact with other bottlenecks by following a specific digraph structure called the *bottleneck structure*. By revealing the collective behavior of the bottlenecks, the theory provides insights into the inherent topological properties of a network. In particular, the bottleneck structure reveals the bounded regions in the network that a perturbation on a link's effective capacity can influence. Such information enables a new methodology to analyze bottleneck and flow performance that takes into account the global topological structure of the network. For instance, this mathematical framework allows to identify flows that, while relatively low in throughput, they have a high-negative impact on the performance of the network as they traverse hotspot links that have a large region of influence.

In this paper, we introduce the *G2 optimization framework*, the first network operational system that uses the Theory of Bottleneck Ordering to learn the complex bottleneck structure of networks and enable operators to optimize their performance. G2—which stands for *Gradient Graph*—generates the collective bottleneck structure of a network and presents it in an interactive graphical user interface. Among other features, G2 allows network operators to:

- Analyze the subtle impacts of different congestion control algorithms (e.g., TCP BBR [2] or TCP Cubic [6]) onto the overall network performance.
- Recommend traffic engineering and capacity planning actions by identifying and quantifying the regions in the network that are influenced by the performance of bottlenecks and flows.
- Generate an optimal baseline (currently based on the max-min optimality condition [7], albeit not limited to it) to help optimize dynamic traffic policing on non-compliant flows.

[1]Work partially funded by the Department of Energy under contract DE-SC0019523. Some of the algorithms are patent pending.

978-1-7281-6666-7/19/$31.00 ©2019 IEEE
DOI 10.1109/INDIS49552.2019.00011

- Analyze the impact of the bottleneck structure on a network's convergence to its steady-state, providing insights onto the stability of the underlying congestion control algorithm.

In our experiments, we deploy the G2 framework to work with two types of networks: (1) Networks emulated using Mininet [8] and (2) real-world Research and Education Networks—ESnet Testbed [9], and SCinet [10]. Using the integration with Mininet networks, we run experiments at a scale up to 1000 concurrent traffic flows to show that G2 can generate optimization decisions for a variety of network configurations. Since the implementation of these network configurations and extensions to Mininet are useful for scientific research, we have made the G2-Mininet codebase available [11] to the research community under the BSD license.

Our contributions can be summarized as follows:

1) We propose and implement the G2 network optimization framework that uses the Theory of Bottleneck Ordering to understand the complex interactions of bottlenecks and enables operators to improve the efficiency of networks.
2) We provide an easy-to-use API for a variety of network technologies to allow rapid integration of G2 with real-world production networks.
3) Using a variety of Mininet-based network architectures, and a scale of up to 1000 flows, we evaluate the efficacy of G2 in providing optimization decisions, including (1) identification of elephant flows, (2) identification of optimal link upgrades, and (3) using the framework as a baseline to optimize the policing of non-compliant flows.
4) We perform the integration of G2 with two real network deployments, namely the ESnet Testbed [9] and SCinet [10], and show that G2 can feasibly work with real-world, large-scale networks.
5) We develop a user-interface that allows operators to use the G2 framework to optimize the underlying network.

II. Brief Summary of the Theory of Bottleneck Ordering

The Theory of Bottleneck Ordering is a new mathematical framework recently introduced in [5] that reveals the bottleneck structure of data networks. This structure takes the form of a digraph and reveals (1) the *hidden* relations that exist between bottleneck links and (2) the bounded regions in the network that bottlenecks can influence. For instance, given the bottleneck structure of a network, one can infer whether the performance of a bottleneck will affect another bottleneck (and to which degree) or how a variation in the effective capacity of a bottleneck will affect other bottlenecks.

In this section, we only provide a summary of the Theory of Bottleneck Ordering necessary to understand the network operational concepts presented in this paper. For a full description of the mathematical framework, refer to [5].

A. Simple Example of Bottleneck Structure

Consider the network configuration illustrated in Fig. 1-a, where links are represented by circles and flows by lines traversing the links. Each link l_i has a capacity c_i bps while each flow f_i transmits data at a rate r_i bps. Since the capacity of link l_1 is lower than the capacity of link l_2, $c_1 < c_2$, flows f_1 and f_2 must be bottlenecked at link l_1. Assuming without loss of generality a fair allocation between these two flows, their transmission rate must be split equally: $f_1 = f_2 = c_1/2 = 1/2$ bps. Now it is easy to see that the transmission rate of flow f_3 is equal to the remaining capacity of link l_2 after flow f_2's rate is subtracted from it: $f_3 = c_2 - f_2 = 3/2$. It is also common [7] to define the *fair share* value of a link l_i, denoted as s_i, as the transmission rate of any of the flows that is bottleneck at link l_i. Thus, in this example we have $s_1 = 1/2$ and $s_2 = 3/2$.

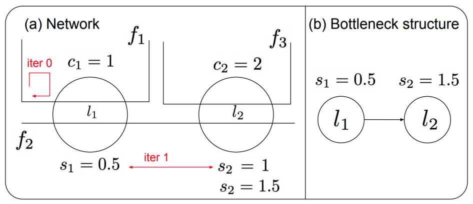


Fig. 1: A 2-level bottleneck structure with direct precedence.

While this network configuration is very simple, it helps to introduce some subtle but relevant relationships that exist amongst links. For instance, it is easy to see that the derivative of s_2 with respect to c_1 is 0.5, $\partial s_2/\partial c_1 = 0.5$, while the derivative of s_1 with respect to c_2 is 0, $\partial s_1/\partial c_2 = 0$. That means the performance of link l_2 depends on the performance of link l_1, but not vice-versa, thus revealing a notion of hierarchy or ordering between these two bottleneck links. This leads to the bottleneck structure illustrated in Fig. 1-b, a digraph where each vertex represents a bottleneck link and each directed edge from a link l to a link l' implies that the performance of link l' depends on link l, but not vice-versa.

The Theory of Bottleneck Ordering mathematically characterizes these subtle relationships amongst links, revealing for any arbitrary network configuration its bottleneck structure.

B. The Bottleneck Precedence Graph (BPG) Algorithm

A key component of the proposed network optimization framework is the construction of the bottleneck structure itself. For this objective, we will be using the Bottleneck Precedence Graph (BPG) algorithm introduced in [5]. In this section, we provide a high-level description of this algorithm and an example to help interpret it.

Algorithm 1 presents the BPG algorithm and Table I provides a summary of the parameters and variables that it uses. The BPG algorithm is based on the water-filling algorithm introduced by Ros-Giralt and Tsai in [12], which provides a generalization of the original algorithm by Bertsekas and Gallager [7] by allowing bottleneck links to converge concurrently. In addition to revealing the order in which links and flows converge to their steady-state regime in distributed congestion control algorithms, this parallelization property provides the key mechanism to construct the bottleneck structure of the network, as demonstrated in [5].

TABLE I: *Notations used in the BPG algorithm [5].*

Variable	Definition
$\mathcal{L}$	Set of links in the input network
$\mathcal{F}$	Set of flows in the input network
$\mathcal{F}_l$	Set of flows going through link l
c_l	Capacity of link l
s_l^k	Fair share of link l at iteration k
u_l^k	Upstream fair share of link l at iteration k
$\mathcal{L}^k$	Set of unresolved links at iteration k
$\mathcal{C}^k$	Set of converged flows at iteration k
$\mathcal{D}_l^k$	Set of direct precedents of link l at iteration k
$\mathcal{I}_l^k$	Set of indirect precedents of link l at iteration k
$\mathcal{R}_l^k$	Set of relays of link l at iteration k
$\mathcal{B}$	Set of bottleneck links
r_f	Rate of flow f
$\setminus$	Set minus operator

Algorithm 1 BPG

```
1: $\mathcal{L}^0 = \mathcal{L}; \mathcal{C}^0 = \{\emptyset\}$;
2: $\mathcal{D}_l^0 = \mathcal{I}_l^0 = \mathcal{R}_l^0 = \{\emptyset\}, \forall l \in \mathcal{L}$;
3: $k = 0$;
4: while $\mathcal{C}^k \neq \mathcal{F}$ do
5:     $s_l^k = (c_l - \sum_{\forall f \in \mathcal{C}^k \cap \mathcal{F}_l} r_f)/|\mathcal{F}_l \setminus \mathcal{C}^k|, \forall l \in \mathcal{L}^k$;
6:     $u_l^k = min\{s_{l'}^k \mid \mathcal{F}_{l'} \cap \mathcal{F}_l \neq \{\emptyset\}, \forall l' \in \mathcal{L}^k\}, \forall l \in \mathcal{L}^k$;
7:     for $l \in \mathcal{L}^k, s_l^k = u_l^k$ do
8:         $r_f = s_l^k, \forall f \in \mathcal{F}_l$;
9:         $\mathcal{L}^k = \mathcal{L}^k \setminus \{l\}$;
10:        $\mathcal{C}^k = \mathcal{C}^k \cup \{f, \forall f \in \mathcal{F}_l\}$;
11:        for $l' \in \mathcal{L}^k, \mathcal{F}_{l'} \cap \mathcal{F}_l \neq \{\emptyset\}$ do
12:            $\mathcal{D}_{l'}^k = \mathcal{D}_{l'}^k \cup l$;
13:        end for
14:        for $l', l_r \in \mathcal{L}^k, \mathcal{F}_{l'} \cap \mathcal{F}_{l_r} \neq \{\emptyset\}, s_{l_r}^k < s_{l'}^k$ do
15:            $\mathcal{R}_{l'}^k = \mathcal{R}_{l'}^k \cup \{l_r\}$;
16:        end for
17:        for $l' \in \mathcal{D}_{l_r}^k \setminus \mathcal{D}_l^k, l_r \in \mathcal{R}_l^k \setminus \mathcal{D}_l^k$ do
18:            $\mathcal{I}_l^k = \mathcal{I}_l^k \cup \{l'\}$;
19:        end for
20:    end for
21:    $\mathcal{L}^{k+1} = \mathcal{L}^k; \mathcal{C}^{k+1} = \mathcal{C}^k$;
22:    $\mathcal{D}_l^{k+1} = \mathcal{D}_l^k; \mathcal{I}_l^{k+1} = \mathcal{I}_l^k; \mathcal{R}_l^{k+1} = \mathcal{R}_l^k$;
23:    $k = k + 1$;
24: end while
25: $\mathcal{B} = \mathcal{L} \setminus \mathcal{L}^k$;
26: $\mathcal{P} = \{\mathcal{D}_l^k, \forall l \in \mathcal{B}\} \cup \{\mathcal{I}_l^k, \forall l \in \mathcal{B}\}$;
27: return $\langle \mathcal{B}, \mathcal{P} \rangle$;
```

The outcome of the algorithm is the *BPG graph* represented with the tuple $\langle \mathcal{B}, \mathcal{P} \rangle$, where:

- $\mathcal{B}$ is the set of vertices, corresponding to the set of links that are a bottleneck to at least one flow;
- $\mathcal{P}$ is the set of edges, where an edge from a vertex link l to another vertex link l' exists if and only if the performance of link l' can be affected through alterations in the performance of link l. (These relationships are formally named *direct and indirect precedences*, see [5].)

As mathematically proven in [5], the BPG algorithm returns the bottleneck structure of the input network in polynomial time, which makes it practical in production environments. We illustrate how the algorithm works and how to interpret the resulting graph using an example:

Example 1. *Computing and interpreting the BPG graph.* To illustrate the process of constructing the BPG graph, we use the network shown in Fig. 2. This topology corresponds to the SDN WAN network called B4 that connects twelve of Google's large scale data centers globally using nineteen links as described in [13]. While we could have chosen any arbitrary topology, we focus on Google's B4 network to base the analysis on a real network. In this simple example we assume the presence of five flows $\{f_1, ..., f_5\}$ as shown in Fig. 2. This configuration leads to the following initial state of the BPG algorithm: $\mathcal{L} = \{l_1, ..., l_{19}\}$, $\mathcal{F} = \{f_1, ..., f_5\}$; $\mathcal{F}_1 = \{f_1, f_2\}$, $\mathcal{F}_4 = \{f_1, f_4, f_5\}$, $\mathcal{F}_5 = \{f_3\}$, $\mathcal{F}_6 = \{f_3, f_4\}$, $\mathcal{F}_{10} = \{f_5\}$, $\mathcal{F}_{12} = \{f_5\}$, $\mathcal{F}_{13} = \{f_4\}$, $\mathcal{F}_{15} = \{f_5\}$ and $\mathcal{F}_{17} = \{f_3\}$. We also assume that the capacity of the links are $c_1 = 80$, $c_4 = 110$, $c_5 = 200$, $c_6 = 130$, $c_{10} = 200$, $c_{12} = 200$, $c_{13} = 200$, $c_{15} = 20$ and $c_{17} = 200$ Gbps.

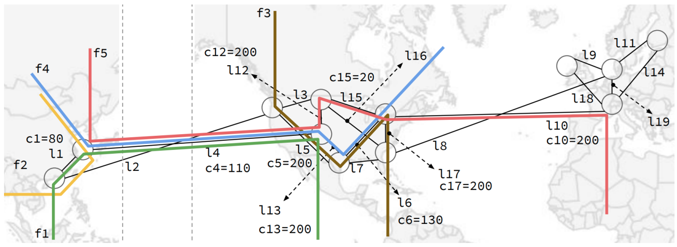


Fig. 2: Example of the B4 network with five flows used in Example 1.

The step by step execution of the BPG algorithm is presented in Table II. Each cell in this table has two numbers: the first number corresponds to the *fair share* of the link at iteration k, s_l^k (computed at line 5 of the BPG algorithm), while the second number corresponds to the *upstream fair share* of the link at iteration k, u_l^k (computed at line 6 of the BPG algorithm). A link l converges at an iteration k (and thus is removed from the set of unresolved links $\mathcal{L}^k$ at line 9) if its fair share is equal to its upstream fair share, $s_l^k = u_l^k$ (line 7 of the BPG algorithm). This condition is marked in Table II using a gray cell. Thus, the resulting set of bottlenecks for this network is $\mathcal{B} = \{l_1, l_4, l_6, l_{15}\}$. This set defines the vertices in the BPG graph. The algorithm also returns the set of link precedences, $\mathcal{P} = \{\mathcal{D}_l^k, \forall l \in \mathcal{B}\} \cup \{\mathcal{I}_l^k, \forall l \in \mathcal{B}\}$, which define the edges in the graph. These edges are progressively added to the graph by reiterating lines 11 through 19. We refer the reader to [5] for a detailed description of how these edges are constructed.

The result of running the BPG algorithm on our network example is presented in Fig. 3. The diameter of the graph is four since that is the length of its longest path $l_{15}-l_1-l_4-l_6$. It also corresponds to the value of k at the end of the algorithm, as shown in Table II. Using the notation in [5], we refer to this value as the *number of levels* of the bottleneck structure.

To interpret the BPG graph, we discuss the concept of *regions of influence*, one of the key results from the Theory of Bottleneck Ordering [5]:

Lemma 1. *Bottleneck influence.* A bottleneck l can influence the performance of another bottleneck l', i.e., $\partial s_{l'}/\partial c_l \neq 0$, if and only if there exists a set of bottlenecks $\{l_1, l_2, ..., l_n\}$

TABLE II: *Fair share s_l^k and upstream fair share u_l^k values at each iteration of the BPG algorithm for the network in Fig. 2. A gray cell denotes the corresponding link has converged.*

k	l_1	l_4	l_5	l_6	l_{10}	l_{12}	l_{13}	l_{15}	l_{17}
1	40 36.6	36.6 20	200 65	65 36.6	200 20	200 20	200 36.6	20 20	200 65
2	40 40	45 40	200 65	65 45	— —	— —	200 45	— —	200 65
3	— —	50 50	200 65	65 50	— —	— —	200 50	— —	200 65
4	— —	— —	200 80	80 80	— —	— —	— —	— —	200 80

such that l_i is a direct precedent of l_{i+1}, for $1 \leq i \leq n-1$, $l_1 = l$ and $l_n = l'$.

Proof. See [5]. □

A direct corollary of this lemma is that a bottleneck l can affect the performance of another bottleneck l' if and only if there exists a directed path from l to l' in the BPG graph. This result means the region of influence any bottleneck link l has on a given network corresponds to the set of links that can be reached from l in the BPG graph. Let us see how this lemma applies to our network example:

- Link l_{15} is a root vertex in the BPG graph, thus, using Lemma 1, it can influence the performance of all other links. That is, a variation on the effective capacity of link l_{15} will affect the fair share of all other links. This implies that its region of influence is the whole network.
- Link l_6 is a leaf vertex, which implies it does not influence the performance of any other link. Thus, its region of influence is empty.

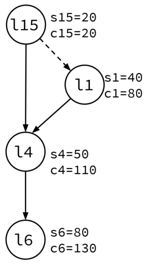


Fig. 3: Bottleneck structure of the network in Fig. 2.

C. Bottleneck and Flow Duality

While the Theory of Bottleneck Ordering helps understand the relations that exist amongst links, by itself the BPG graph resulting from this mathematical framework does not reveal any information regarding the performance of flows. However, as shown in [5], there exists a simple way to transform the BPG graph into a structure that reveals such information. This structure is referred to as the *flow gradient graph* (FGG):

Definition 1. *Flow gradient graph.* The *flow gradient graph* is a digraph such that:

- For every bottleneck link and for every flow, there exists a vertex.
- For every flow f: (1) If f is bottlenecked at link l, then there exists a directed edge from l to f; (2) If f is not bottlenecked at link l but it passes through it, then there exists a directed edge from f to l.

The flow gradient graph can be naturally constructed from the BPG algorithm as described in [5] in detail. Using the following example, we provide an intuitive description for interpreting this graph:

Example 2. *A simple flow gradient graph.* Consider the network in Fig. 4-a consisting of 4 links and 6 flows. Its flow gradient graph as described in Definition 1 is shown in Fig. 4-b, including next to each vertex the fair share values of each bottleneck link $\{s_1, ..., s_4\}$ and the expected transmission rate of each flow $\{r_1, ..., r_6\}$.

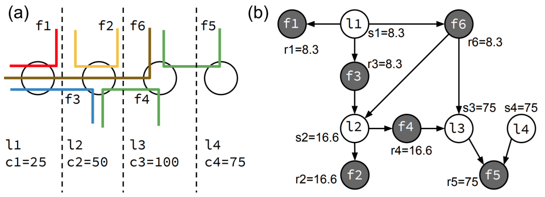


Fig. 4: A network and its flow gradient graph.

The FGG is a useful extension of the BPG since it allows to generalize the bottleneck analysis onto flows. One can see bottlenecks and flows as two sides of the same coin, corresponding to the supply of network resources and demand for them, respectively. As shown in [5], this duality principle implies that all the general properties derived from the Theory of Bottleneck Ordering have a dual correspondence in the domain of flows. For instance, Lemma 1 (*bottleneck influence*) can be translated to the domain of flows as follows:

Lemma 2. *Flow influence.* A flow f can influence the performance of another flow f', i.e., $\partial r_{f'} / \partial r_f \neq 0$, if and only if there exists a set of bottlenecks $\{l_1, l_2, ..., l_n\}$ such that (1) l_i is a direct precedent of l_{i+1}, for $1 \leq i \leq n-1$, (2) flow f' is bottlenecked at link l_n and (3) flow f goes through l_1.

Proof. See [5]. □

Using this Lemma, we can infer the following properties from the flow gradient graph in Fig. 4-b: (1) Flow f_5 has no influence on any of the other flows, since its bottleneck links (l_3 and l_4) have no influence on any other bottlenecks; (2) flows f_1, f_3 and f_6 have an influence on all other flows, since their bottleneck l_1 is a root vertex; (3) flow f_4 can only influence flows f_2 and f_5; and (4) flow f_2 can only influence flows f_4 and f_5. In Section V-B, we perform an experiment where we leverage the insights revealed by the FGG to identify flows that have a high impact on the performance of a network.

III. G2 Optimization Framework

The G2 optimization framework is a Python-based software designed to facilitate the integration of the BPG/FGG graphs and other G2 traffic engineering algorithms from the Theory of Bottleneck Ordering into various practical networking environments. Its plugin based architecture enables rapid integration of G2 with arbitrary real-world networks. This section introduces the components of the G2 framework, while Section IV delves into details of the G2 integration with various networks.

A. G2 Architecture

As reflected in Fig. 5, the high-level architecture of G2 consists of two layers: The *G2 layer* and the *Network Integration layer*. The G2 layer pictured in the middle is the core layer that consists of a set of network agnostic APIs and modules that use (1) traffic flow, (2) routing, and (3) topology information from the underlying network to calculate the BPG and FGG graphs and other traffic engineering computations. It consists of the following components:

- The *G2 Library* is the core component of the architecture. It implements the BPG algorithm shown in Algorithm 1 and other algorithms that are part of the mathematical framework—left outside the scope of this paper. Given a capacity dictionary (describing the capacity of each link) and a flow dictionary (describing the set of links traversed by each flow), the G2 library provides a JSON representation of the bottleneck structure of the network. Additionally, it performs key traffic engineering computations such as (1) obtaining the subset of high-impact flows, (2) making link upgrade recommendations, and (3) performing baseline analysis to identify non-compliant flows. These capabilities are demonstrated later in Sections V-B, V-C, and V-D, respectively.
- The *G2 Controller* acts as the 'glue' engine between the G2 library, the various configuration interfaces of the framework such as the CLI/UI, and the plugin based network integration environment. This module uses the registered plugin instances to retrieve the flow, routing, and topology information from the underlying network to calculate the BPG/FGG graphs and other G2 computations. It saves the network snapshot along with results of the computations to the database. It further provides the APIs needed to keep the G2 algorithms updated with any real-time changes to flows, routes, or topology to compute the most up-to-date bottleneck structure.
- The *Database* module provides an interface to store and retrieve the historical bottleneck and flow structure computed by G2. This module enables the dashboards to render real-time and retrospective bottleneck analysis of a network for traffic engineering purposes.
- *G2 Dashboards* form the web-based user interface (UI) to G2. They help operators visualize the network topology and the effects amongst flows and bottlenecks using the BPG and the FGG interactive graphs. We present the design of the dashboards in more detail in Section III-B.

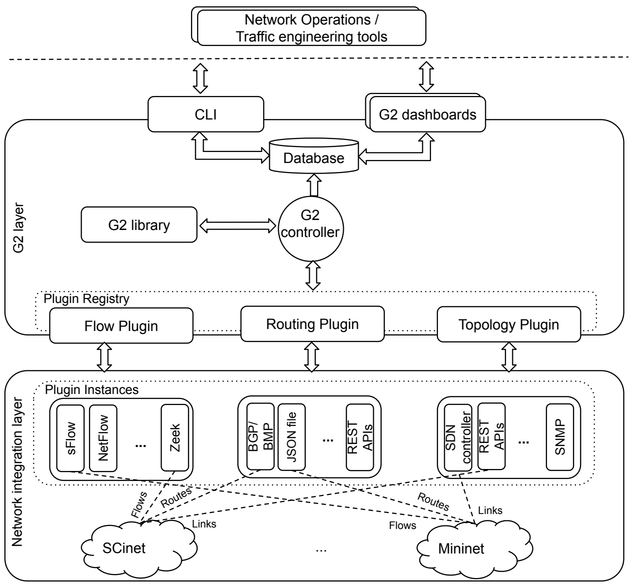


Fig. 5: G2 framework architecture.

- The *CLI* module provides a Python-based command line interface to manually or programmatically configure and monitor G2 at run-time.
- The *Plugin Registry* layer defines the APIs through which the northbound G2 modules interact with the network. This layer abstracts out the network deployment specifics from the rest of the framework using a generic set of flow, routing, and topology plugins. To successfully operationalize the framework for a given network, a suitable instance of each of these plugins must be registered with the G2 controller.

The Network Integration layer, pictured at the bottom of Fig. 5, implements the flow, routing, and topology plugin APIs for a specific production network. Fig. 5 shows various examples of these plugins. For instance, the sFlow plugin is an instantiation of the flow plugin that interacts with an sFlow agent in the network and processes sFlow records to capture real-time traffic flow information. Similarly, the BGP/BMP plugin is a routing plugin instance that can communicate with a BGP/BMP controller to learn routing information. The combination of plugins instantiated by G2 will depend on the protocols and monitoring tools used by the production network. For example, as shown in Fig. 5, to deploy G2 at the SCinet network [10], we instantiate (1) a Zeek/Splunk plugin to poll flow information based on Zeek connection logs from the SCinet's Splunk indexer, (2) a BGP/BMP plugin to poll routing information from the OpenBMP collector, and (3) a REST API based topology plugin instance to read static topology information from the topology datastore. We have developed several flow, routing, and topology plugins in due course of our integration with the various environments. This rich collection of plugins enables quick and easy integration of G2 into real-world networks.

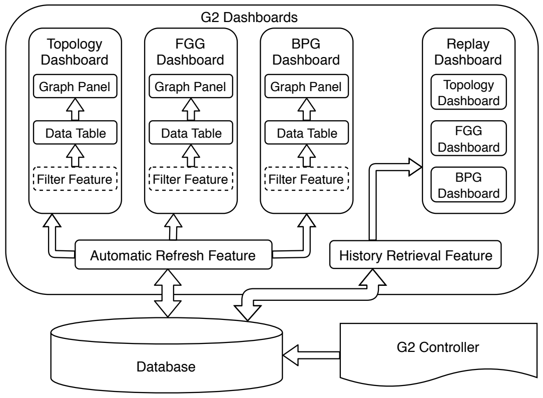
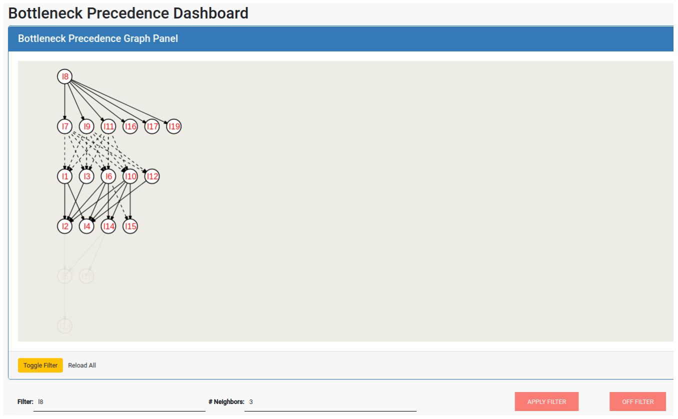
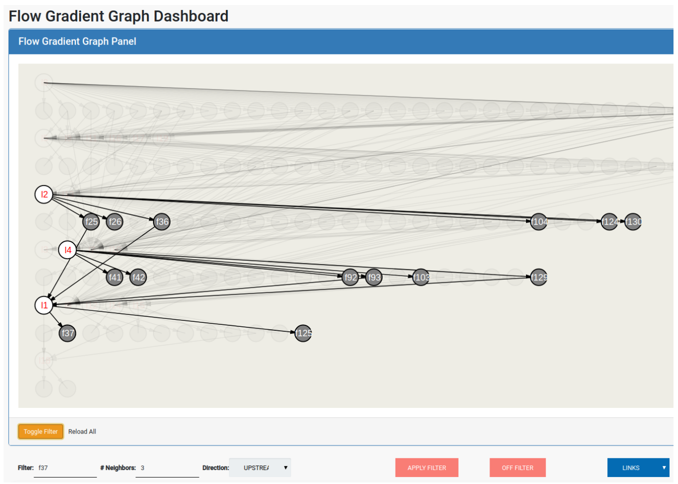


Fig. 6: G2 dashboards architecture.



Filter: l8 # Neighbors: 3

Fig. 7: BPG dashboard with filters on “l_8” and “3 neighbors”



Filter: f37 # Neighbors: 3 Direction: UPSTRE/

Fig. 8: FGG dashboard with filters on “upstream”, “3 neighbors”, and “f_{37}”.

B. Network Operator Dashboards

In this section, we discuss the design and implementation of G2 dashboards. As we mentioned previously, G2 dashboards refer to the web-based UI to the G2 framework. They provide an intuitive way to visualize, filter, and analyze flows and bottleneck links to understand their impact on the network. In particular, G2 dashboards provide network operators with powerful features such as bottleneck and flow filtering. They enable the operators to zoom in and out relevant information on any given dashboard and measure the regions of influence of these bottlenecks and flows. We implement the dashboards using the Angular framework [14].

We show the architecture of G2 dashboards in Fig. 6, which consists of four main pages:

- The *Topology dashboard* displays real-time topology-related information such as network switches, links, and flows with configurable filters to help operators focus on relevant regions of the network.
- The *Bottleneck Precedence Graph (BPG) dashboard* provides real-time visualization of the BPG graph. As described in Section II-B, this graph reveals the bottleneck structure of the network and helps understand the influence of links on the rest of the network. This dashboard hence can be an essential tool in identifying the high-impact bottleneck links and carry out online traffic engineering as well as offline capacity planning decisions. Fig. 7 illustrates a snapshot of BPG dashboard of Google’s B4 network (see Example 1) filtered to study the region of influence of link l_8 within three hops.
- The *Flow Gradient Graph (FGG) dashboard* provides real-time visualization of the interactions amongst flows. This dashboard also lets the user filter the FGG based on flows and links to study relevant sections of the network. Fig. 8 shows an FGG dashboard in which the darkly shaded circles represent flows, the lightly shaded circles represent links, and the arrows indicate the relationship between the links and flows as described in Section II-C. The FGG as shown in Fig. 8 corresponds to a snapshot of Google’s B4 network filtered to study the region in the network that can affect the performance of flow f_{37} within three hops. The FGG dashboard can be particularly useful in identifying the most influential flows and developing optimized traffic engineering actions to improve the overall performance of the network.
- The *Replay dashboard* is designed to enable a retrospective analysis of a network. By selecting the beginning and the end of a period, users can render any of the dashboards and replay all the available snapshots within the selected period. This feature can be very helpful in obtaining a historical analysis of the network, allowing to identify bottleneck links and flows that have consistently had a high impact. This information can also help operators make link-upgrade decisions and derive capacity planning recommendations.

IV. Integration of the G2 Optimization Framework in Production Networks

To demonstrate the ability of G2 to compute the bottleneck structure of a network under a variety of network configurations, we have integrated it with Mininet and other practical high-speed production networks including the ESnet testbed [9] and SCinet [10]. This section discusses the details of these integrations.

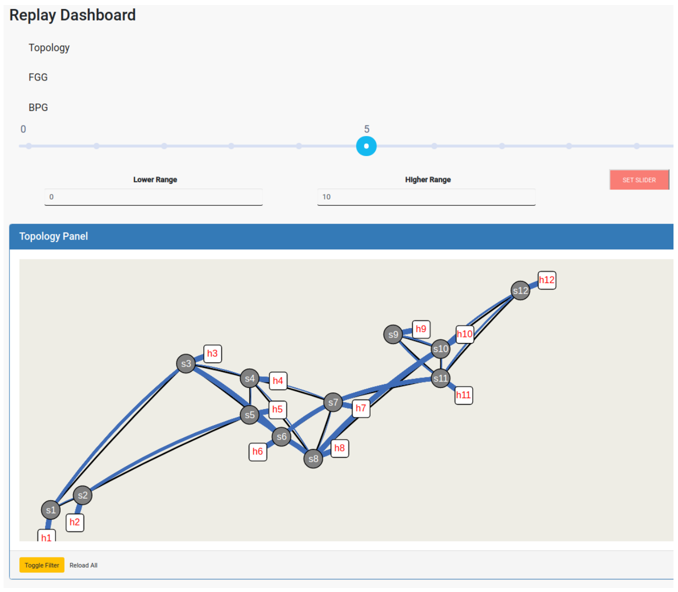


Fig. 9: Replay dashboard.

A. *Mininet Implementation*

Mininet [8] provides an ideal way to create realistic networks using OpenFlow and software-defined networking (SDN) capabilities [15]. It also includes the POX SDN framework [16] that can be used to create an SDN controller and communicate with switches. Networks emulated using Mininet run real TCP/IP, kernel, switch, and application code.

In our current approach to integrate the G2 framework with Mininet, we use sFlow [17] to obtain real-time traffic-flow information from the Mininet switching infrastructure. We pull routing and topology information from the POX SDN controller. Having access to flow, routing, and topology information, as shown in Fig. 5, the G2 library then computes BPG and FGG to facilitate the optimization of the given network.

With the goal of demonstrating the utility of G2 in network optimization operations for a variety of network configurations, we developed a Mininet sandbox [11], which we call *G2-Mininet*. The sandbox can emulate a Mininet network given the topology, routing, and traffic configurations. The topology configuration includes network nodes, links, and their capacities. Routing configuration refers to the scheme used to route traffic between network endpoints, such as shortest path routing or other schemes. Finally, traffic configuration specifies a set of traffic flows between network endpoints. The sandbox uses iPerf [18] to generate network traffic.

As part of the G2-Mininet sandbox, we also implemented a new forwarding module for POX to allow static routing on emulated network architectures. This routing module allows us to compose networks with the desired number of bottleneck levels. The sandbox also provides the ability to select a specific TCP congestion control algorithm (e.g., BBR or Cubic) and scale to larger network configurations on multi-core machines.

Essentially, the sandbox provides a flexible way to create arbitrary (1) topologies, (2) routing schemes and (3) flow configurations to analyze general network architectures with a focus—although not limited to—on understanding the bottleneck structure. We have open-sourced G2-Mininet [11] and included all the network configurations used in this paper for the Mininet and research communities.

B. *Research and Education Networks*

Research and education networks such as ESnet [9], and SCinet [10] prove to be an ideal environment to demonstrate G2's ability to integrate with realistic data networks. The following provides a summary of our current integration efforts:

- *ESnet Testbed.* ESnet is the Department of Energy's dedicated science network, providing researchers with access to a state-of-the-art SDN controlled network. We integrate G2 using a subset of the ESnet Testbed [9] consisting of a fixed topology with eleven SDN-controlled switches and four hosts located in Berkeley, Denver, Washington DC, and Atlanta. The various switches are connected using well known pre-determined routes. With this topology, we can configure arbitrary flow configurations using iPerf3 [18] to demonstrate a variety of multi-level BPG networks using G2. We also developed a tcpdump [19] based flow plugin for collecting flow information at real-time from the testbed.
- *SCinet.* SCinet is a network exclusively built every year to provide communication services at the Supercomputing Conference that aims at leveraging the highest performance network equipment available [10]. This infrastructure provides researchers the ability to test new technologies in a multi-terabit rate, SDN-controlled data network. The G2 integration for SCinet instantiates (1) a Zeek/Splunk based flow plugin to retrieve flow information using Zeek connection logs on SCinet's Splunk indexer, (2) a BGP/BMP based routing plugin to obtain up-to-date routing information, and (3) a REST API topology plugin to gain access to the network topology.

V. Experimental Results

In our experiments, we focused mainly on evaluating the hypothesis that G2 can provide optimization decisions for a variety of real-world network scenarios. To test this hypothesis, we used the sandbox G2-Mininet [11] described in Section IV-A. G2-Mininet provides a flexible way to create networks with arbitrary topologies, routing schemes, and flow configurations and deploy the G2 optimization framework with those networks.

We designed several network configurations that allowed us to test the efficacy of G2 in providing optimization decisions for different types of bottleneck structures. We emulated these configurations in G2-Mininet by varying the topology, the congestion control algorithm, and the number, sizes, and RTTs of traffic-flows.

Subsequently, we implemented a module that enables us to generate network configurations whose scale is similar to real-world networks in terms of the number of concurrent traffic

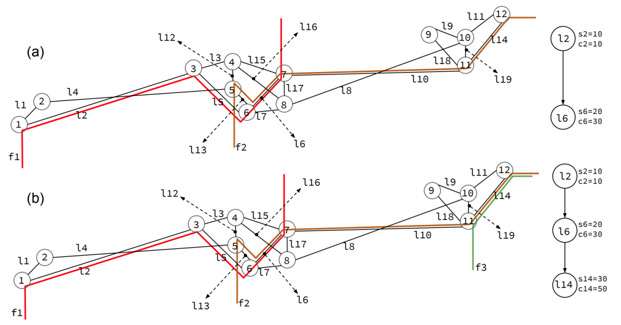


Fig. 10: Network configurations to benchmark (a) 2-level and (b) 3-level bottleneck structures.

flows. For a given base network, the scaling module replicates the traffic flows by the desired factor while keeping the source, destination, and data size fixed for each flow. This allows to scale-up network examples while preserving their topological and bottleneck-related properties.

Finally, to measure the network performance and analyze the effects of optimization by G2, we implemented modules for traffic generation and benchmark collection as part of the G2-Mininet sandbox. The statistics that we collected during the experimentation include instantaneous network throughput, flow completion times, flow convergence times, and Jain's fairness indexes [20].

The experiments presented in this section involved the generation of more than 100 network configurations by using the variations mentioned above. We used Mininet version 2.2.2 (which uses real production TCP/IP stack from the Linux kernel) running on Ubuntu Linux 16.04 LTS with kernel version 4.15.0-54. All traffic was generated running iPerf from each Mininet host.

A. The Bottleneck Structure of TCP Networks

In this first set of experiments, we aim at demonstrating that TCP networks behave according to the bottleneck structure of the network as predicted by the mathematical framework. We provide the empirical demonstration of such behavior as an initial validation test that the G2 framework can accurately characterize and predict bottleneck and flow behavior.

Towards this goal, we use Google's B4 network and compose two simple configurations, as illustrated in Fig. 10. These networks correspond to a 2-level (Fig. 10-a) and a 3-level (Fig. 10-b) bottleneck structures. Their associated BPG graphs are presented in the same figure to the right of the two networks. As shown by these BPG graphs, for the 2-level network, the fair shares of the bottleneck links are $s_2 = 10$ and $s_6 = 20$ Mbps. Thus, the expected transmission rates of the flows are $r_1 = 10$ and $r_2 = 20$ Mbps. (As explained in Section II, the theoretical rate of a flow corresponds to the fair share of its bottleneck link.) For the 3-level network, the fair shares of the bottleneck links are $s_2 = 10$, $s_6 = 20$, $s_{14} = 30$ Mbps, which leads to the flow rates $r_1 = 10$, $r_2 = 20$ and $r_3 = 30$ Mbps.

TABLE III: *Average flow throughput (Mbps) for the 2-level network.*

	BBR			Cubic		
# Flows	2	20	200	2	20	200
f_1	8.42	0.91	0.086	9.01	0.92	0.086
f_2	16.23	1.7	0.16	16.7	1.77	0.16

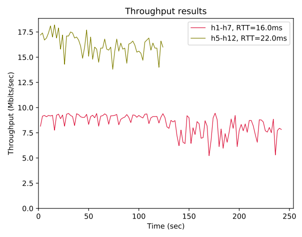
To test whether the theoretical model also holds at higher scales of traffic, we increase the number of flows between any pair of source and destination hosts by a factor of 10X and 100X. Thus, for the 2-level network, we simulate with 2, 20, and 200 flows, whereas for the 3-level network we simulate with 3, 30, and 300 flows. We performed all these experiments for both BBR and Cubic, for a total of twelve experiments.

The results are presented in Fig. 11, where flows in the legend are labeled according to their source and destination data centers. (For instance, the label $h1 - h7$ corresponds to flow f_1 in Fig. 10, which goes from data center 1 to data center 7). We also include the average throughput for each flow in Tables III and IV. As shown from these tables, for the 2-level network with 2 BBR flows, the average transmission rates are $r_1 = 8.42$ and $r_2 = 16.23$ Mbps. While they are below the fair share rates obtained from the BPG graph, $s_1 = 10$ and $s_2 = 20$ Mbps, the two flows behave according to the bottleneck structure. Similarly, for the 3-level network with 3 BBR flows, the average transmission rates are $r_1 = 8.66$, $r_2 = 15.55$ and $r_3 = 25.45$ Mbps. These rates are also slightly below the theoretical fair share, $s_1 = 10$, $s_2 = 20$ and $s_3 = 30$ Mbps, but again the flows follow the bottleneck structure by confining their transmission rates at the right level. The fact that empirical rates are below their theoretical values is attributed to imperfections in the congestion control algorithm—it is well-known that these algorithms do not achieve 100% network utilization [3]—but, as shown in this experiment, this does not invalidate the fact that TCP flows behave according to the network's bottleneck structure.

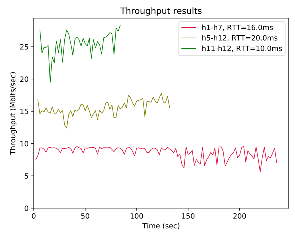
The same behavior is exposed when increasing the number of flows between a source-destination pair by 10X and 100X, as shown in Tables III and IV. Note that in these two cases, the average flow throughputs are approximately reduced by a factor of 10 and 100, respectively. This fact indicates that as the number of flows scales up, the flow throughput scales down by the same factor (since link capacity is now shared among a higher amount of flows). A key result in these experiments is that the bottleneck structure is preserved as the number of flows scales up. This behavior can also be observed graphically in Fig. 11, with flow rates reflecting the levels of the bottleneck structures. Finally, note that this same behavior is exposed by both the congestion-based algorithm (BBR) and the loss-based algorithm (Cubic).

B. Optimal Flow Policing

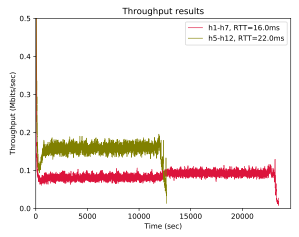
In the next experiment, we demonstrate how the proposed network optimization framework can be used to identify high impact flows. The detection of these flows is commonly used



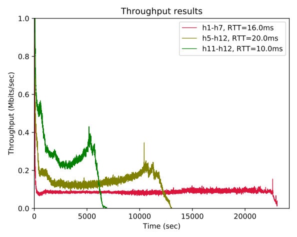
(a) 2-level / 2 BBR flows.



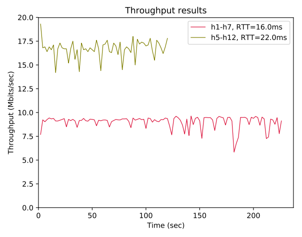
(b) 3-level / 3 BBR flows.



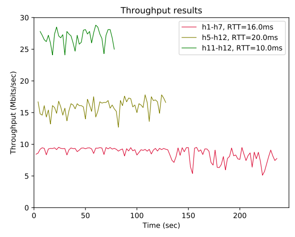
(c) 2-level / 200 BBR flows.



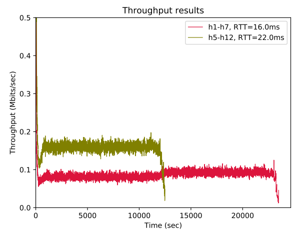
(d) 3-level / 300 BBR flows.



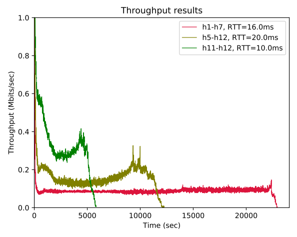
(e) 2-level / 2 Cubic flows.



(f) 3-level / 3 Cubic flows.



(g) 2-level / 200 Cubic flows.



(h) 3-level / 300 Cubic flows.

Fig. 11: Demonstration of the bottleneck structure of TCP networks using a 2-level and 3-level structures for BBR and Cubic.

TABLE IV: *Average flow throughput (Mbps) for the 3-level network.*

	BBR			Cubic		
# Flows	3	30	300	3	30	300
f_1	8.66	0.91	0.08	8.61	0.92	0.08
f_2	15.55	1.63	0.15	15.94	1.69	0.16
f_3	25.45	2.33	0.29	26.86	2.40	0.34

TABLE V: *Total throughput (Mbps) obtained when eliminating each flow.*

	BBR			Cubic		
# Flows	6	60	600	6	60	600
No flows removed	115.0	82.9	111.4	136.7	126.3	153.1
Flow f_5 removed	53.1	57.8	119.9	61.3	60.8	94.3
Flow f_6 removed	123.4	116.6	127.7	129.8	131.7	132.9

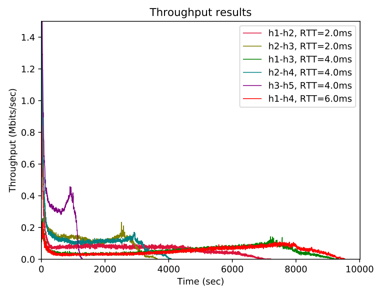
by network operators to perform traffic engineering—e.g., by re-routing or by traffic shaping such flows—in order to optimize the overall performance of a network.

To carry out this experiment, we use the network in Fig. 4. Similar to the previous section, we are interested in evaluating whether the proposed theoretical model scales up with traffic volume. Thus, we also benchmark this network by increasing the number of flows between any source-destination pair by a factor of 10X and 100X. We present the aggregate results in Table V and Fig. 12, where all the values show the average performance of all flows between a source-destination pair. (Later in Section V-D we analyze individual flow performance.) Table V shows the effect of removing flows f_5 and f_6 from the network by measuring the total network throughput as the sum of all flows' throughput. From the baseline experiment performed without removing any flows, we obtain $r_5 = 66.1$ Mbps and $r_6 = 8.04$ Mbps, slightly below the rates projected by the flow gradient graph model, $r_5 = 75$ Mbps and $r_6 = 8.3$ Mbps (see Fig. 4-b).

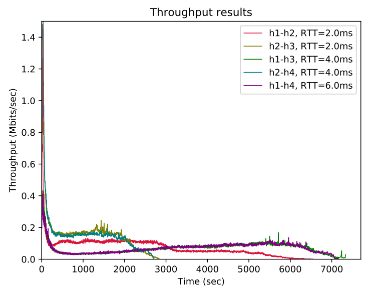
As shown in Table V, for the case of 6 flows, removing the heavy-hitter flow, f_5, reduces total network throughput from 115 Mbps to 53.1 Mbps. If instead we remove the low-hitter flow, f_6, total throughput increases to 123.4 Mbps. These result demonstrates that, against established conventional wisdom in best practices for network optimization, heavy-hitter flows may not be responsible for the higher degradation of network performance. On the contrary, in certain networks as shown in this experiment, low-hitter flows may have a much higher overall performance impact. This result is directly explained by the network's flow gradient graph (Fig. 4-b): From Lemma 2, the region of influence of flow f_5 is empty—since in the FGG it has no children vertices—whereas the region of influence of flow f_6 includes the whole network—since in the FGG there exist a path from any of the links traversed by flow f_6 to any other flow. Finally, note that in Table V the action of removing flow f_6 consistently outperforms removing flow f_5 for both BBR and Cubic and for all benchmarked cases with 6, 60 and 600 flows.

C. Optimal Link Upgrade Identification

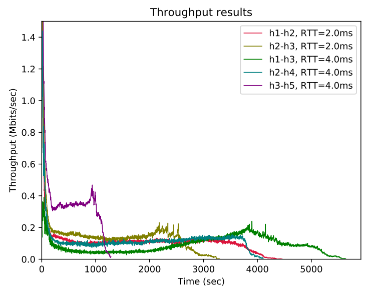
In this experiment, we seek to measure the impact each bottleneck has on the overall performance of a network. This test corresponds to the dual case of the experiments performed in Section V-B as follows: While in the previous section we reduced the rate of a flow and measured the degree to which overall network performance improved, in this section we measure the gains achieved by increasing the capacity of each bottleneck link. This analysis can be used to perform capacity planning as network operators with a fixed budget aim at prioritizing the upgrade of those links that lead to higher performance improvement. The benefit of using the BPG and



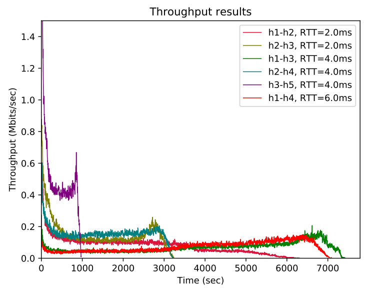
(a) BBR without removing any flow.



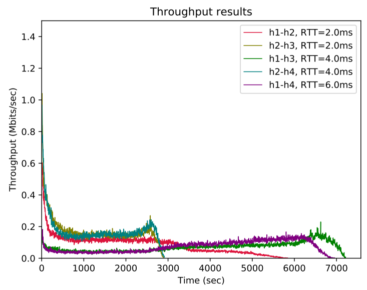
(b) BBR removing flow f_5 and replicas.



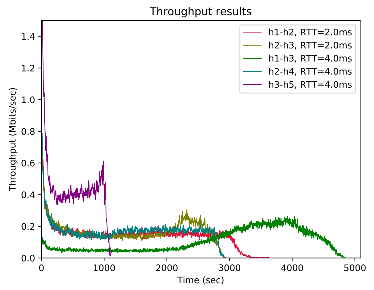
(c) BBR removing flow f_6 and replicas.



(d) Cubic without removing any flow.



(e) Cubic removing flow f_5 and replicas.



(f) Cubic removing flow f_6 and replicas.

Fig. 12: As predicted by the bottleneck structure, flow f_6 and its replicas incur the highest performance impact.

the FGG models to predict the gains from a link upgrade is that in realistic operational environments it is challenging—if not unfeasible—to run benchmarks on the same production network. Because as validated in this section (and throughout this paper) the bottleneck structure model can be used to make such predictions, network operators can leverage the capabilities of the G2 framework to perform capacity planning too.

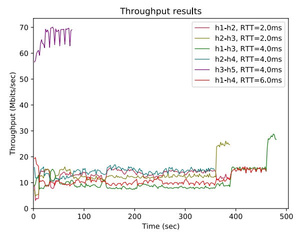
Similar to the previous section, in this experiment, we also use the network in Fig. 4-a. The proper calculation of the impact of a link upgrade requires the computation of its *bottleneck gradient* using the flow gradient graph. While illustrating how to perform this calculation is outside the scope of this paper (see instead [21]), we provide an intuitive description as follows. Considering the FGG in Fig. 4-b, we can first easily observe that upgrading link l_3 or link l_4 by adding more capacity to either of these two links produces no benefit. This is because the only flow that can be influenced by any of these links is flow f_5, but such flow is simultaneously bottlenecked at both of these links, thus in order to improve its performance, both links would need to be upgraded. If instead we consider upgrading link l_1, then flows f_1, f_3 and f_6 will experience a direct increase in their rate. Note however that the rate increase in flows f_3 and f_6 will have a negative effect on link l_2, thus reducing the rates of flows f_4 and f_2, and another negative effect on link l_3, leading also to a reduction on the rate of flow f_5. Using the proper mathematical formulation (see [21] for details), the resulting gradient of link l_1 is $1/3$. This value corresponds to the increase of the total flow throughput (the sum of all flows' rates) when link l_1 is upgraded with one additional unit of capacity. A similar exercise can be done to compute link l_2's gradient, resulting in a value of $1/2$. This analysis reveals that for the given network configuration in Fig. 4-a, an operator should prioritize upgrading first link l_2 and then link l_1, and refrain from upgrading links l_3 and l_4.

The experimental results are presented in Table VI and Fig. 13, where an upgrade on a link l is implemented by increasing the capacity of that link by ten units. (For instance, upgrading link l_1 is implemented by increasing its capacity from 25 to 35 Mbps.) We observe that as predicted by the theoretical framework, upgrading links l_3 and l_4 produces no effective benefit. Upgrading links l_1 and l_2 do lead to better total throughput, also as predicted. Cubic appears to better follow the model by yielding a higher benefit from upgrading l_2 than link l_1, in line with the fact that l_1's theoretical gradient is higher than l_2's.

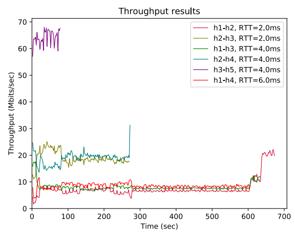
It is worth noticing that while this experiment has been simplified by assuming a fixed set of flows, in the G2 optimization framework such analysis is performed by using the network's historical flow records and taking statistical averages to obtain the links that have had the highest performance impact. Such historical record can be obtained, for instance, from NetFlow [22] or sFlow records [17], as explained in Section III-A.

D. Identification and Policing of Non-Compliant Flows

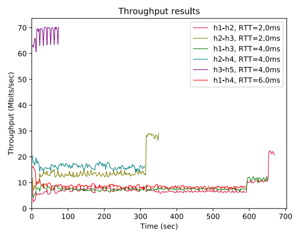
In this experiment, we demonstrate how the G2 optimization framework can be used to identify individual flows that are either non-compliant or under-performing. Because the G2 framework is a model that predicts bottleneck and flow performance based on a given optimal criterion, it can also be used as a target baseline. Comparing actual flow performance against



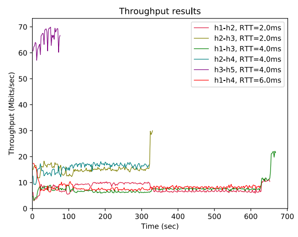
(a) Upgrading link 1 / BBR.



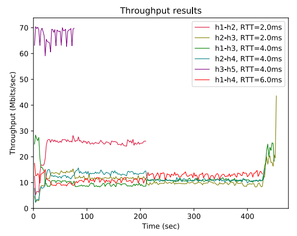
(b) Upgrading link 2 / BBR.



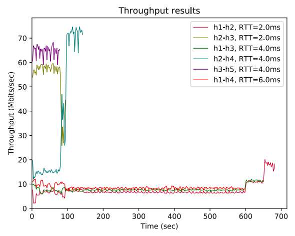
(c) Upgrading link 3 / BBR.



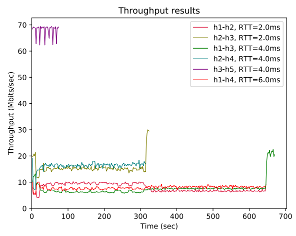
(d) Upgrading link 4 / BBR.



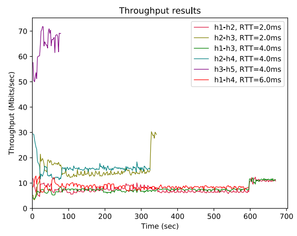
(e) Upgrading link 1 / Cubic.



(f) Upgrading link 2 / Cubic.



(g) Upgrading link 3 / Cubic.



(h) Upgrading link 4 / Cubic.

Fig. 13: As predicted by the bottleneck structure, upgrading links 3 and 4 does not provide any benefit while upgrading links 1 and 2 increases the network's total flow.

TABLE VI: *Total throughput (Mbps) obtained when upgrading a link with 10 additional units of capacity.*

	BBR	Cubic
No links upgrade	119.9	119.6
Upgrade link l_1	128.46	137.88
Upgrade link l_2	126.17	178.59
Upgrade link l_3	122.88	123.54
Upgrade link l_4	120.67	119.10

this baseline provides a mechanism to identify such outlier flows. Note that while the default G2 framework assumes a steady-state regime based on the max-min optimality condition [5], the model does not preclude using other optimization criteria such as weighted max-min or proportional fairness [23] nor extending it with additional features such as minimum or maximum flow rate constraints [12].

We carry out this experiment by leveraging the tests performed in Section V-B. Fig. 14-a and 14-b present the individual throughput of flows 1 through 6 that correspond to the experiments run with BBR and Cubic on the network in Fig. 4 and scaling the number of flows by 10X—thus a total of 60 flows. Using the FGG model shown in Fig. 4-b, flow f_1 (from $h1$ to $h2$) is expected to take a rate of $r_1 = 8.3/10 = 0.83$ Mbps and should operate within the lower bottleneck level together with flows f_3 (from h_1 to h_3) and f_6 (from h_1 to h_4). Instead, as shown in Fig. 14-a/b, this flow transmits data at a higher rate. For instance, during the interval of time from 250 to 300 seconds, flow f_1 performs at an average rate of 1.17 Mbps with peaks of 1.41 Mbps for BBR and an average rate of 1.05 with peaks of 2.39 for Cubic. Since these values are above the expected rate of 0.83 Mbps, we conclude that this flow is acting too aggressively, stealing bandwidth from the flows at its same level. This information can be used by

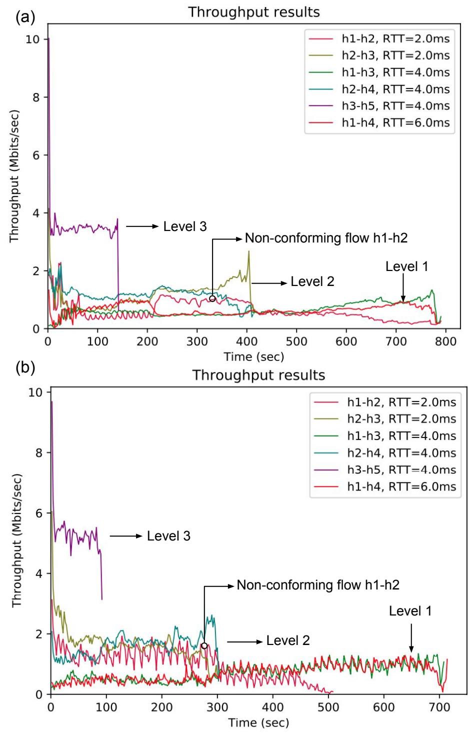


Fig. 14: For both BBR (a) and Cubic (b), flow f_1 $(h_1 - h_2)$ is able to steal bandwidth from its upper layer.

a network operator to deploy a dynamic traffic shaping policy to reduce the rate of this flow.

The aggressive performance of flow f_1 can be explained using the well-understood argument that its round trip time (2 ms, as shown in the legend of Fig. 14) is the lowest among the flows that it shares its bottleneck link l_1 with (flows f_3 and

f_6, which have RTT values of 4 ms and 6 ms, respectively). Having a lower RTT implies that this flow can converge faster and thus steal bandwidth from the rest of the flows it competes with at a given bottleneck level.

Yet the flow gradient graph also provides an alternative (complementary) explanation for flow f_1's behavior. Looking at the network's FGG in Fig. 4-b, we observe that this flow does not have any child vertices, whereas its competing flows f_3 and f_6 do have child vertices. This implies that f_3 and f_6 suffer from additional competition from flows bottlenecked at higher bottleneck levels (flows f_2, f_4 and f_5) which result in additional deterioration of their transmission rate, in benefit of flow f_1.

VI. Related Work

Flow performance optimization has been one of the most widely researched subjects in the field of communication networks. Since Jacobson introduced the first TCP-based congestion control algorithm for the Internet in 1988 [1], more than twenty algorithms have been developed and made available as part of the TCP/IP stack (e.g., [24] [6] [2]). Through this effort, it has been well-understood that the performance of a TCP flow depends exclusively on the *available capacity* at its bottleneck link and the round trip time of its communication path [1] [2]. In part constrained by the end-to-end paradigm upon which the Internet has been built [25], research has focused on the problem of bottleneck characterization from and end-system standpoint [24] [6] [2], taking a black-box model of the network. As a result, the traditional approach has left a gap in the understanding of the dynamic interactions that determine the available capacity of a flow's bottleneck link. As also recently recognized by Lavanya and McKeown [4], bottleneck links influence each other in complex ways, potentially weaving through all links in the network. In [5], the authors introduce the Theory of Bottleneck Ordering, the first mathematical framework that aims at tackling this problem. This proposed framework leads to the concept of *bottleneck structure*, one of the core building blocks of the G2 network optimization framework that we propose in this paper.

While not bound to it, the Theory of Bottleneck Ordering assumes an underlying optimization model based on the max-min criterion. Bertsekas and Gallager [7] were among the first to propose and study max-min as a framework to perform network congestion control and presented the original water-filling algorithm to compute its solution. Ros-Giralt and Tsai generalized Bertsekas algorithm by providing a parallelizable version of the water-filling algorithm in which links converge to their fair share concurrently according to the bottleneck structure of the network [12]. This algorithm is the basic building block used in [5] to develop the Theory of Bottleneck Ordering.

VII. Conclusions

This paper introduces G2, a new network optimization framework that leverages the recently introduced Theory of Bottleneck Ordering to help network operators optimize bottleneck and flow performance. A core building block of the proposed framework is the bottleneck precedence graph (BPG), a digraph that reveals the bottleneck structure of a network and provides a model to (1) characterize and predict the bottlenecks' fair share and the flows' transmission rate and (2) understand (qualitatively and quantitatively) the ripple effects that perturbations on the performance of a bottleneck link can cause to other regions of the network. We have implemented the G2 optimization framework as a modular architecture consisting of: (1) A core library implementing the algorithms, (2) a user and a graphical interface (UI/GUI) that allows network operators to visualize bottlenecks, flows and compute the optimized traffic engineering policies, and (3) an API consisting of a set of plugins that facilitates the integration of G2 into modern real networks.

We present initial experiments that illustrate how the theoretical framework—and thus, operationally the G2 platform—can be used to gain critical insights on the performance of bottlenecks and flows. For instance, we empirically show how the framework can be used to identify flows that, despite being low-hitters, have a high negative impact on the overall performance of the network. Similarly, we show how the same framework can be used to identify critical bottlenecks as a tool for network capacity planning.

While in this paper we focus primarily on describing the analytical capabilities of G2 based on the bottleneck precedent graph (BPG), forthcoming work includes the development of a parallel set of the optimization features based on the flow gradient graph (FGG). Both of these frameworks are dual to each other, and together they enable a complete framework to optimize the performance of bottlenecks and flows. Current work also includes continuing to augment the capabilities of G2 to make it a production-ready real-time network optimization and capacity planning framework.

References

[1] V. Jacobson, “Congestion Avoidance and Control,” in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, 1988, pp. 314–329.

[2] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, “BBR: Congestion-Based Congestion Control,” *ACM Queue*, vol. 14, no. 5, pp. 50:20–50:53, 2016.

[3] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, “The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm,” *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 3, pp. 67–82, 1997.

[4] L. Jose, L. Yan, M. Alizadeh, G. Varghese, N. McKeown, and S. Katti, “High Speed Networks Need Proactive Congestion Control,” in *14th ACM Workshop on Hot Topics in Networks*, 2015, pp. 14:1–14:7.

[5] J. Ros-Giralt, A. Bohara, S. Yellamraju, M. H. Langston, R. Lethin, Y. Jiang, L. Tassiulas, J. Li, Y Tan, M. Veeraraghavan, “On the Bottleneck Structure of Congestion-Controlled Networks,” *To appear at ACM SIGMETRICS*, 2020, Boston, USA.

[6] S. Ha, I. Rhee, and L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant,” *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[7] D. P. Bertsekas, R. G. Gallager, and P. Humblet, *Data Networks*. Prentice-Hall International New Jersey, 1992, vol. 2.

[8] “Mininet,” [Online]. Available: http://mininet.org/ (Retrieved September 10, 2019).

[9] “ESnet Testbed,” [Online]. Available: http://es.net/network-r-and-d/experimental-network-testbeds/test-circuit-service/ (Retrieved September 10, 2019).

[10] "The SCinet Network at Supercomputing," [Online]. Available: https://sc19.supercomputing.org/scinet/all-about-scinet/ (Retrieved September 10, 2019).

[11] "Reservoir Labs G2-Mininet Sandbox: Mininet extensions to support the analysis of the bottleneck structure of networks," [Online]. Available: https://github.com/reservoirlabs/g2-mininet (Retrieved September 10, 2019).

[12] J. Ros and W. K. Tsai, "A Theory of Convergence Order of Maxmin Rate Allocation and an Optimal Protocol," in *IEEE INFOCOM 2001*, vol. 2, 2001, pp. 717–726.

[13] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, and et al., "B4: Experience with a Globally-Deployed Software Defined WAN," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[14] "Angular Web Framework," [Online]. Available: https://angular.io (Retrieved September 10, 2019).

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[16] "The POX Network Software Platform," [Online]. Available: https://noxrepo.github.io/pox-doc/html/ (Retrieved September 10, 2019).

[17] P. Phaal, S. Panchen, and N. McKee, "sFlow Specifications, InMon Corporation," *IETF RFC 3176*, 2001.

[18] "iPerf," [Online]. Available: https://iperf.fr/ (Retrieved September 10, 2019).

[19] "Tcpdump (TCPDUMP Tool)," [Online]. Available: https://www.tcpdump.org/ (Retrieved September 10, 2019).

[20] R. Jain, D.-M. Chiu, and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems," *arXiv preprint cs/9809099*, 1998.

[21] "Reservoir Labs Mathematical Essay: The Theory of Bottleneck Ordering (available upon request)," 2019.

[22] B. Claise, G. Sadasivan, V. Valluri, and M. Djernaes, "NetFlow Specifications, Cisco Systems," [Online]. Available: https://www.ietf.org/rfc/rfc3954.txt (Retrieved September 10, 2019).

[23] F. P. Kelly, A. K. Maulloo, and D. K. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[24] K. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 5–21, 1996.

[25] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end Arguments in System Design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, 1984.