On the Bottleneck Structure of Congestion-Controlled Networks

JORDI ROS-GIRALT*, ATUL BOHARA†, SRUTHI YELLAMRAJU, M. HARPER LANGSTON, and RICHARD LETHIN, Reservoir Labs, USA
YUANG JIANG and LEANDROS TASSIULAS, Yale Institute of Network Science, USA
JOSIE LI, YUANLONG TAN, and MALATHI VEERARAGHAVAN, University of Virginia, USA

In this paper, we introduce the *Theory of Bottleneck Ordering*, a mathematical framework that reveals the bottleneck structure of data networks. This theoretical framework provides insights into the inherent topological properties of a network in at least three areas: (1) It identifies the regions of influence of each bottleneck; (2) it reveals the order in which bottlenecks (and flows traversing them) converge to their steady state transmission rates in distributed congestion control algorithms; and (3) it provides key insights into the design of optimized traffic engineering policies. We demonstrate the efficacy of the proposed theory in TCP congestion-controlled networks for two broad classes of algorithms: Congestion-based algorithms (TCP BBR) and loss-based additive-increase/multiplicative-decrease algorithms (TCP Cubic and Reno). Among other results, our network experiments show that: (1) Qualitatively, both classes of congestion control algorithms behave as predicted by the bottleneck structure of the network; (2) flows compete for bandwidth only with other flows operating at the same bottleneck level; (3) BBR flows achieve higher performance and fairness than Cubic and Reno flows due to their ability to operate at the right bottleneck level; (4) the bottleneck structure of a network is continuously changing and its levels can be folded due to variations in the flows' round trip times; and (5) against conventional wisdom, low-hitter flows can have a large impact to the overall performance of a network.

CCS Concepts: • **Networks** → *Network resources allocation*; *Traffic engineering algorithms*; *Network performance analysis*;

Keywords: Traffic engineering, congestion control, bottleneck link, max-min

ACM Reference Format:
Jordi Ros-Giralt, Atul Bohara, Sruthi Yellamraju, M. Harper Langston, Richard Lethin, Yuang Jiang, Leandros Tassiulas, Josie Li, Yuanlong Tan, and Malathi Veeraraghavan. 2019. On the Bottleneck Structure of Congestion-Controlled Networks. In *Proc. ACM Meas. Anal. Comput. Syst.*, Vol. 3, 3, Article 59 (December 2019). ACM, New York, NY. 31 pages. https://doi.org/10.1145/3366707

*Work partially funded by the Department of Energy under contract DE-SC0019523.
†Atul Bohara was an intern at Reservoir Labs and a PhD candidate at the University of Illinois at Urbana-Champaign when this paper was published.

Authors' addresses: Jordi Ros-Giralt, giralt@reservoir.com; Atul Bohara, bohara@reservoir.com; Sruthi Yellamraju, yellamraju@reservoir.com; M. Harper Langston, langston@reservoir.com; Richard Lethin, lethin@reservoir.com, Reservoir Labs, 632 Broadway, Suite 803, New York, New York, USA, 10012; Yuang Jiang, yuang.jiang@yale.edu; Leandros Tassiulas, leandros.tassiulas@yale.edu, Yale Institute of Network Science, New Haven, Connecticut, USA, 06511; Josie Li, jl9gf@virginia.edu; Yuanlong Tan, yt4xb@virginia.edu; Malathi Veeraraghavan, mv5g@virginia.edu, University of Virginia, Charlottesville, Virginia, USA, 22904.

2476-1249/2019/12-ART59
https://doi.org/10.1145/3366707

1 INTRODUCTION

Congestion control algorithms such as those implemented in TCP/IP stacks focus on two general objectives: (1) Maximizing network utilization and (2) ensuring fairness among flows competing for network bandwidth [3]. For the past three decades since the first congestion control algorithm was implemented as part of the TCP protocol [13], many different variations have been designed, implemented and extensively put into practice to help address these two objectives. In this context, it is well-known that regardless of the complexity of a communication path, the performance of a flow is uniquely determined by the capacity of its bottleneck link and its end-to-end round trip time (RTT) [2, 13]. This funnel view has steered much of the research towards the single-bottleneck characterization problem (e.g., [3], [13], [2], [25]), leading to the implicit assumption that bottlenecks in a network have a *flat structure*, and inadvertently hiding the need for a better understanding of bottlenecks in distributed networks.

While the problem of characterizing the influences and relationships existing between bottlenecks has not been addressed from a formal and practical standpoint, it has naturally not been fully omitted from the literature. For instance, in [16] the authors refer to the bottleneck relationships as *dependency chains* and observe that performance perturbations of bottleneck links may affect other bottleneck links, "potentially weaving through all links in the network", but do not address the problem of formally identifying the hidden structure that controls such perturbations.

To address this subject, in this paper we introduce the Theory of Bottleneck Ordering, a framework that, by revealing the bottleneck structure of a network, describes (qualitatively and quantitatively) the influence that bottlenecks exert on each other. We then use three well-known congestion control algorithms to validate whether bottlenecks and flows in real networks behave as predicted by the theory. To get a broader sense of how the theory performs, we choose representatives from two widely used classes of congestion control algorithms: BBR from the class of congestion-based algorithms [2], and Cubic and Reno [7, 10] from the class of loss-based additive-increase/multiplicative-decrease (AIMD) algorithms. The key contributions of this paper can be summarized as follows:

- Real networks qualitatively behave as predicted by the Theory of Bottleneck Ordering. Thus, the proposed framework can be used as a tool to understand bottleneck and flow performance. (Section 3.1.)
- Bottlenecks do not interact and influence each other following a flat structure, but rather through a structure described by the *bottleneck precedence graph* (BPG) as introduced in this paper. (Sections 2.4 and 3.1.)
- Similarly, flows interact and influence each other through a structure described by the *flow gradient graph*, also introduced in this paper. (Sections 2.6 and 3.3.)
- Congestion control algorithms that can identify the bottleneck structure of the network perform significantly better in key metrics such as (1) flow completion time, (2) fairness, and (3) total throughput. The proposed theory can thus be used as a framework to evaluate the performance of various congestion control algorithms, helping to understand their capabilities and limitations in a rigorous approach, and providing a benchmark against an optimal baseline. (Sections 2.5, 3.1, 3.2, and 3.4.)
- Differences in the round trip time can distort the bottleneck structure of a network. Depending on such differences, congestion-window based algorithms such as Cubic and Reno that are sensitive to RTT may not be able to identify the bottleneck structure; as a result, they perform poorly in the above-mentioned key performance metrics. (Section 3.2.)
- Congestion-based algorithms such as BBR that are more resilient to variations of RTT can identify the bottleneck structure and thus achieve significantly better performance. This

insight provides a new formal explanation to understand the reason why such a class of algorithms may perform better [2]. (Sections 3.1 and 3.2.)
- The BPG graph reveals with precision the bounded regions of influence that bottlenecks and flows exert on each other. (Sections 2.4 and 3.3.)
- The convergence time of a congestion control algorithm depends on both (1) the number of levels in the bottleneck structure and (2) the number of flows that compete with each other in each independent bottleneck level. (Sections 2.5 and 3.4.)
- Finally, against the conventional practice, we show (mathematically and experimentally) that low-hitter flows can have a significantly higher impact on the overall performance of the network than heavy-hitter flows. (Sections 2.6 and 3.3.)

This paper is organized as follows. The Theory of Bottleneck Ordering is formally developed in Section 2. In 2.1 and 2.2, we introduce its core concepts by using some initial simple examples and develop the idea of bottleneck information as a distributed transmission problem. Section 2.3 formally introduces the concepts of direct and indirect precedent link relations, the key building blocks to construct the bottleneck structure of a network. This allows us to introduce and analyze the bottleneck precedence graph (BPG) algorithm in Section 2.4, the procedure that computes the bottleneck structure of a network. In 2.5, we establish the mathematical connection between the bottleneck structure of a network and the minimum convergence time of a distributed congestion control algorithm, while in 2.6 we extend the bottleneck structure model to include flow information. Section 3 provides experiments to validate the existence of bottleneck structures in modern data networks and to illustrate how the Theory of Bottleneck Ordering can be used to optimize network performance. Assumptions, generalizations, and the practical implications of our work are presented in Section 4, while Section 5 summarizes related work. We conclude the paper by summarizing the main results and presenting future lines of research in Section 6.

2 THE THEORY OF BOTTLENECK ORDERING

2.1 Transmission of Bottleneck Information

It is well-known that regardless of how many links a connection traverses, "from TCP's viewpoint an arbitrarily complex path behaves as a single link with the same round-trip time and bottleneck rate" [2, 13]. What is less well-understood is the fact that not all bottleneck links are of equal importance. To provide some intuition to this argument, let us consider some initial simple network examples.

Consider the network configurations illustrated in Figures 1-a, 2-a and 3-a. In these three drawings, 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. For instance, Figure 1-a corresponds to the case of a single bottleneck link, l_1, with two flows, f_1 and f_2. Since the capacity of the link is $c_1 = 1$ bps, each flow can transmit data at 0.5 bps. This value is typically referred as the link's *fair share* [3], which we denote as $s_1 = 0.5$ bps. Thus, we trivially have $r_1 = r_2 = s_1 = 0.5$ bps. Note that while other allocation schemes could result in flow transmission rates different than the fair share, since the existence of bottleneck links is a fundamental invariant in all of these schemes, the theoretical framework described in this paper is applicable without loss of generality. The bottleneck structure of a single link network is thus a trivial graph with just one vertex corresponding to the only link, as shown in Figure 1-b.

Consider now the network configuration in Figure 2-a, corresponding to two links and three flows. Link l_1 is the most constrained since it has a lower fair share than link l_2 ($s_1 = 0.5$ while $s_2 = 1$). Thus, the transmission rates of flows f_1 and f_2 are determined by this link, taking a value equal to its fair share: $r_1 = r_2 = s_1 = 0.5$. Once these two flow rates settle, then flow f_3's rate can be

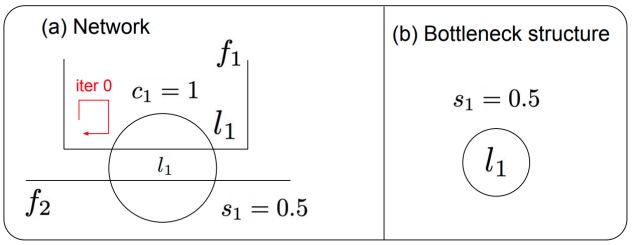


Fig. 1. Bottleneck structure of a trivial single-link network.

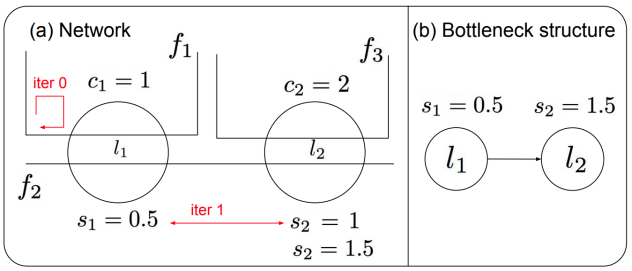


Fig. 2. A 2-level bottleneck structure with direct precedence.

determined as follows: since flow f_2 has a rate $r_2 = 0.5$, flow f_3 will take the excess capacity left in link l_2, corresponding to $s_2 = 2 - 0.5 = 1.5$. In other words, since flow f_3 is bottlenecked at link l_2, f_3's rate corresponds to link l_2's fair share after the rates of all other flows, which go through l_2 and are bottlenecked at another link, have been subtracted from its capacity. (Note that this rate assignment is also known in the literature as the max-min solution and can be computed using a water-filling algorithm [1].)

While this network configuration is still very simple, an interesting question arises from its topological structure: Are both bottleneck links l_1 and l_2 equally important with regards to the overall performance of the network? 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, $\partial s_1/\partial c_2 = 0$. This implies the performance of link l_2 (measured in terms of its fair share) 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. The bottleneck structure we formally introduce in this paper is a graph that captures these relationships.

An intuitive way of understanding the bottleneck structure of a network involves modeling the bottleneck links as nodes communicating with each other, where these nodes implement a distributed congestion control algorithm to determine their fair share while using the smallest possible number of iterations. In Figure 2-a, we can say that link l_2 cannot converge to its fair share value, $s_2 = 1.5$, until link l_1 has converged to its own, $s_1 = 0.5$. Thus, in order to settle its own fair share, s_2, link l_2 has to wait until link l_1 broadcasts its fair share, s_1. This leads to a bottleneck structure consisting of two nodes, l_1 and l_2, and a directed edge from l_1 to l_2 denoting the communication that must occur so that the bottlenecks can settle their fair shares. This two-level bottleneck structure is shown in Figure 2-b.

Consider yet another network configuration consisting of three links and four flows, as shown in Figure 3-a. Using our communication model, the first bottleneck link to settle its fair share is l_1 with

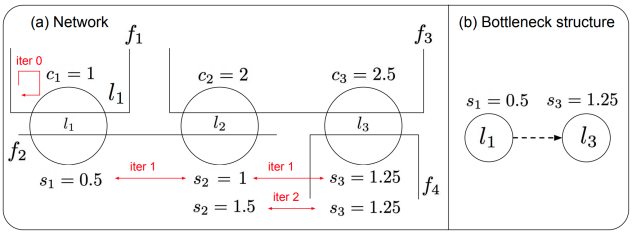


Fig. 3. A 2-level bottleneck structure with indirect precedence.

a value of $s_1 = 0.5$. This implies that flows f_1 and f_2 get a transmission rate of $r_1 = r_2 = s_1 = 0.5$. Once link l_1's fair share is communicated to links l_2 and l_3, link l_3 becomes the next bottleneck with a fair share of $s_3 = 1.25$ (since this value is lower than link l_2's fair share of $s_2 = 1.5$). Consequently, flows f_3 and f_4 get a transmission rate of $r_3 = r_4 = s_3 = 1.25$. Note that link l_2 is not a bottleneck link in this network, since its total flow is lower than its capacity, $r_2 + r_3 = 1.75 < 2$. Thus, the resulting bottleneck structure consists of two nodes for links l_1 and l_3 and a directed edge between them as shown in Figure 3-b.

While the bottleneck structures of the second and third examples are apparently similar (both have two bottleneck links connected by an edge), there is a subtle yet important difference: In the second example (Figure 2), there exists a flow that traverses both bottlenecks (flow f_2 goes through both links l_1 and l_2) while in the third example (Figure 3), the two bottlenecks (l_1 and l_3) don't share any flow. The latter case is important as it demonstrates that bottleneck information can travel between two links even if there is no direct communication path between them. (In the next section we mathematically prove this result.) In this case, links l_1 and l_2 are connected via flow f_2 while links l_2 and l_3 are connected via flow f_3, enabling a bottleneck communication path between links l_1 and l_3 via link l_2. To differentiate the two types of communication paths between links illustrated in Figures 2-b and 3-b, we use the terms *direct precedence* and *indirect precedence*, respectively, and denote this in the bottleneck structure graph by using a solid and a dashed edge connecting the two links. In the theoretical framework introduced by this paper, we refer to the bottleneck structure of a network as its *bottleneck precedence graph (BPG)*, which we formally introduce in Section 2.4).

Note that the BPG also reveals the number of iterations required by links to converge to their fair share. For instance, while two links related by direct precedence can communicate in one iteration (Figure 2), links that are connected via indirect precedence require two iterations, since the bottleneck information needs to be conveyed via a *relay* link (Figure 3). In the next sections, we mathematically prove that the BPG graph is also an effective tool to measure the minimum convergence time required by any distributed congestion control algorithm.

The above examples helped us to informally introduce the bottleneck structure of some simple network configurations providing initial intuition behind the existence of (1) bottleneck hierarchies, (2) the notion of convergence time for a distributed congestion control algorithm, and (3) the concept of direct and indirect precedence between two bottlenecks. In the next sections, we formalize all these intuitive concepts into a mathematical theory of bottleneck ordering and introduce an algorithm to compute the bottleneck precedence graph for arbitrary network topologies.

2.2 Parallel Convergence in the Water-Filling Algorithm

We start our work with an implicit assumption: In steady state, all congestion-controlled flows are bottlenecked at least at one link. The general intuition for this statement is that all congestion control algorithms—by their fundamental objective—are designed not to leave unused bandwidth.

This fact is certainly true for algorithms implemented as part of a TCP/IP stack. The second observation we make is that, as shown in the previous section, the max-min optimal allocation provides a natural way to identify the bottleneck link of every flow in steady state. Intuitively, a flow cannot get a rate above its bottleneck's fair share as that would be a capacity violation if all other flows acted in the same way. Thus, in what follows, we use the max-min assignment to analyze the influence that the bottlenecks exert on each other and reveal the topological structure interrelating them. Two notes are relevant regarding the assumption of *steady-state optimal network regime* implicit in our theoretical framework. First, while we leave it outside the scope of this paper, the same general bottleneck principles apply to other allocation schemes different than max-min (for instance, weighted max-min or proportional fairness [17]). The bottleneck structure in these other schemes might be different, but the concept of bottleneck structure is universal and applicable to them too. Secondly, as we show in our experiments (Section 3), TCP's congestion control (a widely used algorithm in modern networks) does qualitatively follow the bottleneck structure revealed by the max-min assumption, making this choice highly practical.

Next, we describe an algorithm to compute the max-min rate allocation of a network. The original algorithm, called the *water-filling Algorithm*, was introduced by Bertsekas and Gallager in [1]. To develop the theoretical framework, however, we use a variant of the water-filling algorithm introduced by Ros-Giralt and Tsai in [27]. This variant, presented in Algorithm 1, exploits a parallelization property in the original algorithm to allow certain bottleneck links to converge concurrently, leading to a lower execution time when run on a parallel computing architecture.

Algorithm 1 FastWaterFilling (Inputs: $\mathcal{L}, \mathcal{F}, \{\mathcal{F}_l, \forall l \in \mathcal{L}\}, \{c_l, \forall l \in \mathcal{L}\}$)

$\mathcal{L}$:= *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;
$\mathcal{B}$:= *Set of bottleneck links*;
r_f := *Rate of flow* f;
$\mathcal{L}^k$:= *Set of unresolved links at iteration* k;
C^k := *Set of converged flows at iteration* k;
$\mathcal{L}^0 = \mathcal{L}; C^0 = \{\emptyset\}$;
$k = 0$;
while $C^k \neq \mathcal{F}$ **do**
  $s_l^k = (c_l - \sum_{\forall f \in C^k \cap \mathcal{F}_l} r_f)/|\mathcal{F}_l \setminus C^k|, \forall l \in \mathcal{L}^k$;
  $u_l^k = min\{s_{l'}^k \mid \mathcal{F}_{l'} \cap \mathcal{F}_l \neq \{\emptyset\}, \forall l' \in \mathcal{L}^k\}, \forall l \in \mathcal{L}^k$;
  for $l \in \mathcal{L}^k, s_l^k = u_l^k$ **do**
    $r_f = s_l^k, \forall f \in \mathcal{F}_l \setminus C^k$;
    $\mathcal{L}^k = \mathcal{L}^k \setminus \{l\}$;
    $C^k = C^k \cup \{f, \forall f \in \mathcal{F}_l \setminus C^k\}$;
  end for
  $\mathcal{L}^{k+1} = \mathcal{L}^k; C^{k+1} = C^k$;
  $k = k + 1$;
end while
$\mathcal{B} = \mathcal{L} \setminus \mathcal{L}^k; s_l = s_l^k, \forall l \in \mathcal{B}$;
return $\langle \mathcal{B}, \{r_l, \forall f \in \mathcal{F}\}, \{s_l, \forall l \in \mathcal{B}\}, k \rangle$;

The algorithm tracks the set of unresolved links $\mathcal{L}^k$ (links whose final fair shares have not yet been determined) and the set of converged flows C^k (flows whose final transmission rates have been determined). Its functioning also relies on two parameters:

- The *fair share* of link l at iteration k: s_l^k. As introduced in the previous section, this corresponds to the fair share of a link after removing all flows that have converged up to iteration k and is computed in line 12.
- The *upstream fair share* of link l at iteration k: u_l^k. This parameter corresponds to the minimum fair share among all links sharing a flow with l and is computed in line 13.

At each iteration k, for each unresolved link l that has a fair share equal to its upstream fair share, $s_l^k = u_l^k$ (line 14), three actions are taken: (1) The transmission rate of the remaining flows going through link l is assigned to the fair share value (line 15), (2) link l is removed from the set of unresolved links (line 16), and (3) all remaining flows going through it are added to the set of converged flows (line 17). As shown in [27], this process ends when all flows and bottleneck links have converged to their final transmission rate and fair share values, respectively. Upon completion, the algorithm returns the set of bottleneck links $\mathcal{B}$, the transmission rate of all flows $\{r_l, \forall f \in \mathcal{F}\}$, the fair share of all bottleneck links $\{s_l, \forall l \in \mathcal{B}\}$, and the value of k, corresponding to the depth of the bottleneck structure as we demonstrate in the next section.

The FastWaterFilling algorithm exploits the following property formally proven in [26]: A link that has the minimum fair share among all links with which it shares flows, can immediately converge to its final fair share. In general, because more than one link satisfies this property at each iteration k, this allows multiple links to converge concurrently at the same iteration. In [26] the author shows that on average this approach reduces the number of iterations from $O(|\mathcal{L}|)$ in Bertseka's original water-filling algorithm down to $O(log(|\mathcal{L}|))$. In this paper, we observe that this parallelization property captures the exact natural convergence time of bottleneck links in real networks. Our observation relies on the fact that modern networks use distributed congestion control algorithms, such as TCP BBR, Cubic, and Reno, that effectively behave as a large parallel computer. As we show next, this property plays a crucial role in understanding the bottleneck structure of a network.

2.3 Precedent Link Relations

A more detailed analysis of the FastWaterFilling algorithm shows that the order of execution of its *while* loop reveals a bottleneck structure that is unique to every network. In this *hidden* structure, bottleneck links relate to each other by following well defined mathematical relationships that describe how (and in what magnitude) one link can affect the performance of another link. In particular, we show there exists only two essential relationships between bottlenecks, *direct precedence* and *indirect precedence*, which we formally introduce as follows:

Definition 2.1. Direct precedence. Let l and l' be two links such that (1) l converges at iteration k, (2) l' converges at iteration k', for some $k' > k$, and (3) $\mathcal{F}_l \cap \mathcal{F}_{l'} \neq \{\emptyset\}$. Then, we say that link l is a *direct precedent link* (or simply a *direct precedent*) of link l'.

Definition 2.2. Indirect precedence. Let l and l' be two links such that (1) l converges at iteration k, (2) l' converges at iteration k', for some $k' > k$, and (3) $\mathcal{F}_l \cap \mathcal{F}_{l'} = \{\emptyset\}$ but there exists another link l_r such that (4.1) $\mathcal{F}_l \cap \mathcal{F}_{l_r} \neq \{\emptyset\}$ and $\mathcal{F}_{l'} \cap \mathcal{F}_{l_r} \neq \{\emptyset\}$, (4.2) $s_{l'}^k > s_{l_r}^k$ and (4.3) l_r converges at an iteration k'', for some $k'' > k'$. Then, we say that link l is an *indirect precedent link* (or simply an *indirect precedent*) of link l', and we refer to link l_r as the *relay link* between links l and l'.

The relevancy of these two definitions is justified in the next two lemmas, which state the order of bottleneck convergence and the degree to which one bottleneck can affect the performance of another one in arbitrary networks.

Lemma 2.3. *Bottleneck convergence order. A link l is removed from the set of unresolved links $\mathcal{L}^k$ at iteration k, $\mathcal{L}^k = \mathcal{L}^k \setminus \{l\}$, if and only if all of its direct and indirect precedent links have already been removed from this set at iteration $k-1$.*

Proof. See Appendix A.3. □

This lemma introduces the notion of *bottleneck link convergence*. We use this term to indicate that a link's fair share has been resolved and will no longer change as the FastWaterFilling algorithm continues iterating. From Algorithm 1, it is easy to see that a link converges when it is removed from the set of unresolved links in line 16.

Intuitively, Lemma 2.3 says that for a link l to converge, all of its direct and indirect precedent links must have converged first (this is the necessary condition). Moreover, it also says that if k is the highest iteration at which any of its direct and indirect precedent links converges, then link l will converge immediately after it at iteration $k+1$ (this is the sufficient condition). The importance of this lemma lies in the fact that it reveals the hidden bottleneck structure of the network. Bottleneck links in a network are not all equal with respect to the performance impact they exert on each other. On the contrary, they follow a well-defined structure uniquely characterized by the topological properties of the network. The following lemma characterizes the influence of bottlenecks onto each other:

Lemma 2.4. *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 $\{l_1, l_2, ..., l_n\}$ 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 Appendix A.4. □

Note that in the above lemma we capture the influence of a link l against another link l' using the derivative $\partial s_{l'}/\partial c_l$. That is, if a change on the effective capacity c_l of link l changes the fair share $s_{l'}$ of another link l', then we say that link l influences link l'. The lemma states that a bottleneck link l influences another bottleneck link l' if there exists a set of direct precedent links that form a path between l and l'.

In Figure 2 we saw an example of bottleneck influence: as stated by the lemma, link l_1 can influence link l_2 ($\partial s_2/\partial c_1 \neq 0$) but not vice versa ($\partial s_1/\partial c_2 = 0$), since there is a path of direct precedents from link l_1 to l_2 (but not a reverse path). Note also that in Figure 3 we have $\partial s_3/\partial c_1 = 0$, since bottlenecks l_1 and l_3 are related via an indirect precedence, since the lemma only works for direct precedences.

In the next section, we provide an algorithm to compute all the precedent links to help construct the bottleneck structure of arbitrary networks and a detailed example to illustrate the practical implications of Lemmas 2.3 and 2.4.

2.4 The Bottleneck Structure of Networks

To compute the bottleneck structure of a network, we need to obtain the direct and indirect precedent links of every bottleneck link. These precedent links correspond to edges in a digraph that has all the bottleneck links as its vertices, revealing the inherent mesh structure that characterizes the relationships between the bottlenecks and how they influence each other. This structure can be formally defined as follows:

Definition 2.5. Bottleneck precedence graph. We define the *bottleneck precedence graph* (BPG) as a tuple $\langle V, E \rangle$ of vertices V and edges E such that $V = \mathcal{B}$ and $E = \{\mathcal{D}_l, \forall l \in \mathcal{B}\} \cup \{\mathcal{I}_l, \forall l \in \mathcal{B}\}$, where $\mathcal{D}_l$ and $\mathcal{I}_l$ are the sets of direct and indirect precedents of link l, respectively. To differentiate them, we typically represent edges corresponding to direct and indirect precedents links with solid and

dashed lines, respectively. We also equivalently refer to the BPG graph as the *bottleneck structure of a network*.

As noted in Section 2.2, the FastWaterFilling algorithm already computes the set of bottleneck links $\mathcal{B}$. Thus, to obtain the bottleneck structure, we can extend this algorithm with additional logic to compute the direct and indirect precedent links. We refer to this procedure as the *Bottleneck Precedence Graph* (BPG) Algorithm, which we introduce in Algorithm 2.

Algorithm 2 BPG (Inputs: $\mathcal{L}, \mathcal{F}, \{\mathcal{F}_l, \forall l \in \mathcal{L}\}, \{c_l, \forall l \in \mathcal{L}\}$)

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

For every link l and at every iteration k, the BPG Algorithm keeps track of the sets of direct precedents $\mathcal{D}_l^k$, indirect precedents $\mathcal{I}_l^k$, and relays $\mathcal{R}_l^k$ (lines 2-4). Every time a link converges, these sets are updated as follows (lines 15-23):

- *Direct links* (lines 15-17). When a link l converges, it becomes a direct precedent of all the links that it shares flows with and that have not converged yet.
- *Relay links* (lines 18-20). For any two links l' and l_r that have still not converged, if they share a flow and $s_{l_r}^k < s_{l'}^k$, then flow l_r is added to the set of relays $\mathcal{R}_{l'}^k$ of link l'. Note that this set only tracks *potential* relays. It is not until the actual calculation of the indirect precedents (lines 21-23) is carried out that the algorithm confirms whether an element in this set is an actual relay leading to an indirect precedence.
- *Indirect links* (lines 21-23). When a link l converges, its indirect precedents correspond to the set of direct precedents of its relay links. Note also that its own set of direct precedents has

to be subtracted, since a link that is a direct precedent of another link, cannot be an indirect precedent of the same link.

The BPG algorithm returns a tuple of vertices (corresponding to the bottleneck links $\mathcal{B}$) and edges (corresponding to the set of direct $\mathcal{D}_l^k$ and indirect $\mathcal{I}_l^k$ precedents for every bottleneck link l, where k is the algorithm's last iteration), which provide all the necessary information to construct the bottleneck structure of the network. The last value of the iterator k is also of relevance because it corresponds to the length of the longest path in the BPG graph, as shown in the next lemma:

Lemma 2.6. *Depth of the bottleneck structure. The diameter of the BPG graph is equal to the last value of the iterator k in the BPG algorithm. We refer to this value as the depth or simply the number of levels of the bottleneck structure.*

Proof. See Appendix A.5 □

Next, we use two examples to illustrate how the BPG graph is constructed. In both of these examples, we use the network shown in Figure 4 to illustrate the process of constructing the bottleneck structure. 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 [15]. While we could have chosen any arbitrary topology, we focus on Google's B4 network to base the analysis on a real network.

Fig. 4. Google's SDN WAN B4 network as described in [15].

Example 2.7. Computation of the BPG graph. In this simple example we assume the presence of five flows $\{f_1, ..., f_5\}$ as shown in Figure 5-a. This configuration leads to the following initial state of the BPG algorithm: $\mathcal{L} = \{l_1, ..., l_{19}\}$, $\mathcal{F} = \{f_1, ..., f_5\}$; $\mathcal{F}_1 = \{f_1, f_2\}$, $\mathcal{F}_4 = \{f_1, f_4, f_5\}$, $\mathcal{F}_5 = \{f_3\}$, $\mathcal{F}_{12} = \{f_5\}$, $\mathcal{F}_{15} = \{f_5\}$, $\mathcal{F}_6 = \{f_3, f_4\}$, $\mathcal{F}_{10} = \{f_5\}$. We also assume that the links' effective capacities are such that $c_1 = 80$, $c_4 = 110$, $c_6 = 130$, $c_{15} = 20$ Gbps, while all the other links are assumed to have a capacity large enough such that they are not bottlenecks. Note that in general these effective capacities need not be equal to the physical capacity of the links. This is because it is possible to

reserve a fraction of the physical capacity to high priority traffic (traffic whose bandwidth must be guaranteed) so that the analysis of the bottleneck structure is performed without taking into account such traffic, effectively reducing the available capacity of the links.

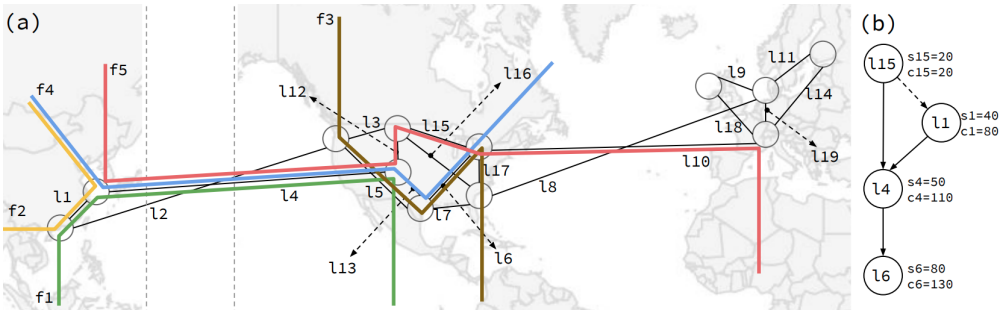


Fig. 5. Example of the B4 network with five flows used in Example 2.7.

Figure 5-b shows the BPG graph as computed by Algorithm 2. The number of levels of this bottleneck structure is four since that's the length of its longest path $l_{15} - l_1 - l_4 - l_6$ (see Lemma 2.6). Link l_{15} is a root vertex, which means it can influence the performance of all other links. It also means that in any distributed congestion control algorithm, flows bottlenecked at any other link cannot converge until flows bottlenecked at l_{15} converge. (As noted in Lemma 2.3, this is true for any pair of links for which there exists a path in the BPG graph.) Link l_6 is a leaf vertex, which implies it does not influence the performance of any other link. Note also that there exists an indirect precedence between links l_{15} and l_1. As in a direct precedent case, this also means flows bottlenecked at link l_1 cannot converge until flows bottlenecked at link l_{15} have converged. The difference here is that there exists no common flow going through both links, unlike in the case of direct precedents. Instead, execution of the BPG algorithm shows that links l_{15} and l_1 have link l_4 as a relay link ($\mathcal{R}^k_{l_1} = \{l_4\}$ when the algorithm terminates) that helps transmit the bottleneck information from one to the other.

Example 2.8. B4's bottleneck structure for full-mesh/shortest-path. In this example, we compute the bottleneck structure of the B4 network for a more realistic case considering a larger number of flows. In particular, we assume the existence of a flow between each pair of data centers—i.e., a full mesh connectivity—with each flow taking the shortest path. As a first-order approximation, we reason that the full mesh assumption is intuitively meaningful if we consider that in general, every data center may need to communicate with each other. Also, the choice of shortest paths can serve as a raw approximation of the actual routing. Note that in a production system, both of these assumptions can be relaxed by using actual historical or real-time flow and routing information of the network to compute its precise bottleneck structure. This could, for instance, be inferred using NetFlow [4], sFlow [23] or BGP-LS [9] protocols among others—see also Appendix E for details. The full-mesh/shortest-path model provides a simple but powerful base benchmark that can help measure the approximate bottleneck structure of a network by using only its fixed topological structure.

The BPG graph of the B4 network under the full-mesh/shortest-path assumption and with all link capacities set to 100 Gbps is presented in Figure 6. As shown, the root vertex corresponds to link l_8. This vertex has a region of influence equal to the full network. (Since there exists a directed path from this vertex to any other vertex in the BPG graph.) That is, variations on the effective capacity of this link are critical as they propagate to (and affect the performance of) the rest of

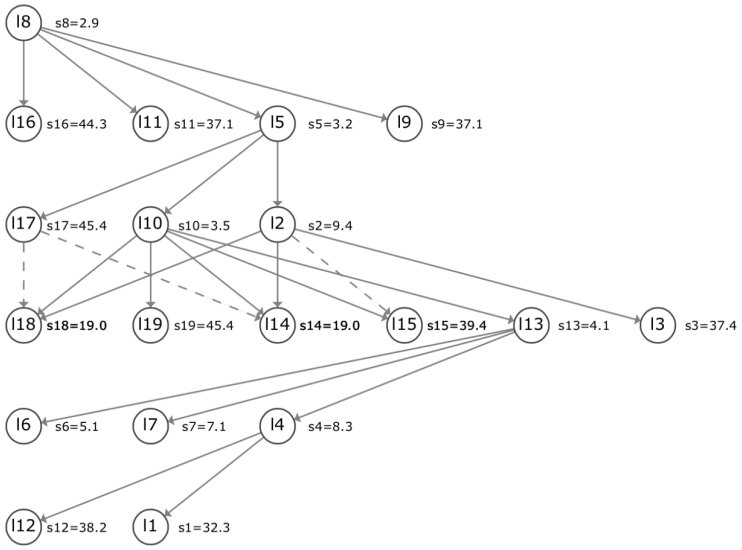


Fig. 6. Google's B4 Bottleneck structure assuming full mesh connectivity with shortest path.

the links. Looking at Figure 4, link l_8 is indeed a highly strategic link as it connects the US and the EU data centers, which are the two regions with the most data centers (Asia being the less dense region). In contrast, links l_{16}, l_{11}, l_9, l_{18}, l_{19}, l_{14}, l_{15}, l_3, l_6, l_7, l_{12}, l_1, are leaves in the bottleneck structure, which means they have no impact on the performance of any other link. This analysis reveals that when looking for high-impact bottlenecks, network operators should consider links that are located at higher levels in the BPG graph. Of interest is comparing the influence of links l_8 and l_{10}, both being the transatlantic connections between the US and EU regions. The bottleneck structure—which takes into account the structural properties of the topology, the routing and the flow information of the network—indicates that link l_8 has a higher influence than l_{10} (from Figure 6, link l_8 influences all the other links, while link l_{10} influences only a fraction of links in the BPG graph). Furthermore, the graph shows that link l_8 can affect the performance of link l_{10} (since there is a directed path from l_8 to l_{10}), but not vice versa. Similarly, on the transpacific connections side, the BPG graph shows that link l_2 has a more significant influence than link l_4 in terms of nodes they can affect, but in this case, neither of these two links can influence the performance of each other since there is no directed path connecting them. Accordingly, the kind of structural information that the BPG graph reveals helps network operators identify and organize the relevancy of each bottleneck.

Figure 6 also shows the value of the fair share for each link (positioned to the right of each vertex). It is easy to check that the values of the fair share are always monotonically increasing along a directed path in this BPG graph. This is actually a mathematical property of the bottleneck structure and is true for all BPG graphs:

Property 1. *Monotonic fair shares along a precedence path. Let $s_{l_1}, s_{l_2}, \ldots, s_{l_n}$ be the fair share values of links $l_1, l_2, \ldots, l_n$, respectively. If l_i is a direct precedent of l_{i+1}, then $s_{l_i} < s_{l_{i+1}}$, for all $1 \leq i \leq n-1$*

Proof. See Appendix A.1. □

We complete this section stating the time complexity of the BPG algorithm to compute the network's bottleneck structure. Two bounds are provided: One to compute the bottleneck links,

the direct and indirect precedents (i.e., the full bottleneck structure) and another one to compute the bottleneck links and direct precedents. While both bounds are polynomial, the latter is tighter and is useful in cases where only direct precedent relations are needed. For instance, this lower bound can be achieved when computing the regions of influence of all bottleneck links, since as shown in Lemma 2.4 there is no need to know the indirect precedents to perform such calculation.

Lemma 2.9. *Polynomial time complexity. The time complexity of computing the set of bottleneck links $\mathcal{B}$ and the sets of direct $\{\mathcal{D}_l, \forall l \in \mathcal{B}\}$ and indirect precedents $\{\mathcal{I}_l, \forall l \in \mathcal{B}\}$ is $O(|\mathcal{L}| \cdot |\mathcal{F}| + |\mathcal{L}|^3)$. The time complexity of computing the set of bottleneck links $\mathcal{B}$ and the sets of direct precedents $\{\mathcal{D}_l, \forall l \in \mathcal{B}\}$ is $O(H \cdot |\mathcal{L}|^2 + |\mathcal{L}| \cdot |\mathcal{F}|)$, where H is the maximum number of links traversed by any flow.*

Proof. See Appendix A.2. □

2.5 Minimum Convergence Time of Distributed Congestion Control Algorithms

In Section 2.1 we saw that an intuitive way to understand the bottleneck structure of a network consists of modeling its bottleneck links as nodes that are trying to compute a globally optimal fair-share value by exchanging locally available information. This dualism between the two models—bottleneck structure and distributed communication—provides a robust framework to measure the minimum convergence time attainable by any distributed congestion control algorithm. We capture this concept in the following lemma:

Lemma 2.10. *Minimum convergence time of a distributed congestion control algorithm. Let $\tau(l_i, l_j)$ be a weight assigned to each edge (l_i, l_j) of the BPG graph as follows: (1) If l_i is a direct precedent of l_j, then $\tau(l_i, l_j)$ is the time that it takes for a message to be sent from l_i to l_j; (2) If l_i is an indirect precedent of l_j, then $\tau(l_i, l_j) = max\{\tau(l_i, l_r) + \tau(l_r, l_j) \mid$ for any relay link l_r between l_i and $l_j\}$. Let $l_1 - l_2 - \ldots - l_n$ be a longest path terminating at link l_n according to these weights. Then the minimum convergence time for link l_n is $\sum_{1 \le i \le n-1} \tau(l_i, l_{i+1})$.*

Proof. See Appendix A.6. □

It is important to note that the above lower bound is not attainable in the case of TCP congestion control algorithms because, in such distributed protocols, there is no direct communication mechanism between links—since they are end-to-end protocols [29]. Instead, TCP algorithms rely on implicit feedback based on signals such as packet loss or round trip time, effectively requiring a significantly longer time to propagate from one link to another. Nevertheless, the importance of the above lemma is in the claim that convergence time increases as the depth of the bottleneck structure increases, and this general principle holds even for end-to-end protocols. In Section 3.4, we test this lemma against a TCP congestion control algorithm and empirically demonstrate that convergence time does increase as the depth of the bottleneck structure increases.

2.6 The Influence of Flows onto Bottleneck Links and Onto Each Other

While the Theory of Bottleneck Ordering helps understand the relations that exist between links and the influences they exert on each other, the BPG graph resulting from this mathematical framework does not reveal any information regarding the performance of flows. There exists, however, a simple way to transform the BPG graph into a structure that does take into account flows and helps characterize their performance. We refer to this new structure as the *flow gradient graph*, formally defined as follows:

Definition 2.11. 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 constructed using either the FastWaterFilling or the BPG algorithms, since both of these procedures compute the set of bottleneck links $\mathcal{B}$ and, for every flow in the network f, they both discover its bottleneck link l when f is entered into the converged set: $\mathcal{C}^k = \mathcal{C}^k \cup \{f, \forall f \in \mathcal{F}_l\}$ (line 17 in Algorithm 1 and line 14 in Algorithm 2). Next, we use an example to illustrate how this graph is constructed.

Example 2.12. Computation of the flow gradient graph. Consider the network in Figure 7-a consisting of 4 links and 6 flows. If we run the BPG algorithm and construct the flow gradient graph as indicated in Definition 2.11, we obtain the graph structure shown in Figure 7-b.

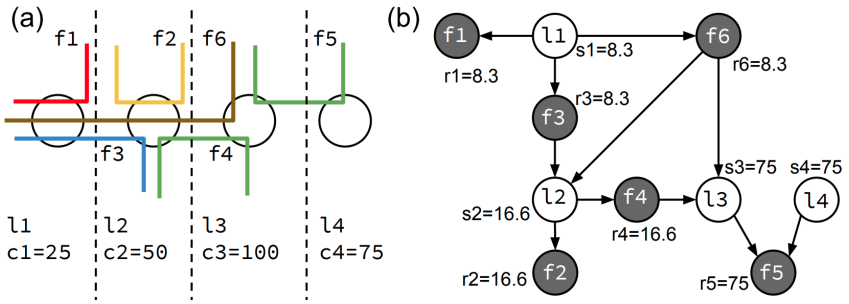


Fig. 7. Construction of the flow gradient graph.

The flow gradient graph is a useful extension of the BPG since it allows to generalize the bottleneck analysis onto flows. We can see bottlenecks and flows as two sides of the same coin so that, if the network were an economy, links and flows would correspond to the supply and the demand, respectively. This duality principle implies that all the general lemmas and properties described in this paper regarding the bottleneck structure of a network have a dual correspondence in the domain of flows. For instance, Lemma 2.4 (*bottleneck influence*) can be translated to the domain of flows as follows:

Lemma 2.13. *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 Appendix A.7. □

Using this Lemma, we can infer the following properties from the flow gradient graph in Figure 7-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 in the BPG graph; (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 3.3, we perform an experiment where we leverage the insights revealed by the flow gradient graph to identify flows that have a high impact on the performance of a network.

3 EXPERIMENTAL RESULTS

To experimentally evaluate the ability of the proposed theory to predict bottleneck and flow performance in real networks, we developed a sandbox [18] using Mininet [22] and the POX SDN controller [24]. The sandbox takes as input information related to a network's topology, routing, and traffic flows. With that, the sandbox can create the desired network configurations, including the configurations presented in Figures 4 and 7 and others introduced in this section. As part of the 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. We used iPerf [11] to generate network traffic. The sandbox also provides the ability to select a specific TCP congestion control algorithm (e.g., BBR, Cubic, or Reno) and scale to larger network configurations on multi-core machines. Since our sandbox utilizes Mininet, the implementation of these algorithms is based on real production TCP/IP code from the Linux kernel. The Linux kernel version used was 4.15.0-54. To test the various principles of the proposed theory, we developed Python scripts that process the results from the experiments run on the sandbox. In particular, for each experiment, we measured instantaneous network throughput, flow completion times, flow convergence times, and Jain's fairness indexes [14]. Essentially, the sandbox provides a flexible way to create and analyze general network architectures. We have open sourced the sandbox [18], including all the network configurations used in this paper so the Mininet research community can verify the following results.

3.1 On the Bottleneck Structure of TCP Networks

In this initial experiment, we test the hypothesis that TCP congestion-controlled networks behave as predicted by the Theory of Bottleneck Ordering. We carry out three experiments based on Google's B4 network (Figure 8), each with a different number of BPG levels (from 2 to 4 levels), and evaluate whether three well-known congestion control algorithms—BBR [2], Cubic [10] and Reno [7]—are able to recognize the bottleneck structure. We choose these three algorithms as representatives of two broad classes of algorithms. Cubic and Reno are amongst the most widely used algorithms within the class of *additive-increase/multiplicative-decrease* (AIMD) methods. BBR is a new algorithm developed by Google [2] and widely used in its infrastructure that belongs to a class known as *congestion-based* methods.

We show the corresponding BPG graphs in Figure 8 to the right of each network. The graphs include also the fair share s_i and the link capacity c_i in Mbps next to each vertex corresponding to bottleneck link $l_i \in \mathcal{B}$. The rest of the links are assumed to have a capacity large enough so that they are not bottlenecks. All flows are configured to transmit a data set of 250MB using TCP, and all links have a 2-millisecond latency. Throughout all tests, switch buffer sizes are set to 1000 packets.

In this experiment, we also measure the degree of fairness achieved by each congestion control algorithm using Jain's fairness index [14] normalized to the max-min rate allocation. (See Appendix D for a description of this index.) Since the bottleneck structure of the network is also the solution to the theoretical max-min problem, this index effectively serves as an indicator of the degree to which the congestion control algorithm under test can assign rates according to the bottleneck structure itself.

The results for BBR and Cubic are presented in Figure 9, while the results for Reno are presented in Appendix B. In this figure, flows are labeled according to their source and destination data centers. (For instance, the label $h1 - h7$ corresponds to flow f_1 in Figure 8 that goes from data center 1 to data center 7). We make the following observations:

- BBR can cleanly capture the bottleneck structure, but AIMD algorithms (Cubic and Reno) fail to do so. For instance, for the 3-level network (Figure 9-b/e), each BBR flow converges to

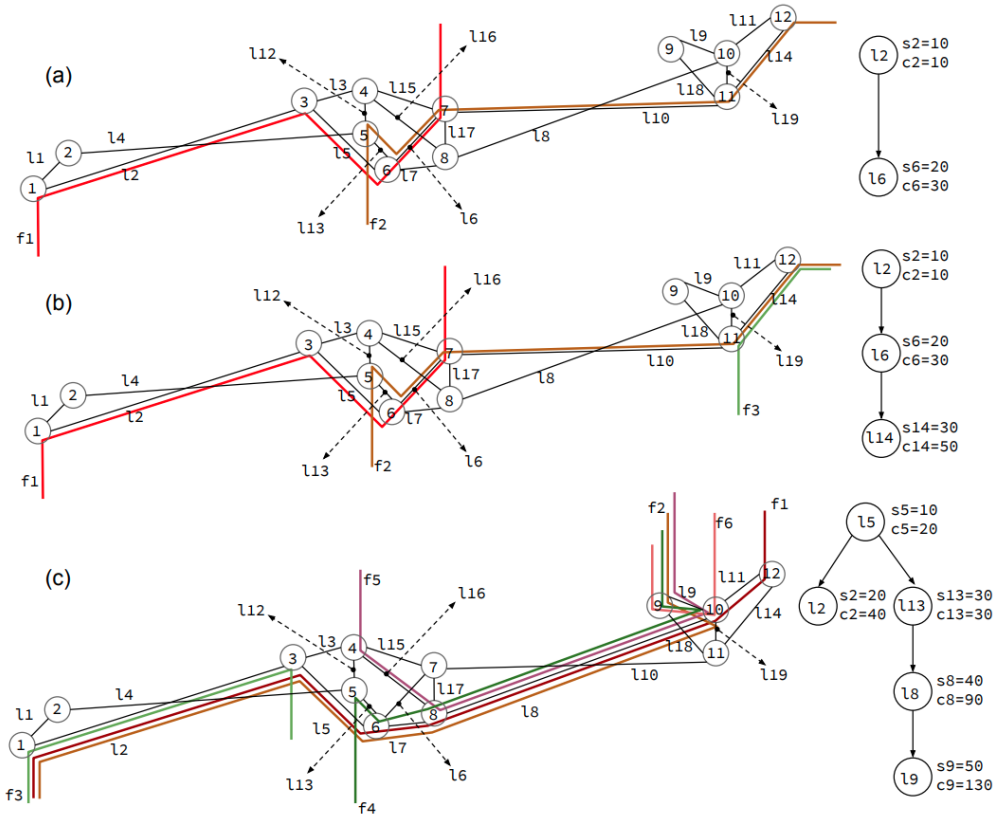
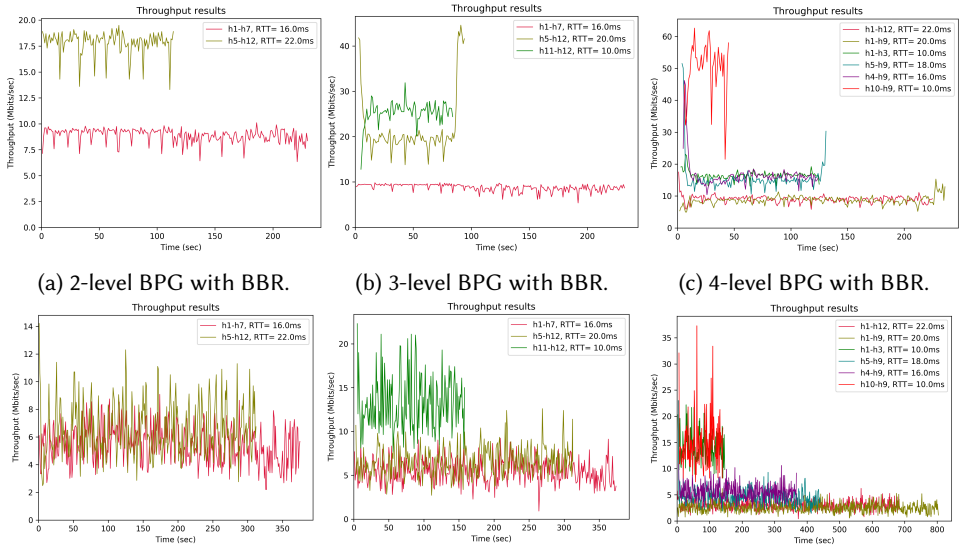


Fig. 8. Network configurations to benchmark (a) 2-level, (b) 3-level and (c) 4-level bottleneck structures.

a rate corresponding to its bottleneck level (see the BPG graph in Figure 8-b), while the two lower throughput flows in the Cubic case collapse onto a single level. We believe this is a direct effect of the *congestion-based* approach taken by BBR [2], which aims at determining the actual bottleneck capacity and effectively tries to stabilize the flow's transmission rate to its bottleneck fair share. In contrast, AIMD algorithms implement loss-based heuristics and have no notion of bottleneck's fair share. Instead they act by constantly increasing and decreasing the flow rate, and as a result, flows have a higher propensity to jump between consecutive levels in the bottleneck structure, as shown in Figure 9.

- Because BBR flows can identify their correct bottleneck levels, they don't compete against each other. AIMD flows, however, are not able to identify their bottleneck levels and end up competing against each other, leading to worse performance.
- As a result, the flow completion times for BBR are significantly lower than those of AIMD algorithms (see Table 1).
- BBR performance is relatively independent of the number of levels, as its slowest flow completion time stays between 230 and 235 seconds. For AIMD algorithms, however, performance degrades significantly when increasing the number of levels from 3 to 4, with an increase of the slowest flow completion time from 385 seconds to 810 (more than double), for Cubic.



(a) 2-level BPG with BBR. (b) 3-level BPG with BBR. (c) 4-level BPG with BBR.

(d) 2-level BPG with Cubic. (e) 3-level BPG with Cubic. (f) 4-level BPG with Cubic.

Fig. 9. BBR can identify the bottleneck structure of the network (thus achieving significantly better performance), while AIMD algorithms can't.

Table 1. Completion time of the slowest flow (seconds).

Algorithm	2-Level	3-Level	4-Level
BBR	230	235	230
Cubic	380	385	810
Reno	375	370	750

Table 2. Jain's fairness index.

Algorithm	2-Level	3-Level	4-Level
BBR	0.99	0.98	0.92
Cubic	0.94	0.96	0.72
Reno	0.93	0.96	0.73

- BBR achieves better fairness as measured by Jain's fairness index (see Table 2). Fairness index deteriorates significantly for AIMD flows when going from 3 levels to 4 levels, while BBR flows continue to maintain a good level of fairness. This result is also because BBR can better identify the bottleneck structure of the network.

3.2 On The Effect of Latency on the Bottleneck Structure of a Network

In Section 3.1, we saw that AIMD algorithms fail to identify the bottleneck structure of the test networks, and as a consequence, they perform significantly poorer than BBR. In this section, we study this phenomenon in more detail.

The first observation we make is that differences in flow RTTs can play a role in the capabilities of a congestion control algorithm to identify the bottleneck structure. For instance, in Figure 9-d

corresponding to the 2-level network with Cubic, the RTT of the flow at the highest level (flow f_2 corresponding to $h5 - h12$ in the plot's legend) is 22 milliseconds while the RTT of the flow at the lowest level (flow f_1 corresponding to $h1 - h7$ in the plot's legend) is 16 milliseconds. (These RTT values are the sum of the latencies of the links each flow goes through multiplied by two.) Due to its lower RTT, the lower level flow f_1 can capture bandwidth from the higher-level flow f_2, and this leads to the collapse of the 2 levels in the Cubic case. This is in contrast with BBR (Figure 9-a), where flows show to be resilient against these RTT variations and can stay confined at their correct bottleneck levels. Similarly, in Figure 9-e the same two flows f_1 and f_2 collapse on each other for the same reason, this time though as part of the two lower bottleneck levels in a 3-level network. As mentioned in Section 3.1, this is in contrast with BBR that, for these tested networks, can identify the bottleneck levels despite these differences in RTT.

To demonstrate the effect of RTT, we test the 3-level network (Figure 8) using Cubic and setting all link latencies to zero. While queuing delay at the switches is still non-zero, this has the effect of setting all flows to a much more similar latency, thus reducing the distortions introduced by highly different RTTs. The results are presented in Figure 10, which show that Cubic flows are now able to identify the bottleneck structure of the network. This result is in contrast with Figure 9-e, where Cubic flows were not able to identify their bottleneck rates. A relevant outcome is that due to the bottleneck distortion introduced by RTT, the performance of flows deteriorates from a flow completion time of 245 seconds in Figure 10 to 385 seconds in Figure 9-e. Note also that BBR flows were able to perform well even under the presence of link latencies. (As shown in Figures 9-a, 9-b and 9-c, BBR achieves a flow completion time of 235 seconds with non-zero link latencies, even lower than Cubic with zero link latencies.) Thus this seems to indicate that BBR is a significantly more robust congestion control algorithm against the distorting effects of RTT.

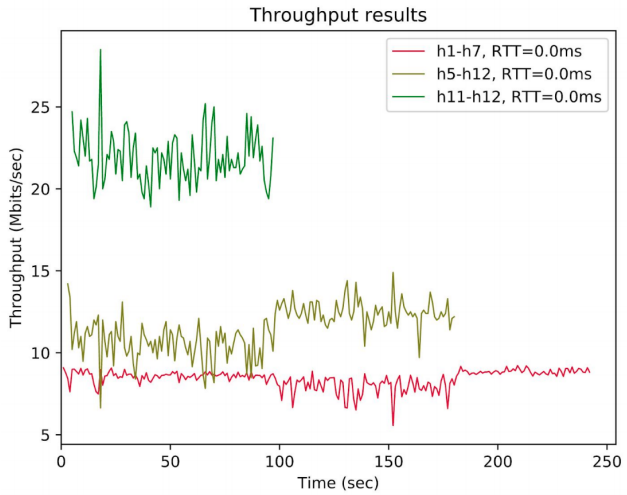


Fig. 10. By eliminating round trip times, AIMD algorithms are able to identify the bottleneck structure.

The above results show that BBR does significantly better at identifying each flow's true bottleneck, but can we still fool this congestion control algorithm to collapse its bottleneck structure? In the next experiment, we take this objective by using the 3-level network, initially setting all link latencies to zero, and progressively increasing the RTT of the flows on the upper bottleneck levels to verify whether the structure collapses. To control the RTT of each flow individually, we add a

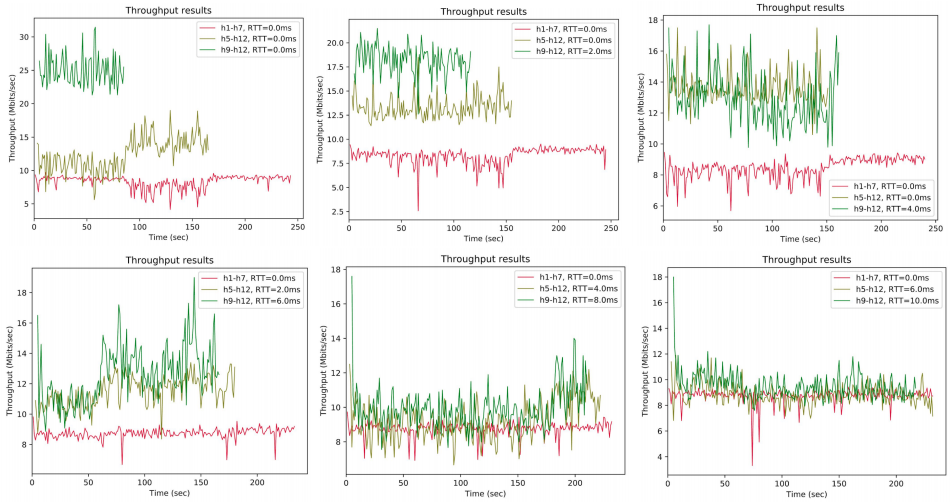


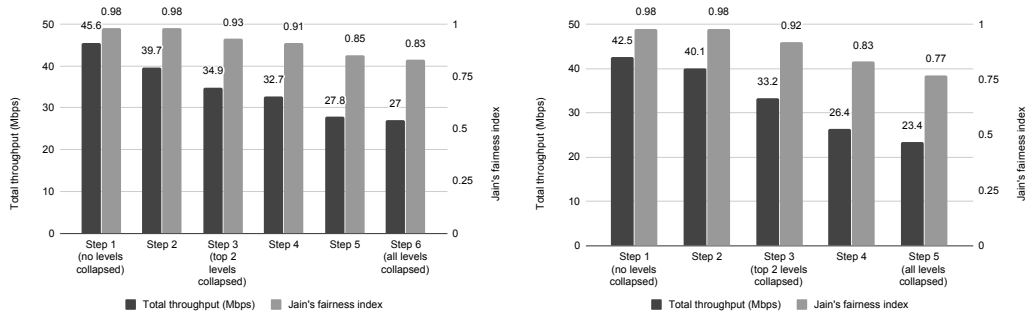
Fig. 11. The bottleneck structure of a network can be collapsed by progressively increasing the round trip time of those flows in the upper levels (BBR case).

fictitious link attached to the source of each flow with a large capacity (so it does not become a bottleneck) and a latency that we adjust to achieve a specific end-to-end RTT value for each flow. Figure 11 presents the sequence of steps that lead to the collapse of the bottleneck structure. First, flow f_3 (from data centers 9 to 12) is folded onto flow f_2 by increasing its RTT to 4 milliseconds. Then, flows f_3 and f_2 are folded onto flow f_1 by increasing their RTTs to 10 and 6 milliseconds, respectively. Thus, this experiment shows that while BBR is a more robust algorithm to the effects of RTT, its bottleneck structure can still collapse if RTTs are carefully chosen.

In Figure 12 we show the effect of collapsing the bottleneck structure on the overall performance of the network for both BBR and Reno. (Although omitted from this figure, a similar qualitative result was obtained with Cubic.) On the x axis we show the series of steps taken to collapse the bottleneck structure. Each step progressively increases the flow RTT values of the upper level flows to collapse them onto the lower neighboring flows, until the structure is fully collapsed in the last step. (So for the BBR case, the six steps shown in Figure 12-a correspond to the six graphs shown in Figure 11.) As shown, the performance of the network decreases significantly in terms of both total throughput (measured by summing up all flow rates) and fairness (measured using the Jain's fairness index). The connection between the collapse of the bottleneck structure and network performance provides a formal model to understand the reason why the performance of TCP congestion control algorithms severely deteriorates due to the distorting effects of flow RTT variations. This result also indicates the importance of designing congestion control algorithms that can robustly identify the bottleneck structure of the network against of such distortions.

3.3 On How Low-Hitter Flows Can Be High-Impact Flows

Next, we illustrate an example of how the Theory of Bottleneck Ordering can be used to support traffic engineering operations. We start with an initial network configuration, and our objective is to identify the flows that have the largest impact on the performance of the rest of the flows. Based on the information from this analysis, a network operator can choose to re-route high-impact traffic or assign it to a lower priority queue to help improve the overall performance of the network.



(a) BBR case

(b) Reno case

Fig. 12. As the bottleneck structure collapses, network performance (measured in terms of total throughput and fairness) collapses too.

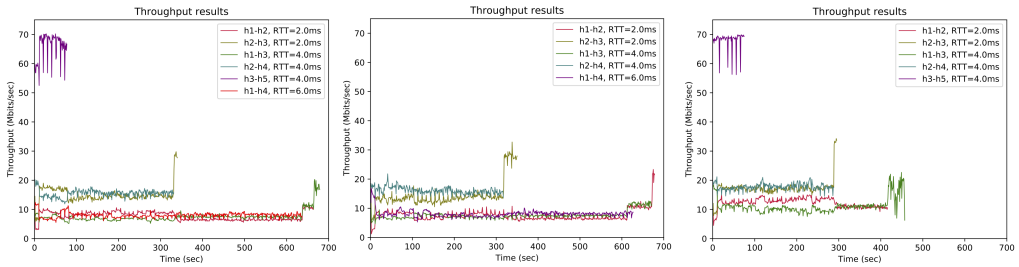
To drive this experiment, we consider the network introduced in Figure 7-a and previously used in Example 2.12. Running the BPG algorithm we obtain the following theoretical flow rate allocations: $r_1 = 8.3, r_2 = 16.6, r_3 = 8.3, r_4 = 16.6, r_5 = 75$, and $r_6 = 8.3$ Mbps. Following a traditional traffic optimization approach (e.g., [20], [8], [19], [34] , [30]), we could reach the conclusion that flow f_5 at a rate of 75 Mbps is a high-impact flow since it takes the highest bandwidth. (This flow uses more than four times the bandwidth of the second highest throughput flow.) This approach however does not take into account the bottleneck structure of the network and, as we will see, leads to a non-optimal choice.

In Figure 13-a we show the results of running BBR on the network in Figure 7. The resulting average flow throughputs are (see Table 3): $r_1 = 7.7, r_2 = 15.1, r_3 = 7.5, r_4 = 15.4, r_5 = 65.8$, and $r_6 = 8.1$ Mbps, which are relatively similar to the flow rates computed by the BPG Algorithm as provided in the previous paragraph. In Figure 13-b, we see the result of removing the heavy hitter flow f_5 from the network, showing that the impact on the rest of the flows is effectively none. (See also Table 3 for the exact values of the flows' average throughputs and completion times.) If instead we remove flow f_6 (a flow about 8 times smaller than f_5 in terms of its average throughput), we see that the rest of the flows improve their performance significantly (Figure 13-c). For instance, as shown in Table 3, the flow completion time of the slowest flow is reduced from 679 to 457 seconds. This result might seem counter-intuitive if we consider the fact that flow f_6 is among the smallest throughput flows, but it is easy to explain if we take into account the bottleneck structure of the network. In particular, from the flow gradient graph in Figure 7-b, we observe that flow f_5 is bottlenecked at links l_3 and l_4, which are leaf vertices in the bottleneck structure and thus have no impact on the performance of any other links and flows. However, flow f_6 is bottlenecked at link l_1 which sits right in the middle of the bottleneck structure, thus having a significantly higher impact. This result indicates that traditional approaches, which focus on finding the heavy hitter flows, might tend to sub-optimal choices because they do not take into account the full bottleneck structure of the network. In this paper, we reason that network operators interested in finding the high impact flows should also consider the BPG and the flow gradient graph structures to aid their optimization process.

Finally, we also note that we performed the same experiments using TCP Cubic and Reno and obtained the same behavior in both algorithms. (We include these results in Appendix C.) Overall, these experiments show that the same traffic engineering analysis based on the bottleneck structure of the network applies not only to BBR but to TCP congestion-controlled networks in general.

Table 3. As predicted by the Theory of Bottleneck Ordering, flow f_6 is a significantly higher impact flow than flow f_5.

Comp. time (secs)	f_1	f_2	f_3	f_4	f_5	f_6	Slowest
With all flows	664	340	679	331	77	636	679
Without flow f_5	678	350	671	317	—	611	678
Without flow f_6	416	295	457	288	75	—	457
Avg rate (Mbps)	f_1	f_2	f_3	f_4	f_5	f_6	Total
With all flows	7.7	15.1	7.5	15.4	65.8	8.1	119.6
Without flow f_5	7.5	14.5	7.6	16.1	—	8.3	54
Without flow f_6	12.2	17.2	11.1	17.7	68.1	—	126.3



(a) Without removing any flow. (b) Removing the heavy-hitter. (c) Removing a low-hitter.

Fig. 13. Against general best practices but as predicted by the Theory of Bottleneck Ordering, removing flow f_6 (which is a smallest low-hitter flow), maximally reduces flow completion time for the rest of flows.

3.4 Convergence Time with Increased Number of Bottleneck levels and Flow Competition

In Section 2.5, we saw that the depth of the bottleneck structure of a network is closely related to the convergence time of a distributed congestion control algorithm. We set to experimentally verify this property by using the network configuration in Figure 14-a. Its BPG graph is shown in 14-b, which corresponds to a linear (single path) graph, with a fair share for each link l_i of $s_i = 10i$, $1 \leq i \leq n$. This configuration has one BPG level per-link, and thus it corresponds to a network with maximal depth for a given number of links.

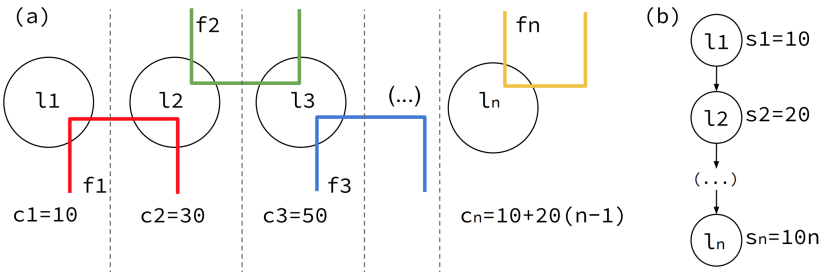


Fig. 14. Network configuration with a linear BPG graph of depth n.

To complement the convergence time analysis, we also test the impact of increasing the number of flows at each level. With this objective, we implement a flow multiplier factor as part of our sandbox

Table 4. Converge time (in seconds) increases with the number of levels and the number of level-competing flows.

	1-Level	2-Level	3-Level	4-Level
num. flows x 1	2	2	2	2
num. flows x 2	2	2	12	26
num. flows x 3	4	16	14	54
num. flows x 4	14	26	34	72

[18]. Multiplying the number of flows by a factor m means that each flow in a given network is replicated (preserving its source, destination, and route) m times. This approach effectively increases by m the number of flows competing at the same level, without modifying the bottleneck structure of the network.

Table 4 presents the results of running the described experiment varying the number of levels n and the flow multiplier factor m from 1 to 4, for a total of 16 experiments, using the BBR algorithm. The results show the behavior predicted by the theoretical framework: Convergence time increases (1) as the number of bottleneck levels increases, and (2) as the number of flows at the same level increases.

While network operators cannot influence the number of flows, they have some ability to alter the number of levels in the BPG graph. For instance, one strategy would consist of configuring routes (or even design network topologies) that tend to generate bottleneck structures of smaller depth. This strategy would help flows converge faster and lead to networks operating at higher utilization. The study and design of routes and topologies that yield *shallow* bottleneck structures, while very interesting, is left for future work.

4 ASSUMPTIONS, GENERALIZATIONS, AND PRACTICAL IMPLICATIONS

As mentioned in Section 2.2, the Theory of Bottleneck Ordering presented in this paper assumes a *steady-state optimal network regime*. Real-world networks, however, are highly dynamic systems continually transitioning from one operational regime to another due to the arrival and departure of flows. Our first observation is that, at any given time, even highly dynamic production networks have a bottleneck structure. Such a structure is not random, on the contrary, it exposes general patterns that are both qualitatively and quantitatively meaningful to understand the performance of a network. For instance, if a network shows that certain links (or certain flows) consistently tend to have a large region of influence, that reveals a specific structural pattern that network operators can use to optimize system-wide performance. Secondly, the concept of bottleneck structure provides a base mathematical framework that can open a new line of research in the field of network performance modeling and optimization. While not developed in this paper, the proposed model can be extended with both generalizations and specific features to make the model progressively more accurate. For instance, bottleneck structures can also be developed for other optimization objectives different than max-min, such as generalized weighted max-min [1] or proportional fairness [17]. Extending the theory to support proportional fairness can be of particular interest as it is believed that the Internet behaves closely to this model due to the AIMD heuristic implemented in most of TCP congestion control algorithms [17].

While in this paper we provide empirical evidence that congestion-controlled networks based on TCP qualitatively behave as predicted by their bottleneck structures, there exists a class of distributed algorithms for which the present model would theoretically achieve perfect accuracy. This class corresponds to the group of *max-min explicit rate* congestion control algorithms available

in the literature (e.g., [16, 27]). By leveraging explicit feedback from the network nodes (routers and switches), these protocols are known to converge to steady state about one or two orders of magnitude faster than TCP [16, 27, 28]. Further, as these algorithms are capable of converging to max-min, the result is that bottleneck links and flows behave precisely as forecasted by the bottleneck structure model introduced in this paper.

We reason there are at least five different areas where the Theory of Bottleneck Ordering can have a practical impact:

- *Traffic engineering*. The bottleneck structure reveals patterns in the network that can help identify critical bottlenecks and flows. For instance, in Section 3.3, we used the proposed framework to identify a low-hitter flow that—against conventional wisdom—had a high impact on the performance of the network.
- *Design of congestion control algorithms*. Between today's implicit rate-based congestion control algorithms implemented as part of TCP and the more theoretical explicit rate approaches, there is a continuum of algorithm solutions that can function according to the bottleneck structure to a greater or lesser degree. As shown in Section 3.1, those algorithms that behave closer to the bottleneck structure will have better performance according to the optimization objective captured by it. Thus, the proposed framework can also be used to help design and evaluate congestion control algorithms.
- *Network baselining*. Because the bottleneck structure provides an ideal model of network behavior, network operators can use it to create a baseline. Using such a framework, links and flows whose performance deviates from the model can be flagged for further analysis.
- *Network capacity planning*. The bottleneck structure reveals that not all bottleneck links have an equal impact on the overall performance of the network. For instance, increasing the capacity of a link that has a high region of influence can result in higher system-wide performance. Thus, network planners can use the framework described in this paper to identify optimal link upgrade strategies.
- *Artificial intelligence*. As an explicit representation of the network, the bottleneck structure could also be used to create more accurate machine learning models of modern data networks.

5 RELATED WORK

To the best of our knowledge, the proposed theoretical framework is the first to address the problem of characterizing the bottleneck structure of a network and the influences and relationships existing between bottlenecks from both formal and practical standpoints. Previous research that comes close to the proposed framework includes work presented in [16]. In that paper, the authors refer to the bottleneck relationships as dependency chains and observe that performance perturbation of bottleneck links may affect other bottleneck links. The authors, however, do not address the problem of formally identifying the hidden structure that controls such perturbations.

The majority of the previous research on bottleneck characterization has implicitly assumed that bottlenecks in a network are structured according to a flat structure, e.g., [2, 3, 13, 21, 25]. A classic result from this line of research is the well-known Mathis equation [21], which models the transmission rate of a TCP flow based on the performance characteristics of its single bottleneck link. This approach, however, does not take into account the distributed nature of communication networks. In contrast, the proposed approach reveals the bottleneck structure of a network, which allows to model flow performance by taking into account the network's topological, routing, and flow properties.

In this work, we base the bottleneck analysis on the max-min rate assignment. An algorithm to compute the max-min rate allocation of a network was initially introduced in [1] and later

improved by [27]. Our algorithm to construct the bottleneck structure is based on an extension of the algorithm proposed in [27], although in that paper the authors use the algorithm to design a distributed algorithm, without addressing the general bottleneck structure of networks.

6 CONCLUSIONS

Congestion control algorithms have been one of the most widely researched subjects in the field of computer communications since the Internet collapsed in 1986 due to the lack of them. Starting from the first TCP congestion control algorithm implemented in 1987, key research has focused on characterizing the bottleneck to help maximize the performance of flows individually while achieving fairness collectively. Such single-bottleneck oriented approach, however, has left a gap in attempting to understand the global structure of the problem. In this paper, we address this gap by introducing the Theory of Bottleneck Ordering. The theoretical framework leads to the BPG algorithm, a procedure that reveals the bottleneck structure of a network in the form of a graph—the BPG graph. This graph provides vital insights to the understanding of the collective bottleneck behavior in a distributed network, among others: (1) The implicit relationships that exist between bottlenecks; (2) the bounded regions in the network that links and flows can influence; and (3) the convergence properties of the congestion control algorithm run by the network. Our experimental results show that: (1) Real networks do have a bottleneck structure that may follow the BPG graph closely; (2) RTT variations can lead to imperfections (levels that fold) in the bottleneck structure, which can severely affect performance; (3) congestion-based algorithms such as BBR are able to infer the bottleneck structure significantly better than loss-based/AIMD algorithms such as Cubic and Reno, which leads them to achieve superior performance in both throughput and fairness. The theory also reveals intriguing results, including the fact that low-hitter flows can have a higher impact on a network than heavy-hitter flows, or the notion that routing and topology can play a role in the design of shallower bottleneck structures that would improve the convergence time of congestion control algorithms.

Current and future work centers around three lines of effort. First, in this paper, we have primarily focused on the problem of constructing the bottleneck structure, and the resulting analysis has been qualitative. In forthcoming work, we will show that the flow gradient graph structure (only briefly introduced in this paper) reveals an algorithm to efficiently compute the flow and bottleneck gradients. The theoretical development of the flow gradient graph leads to a formal quantitative framework, providing a quantifiable methodology to measure the impact that bottlenecks and flows exert on each other. Secondly, the concept of bottleneck structure can be generalized to include other optimal objectives beyond max-min, such as weighted max-min or proportional fairness, among others. Lastly, while in this paper we provide empirical evidence of the existence of a bottleneck structure in TCP networks using Mininet, more experiments should be performed on real-world production networks. We are currently undertaking such tests in three different networks: the US nationwide ESnet 100Gbps Testbed Network [6], the Cosmos Wireless network in New York City [5] and the SCinet terabit network implemented as part of the Supercomputing industry event [31].

REFERENCES

[1] Dimitri P. Bertsekas and Robert G. Gallager. 1992. *Data Networks*. Vol. 2. Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632.

[2] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2016. BBR: Congestion-Based Congestion Control. *ACM Queue* 14, 5, Article 50 (October 2016), 34 pages. https://doi.org/10.1145/3012426.3022184

[3] Dah-Ming W. Chiu and Raj Jain. 1989. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN systems* 17, 1 (June 1989), 1–14. https://doi.org/10.1016/

0169-7552(89)90019-6

[4] Benoit Claise, Ganesh Sadasivan, Vamsi Valluri, and Martin Djernaes. 2004. NetFlow Specifications, Cisco Systems. (2004). Retrieved October 24, 2019 from https://www.ietf.org/rfc/rfc3954.txt

[5] COSMOS Lab. 2019. The Cosmos Testbed. (2019). Retrieved October 24, 2019 from https://cosmos-lab.org

[6] ESnet. 2019. ESnet Energy Sciences Network. (2019). Retrieved October 24, 2019 from http://es.net/network-r-and-d/experimental-network-testbeds/test-circuit-service/

[7] Kevin Fall and Sally Floyd. 1996. Simulation-based Comparisons of Tahoe, Reno and SACK TCP. *SIGCOMM Computer Communication Review* 26, 3 (July 1996), 5–21. https://doi.org/10.1145/235160.235162

[8] Tiago Fioreze, Lisandro Granville, Ramin Sadre, and Aiko Pras. 2009. A Statistical Analysis of Network Parameters for the Self-Management of Lambda-Connections. In *Scalability of Networks and Services*. Springer Berlin Heidelberg, Berlin, Heidelberg, 15–27. https://doi.org/10.1007/978-3-642-02627-0_2

[9] Hannes Gredler, Jan Medved, Stefano Previdi, Adrian Farrel, and Saikat Ray. 2016. BGP-LS Protocol Specification. (2016). Retrieved October 24, 2019 from https://tools.ietf.org/html/rfc7752

[10] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS operating systems review* 42, 5 (July 2008), 64–74. https://doi.org/10.1145/1400097.1400105

[11] Iperf.fr. 2019. iPerf - The ultimate speed test tool for TCP, UDP and SCTP. (2019). Retrieved October 24, 2019 from https://iperf.fr/

[12] M. Schoffstall J. Davin J. Case, M. Fedor. 1990. A Simple Network Management Protocol (SNMP). (1990). Retrieved October 24, 2019 from https://tools.ietf.org/html/rfc1157

[13] Van Jacobson. 1988. Congestion Avoidance and Control. *SIGCOMM computer communication review* 18, 4 (August 1988), 314–329. https://doi.org/10.1145/52325.52356

[14] Raj Jain, Dah-Ming W. Chiu, and William R. Hawe. 1998. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. *CoRR* cs.NI/9809099 (1998), 38. http://arxiv.org/abs/cs.NI/9809099

[15] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jon Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally-Deployed Software Defined WAN. *SIGCOMM Computer Communication Review* 43, 4 (August 2013), 3–14. https://doi.org/10.1145/2534169.2486019

[16] Lavanya Jose, Lisa Yan, Mohammad Alizadeh, George Varghese, Nick McKeown, and Sachin Katti. 2015. High Speed Networks Need Proactive Congestion Control. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks (HotNets-XIV)*. ACM, New York, NY, USA, Article 14, 7 pages. https://doi.org/10.1145/2834050.2834096

[17] Frank P. Kelly, Aman K. Maulloo, and David K. H. Tan. 1998. Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research society* 49, 3 (01 March 1998), 237–252. https://doi.org/10.1057/palgrave.jors.2600523

[18] Reservoir Labs. 2019. G2-Mininet Sandbox: Mininet extensions to support the analysis of the bottleneck structure of networks. (2019). Retrieved October 24, 2019 from https://github.com/reservoirlabs/g2-mininet

[19] Kun-Chan Lan and John Heidemann. 2006. A Measurement Study of Correlations of Internet Flow Characteristics. *Computer Networks* 50, 1 (2006), 46–62. https://doi.org/10.1016/j.comnet.2005.02.008

[20] Yi Lu, Mei Wang, Balaji Prabhakar, and Flavio Bonomi. 2007. ElephantTrap: A Low Cost Device for Identifying Large Flows. In *15th Annual IEEE Symposium on High-Performance Interconnects (HOTI 2007)*. IEEE, 99–108. https://doi.org/10.1109/HOTI.2007.13

[21] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. 1997. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm. *SIGCOMM Comput. Commun. Rev.* 27, 3 (July 1997), 67–82. https://doi.org/10.1145/263932.264023

[22] Mininet. 2019. Mininet: An Instant Virtual Network on your Laptop (or other PC). (2019). Retrieved October 24, 2019 from http://mininet.org/

[23] Peter Phaal, Sonia Panchen, and Neil McKee. 2001. sFlow Specifications, InMon Corporation. *IETF RFC 3176* (2001).

[24] NOX Repo POX. 2019. The POX Network Software Platform. (2019). Retrieved October 24, 2019 from https://noxrepo.github.io/pox-doc/html/

[25] Konstantinos Psounis, Arpita Ghosh, Balaji Prabhakar, and Gang Wang. 2005. SIFT: A Simple Algorithm for Tracking Elephant Flows, and Taking Advantage of Power Laws. In *43rd Allerton Conference on Communication, Control and Computing*.

[26] Jordi Ros-Giralt. 2003. A Theory of Lexicographic Optimization for Computer Networks. University of California, Irvine, Irvine, California. https://doi.org/10.13140/RG.2.1.2188.1368 AAI3101616.

[27] Jordi Ros-Giralt and Wei K. Tsai. 2001. A Theory of Convergence Order of Maxmin Rate Allocation and an Optimal Protocol. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, Vol. 2. IEEE, 717–726 vol.2. https://doi.org/10.1109/INFCOM.2001.916260

[28] Jordi Ros-Giralt and Wei K. Tsai. 2010. A Lexicographic Optimization Framework to the Flow Control Problem. *IEEE Transactions on Information Theory* 56, 6 (June 2010), 2875–2886. https://doi.org/10.1109/TIT.2010.2046227

[29] Jerome H. Saltzer, David P. Reed, and David D. Clark. 1984. End-to-end Arguments in System Design. *ACM Transactions on Computer Systems (TOCS)* 2, 4 (November 1984), 277–288. https://doi.org/10.1145/357401.357402

[30] Shriram Sarvotham, Rudolf Riedi, and Richard Baraniuk. 2001. Connection-level Analysis and Modeling of Network Traffic. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement (IMW '01)*. ACM, New York, NY, USA, 99–103. https://doi.org/10.1145/505202.505215

[31] SCinet. 2019. The SCinet Network at Supercomputing. (2019). Retrieved October 24, 2019 from https://sc19.supercomputing.org/scinet/all-about-scinet/

[32] Suricata. 2019. Suricata: Open Source IDS / IPS / NSM engine. (2019). Retrieved October 24, 2019 from https://suricata-ids.org/

[33] Zeek. 2019. The Zeek Network Security Monitor. (2019). Retrieved October 24, 2019 from https://www.zeek.org/

[34] Yu Zhang, Binxing Fang, and Yongzheng Zhang. 2010. Identifying High-Rate Flows based on Bayesian Single Sampling. In *2010 2nd International Conference on Computer Engineering and Technology*, Vol. 1. IEEE, V1–370–V1–374. https://doi.org/10.1109/ICCET.2010.5486097

A MATHEMATICAL PROOFS

Property 2. *Monotonic fair shares. Let s_l^k be the fair share of link l at iteration k of the BPG algorithm. Then $s_l^k \leq s_l^{k+1}$.*

Proof. This result was proven by Ros-Giralt in Property 3.6 of [26]. □

A.1 Property 1: Monotonic fair shares along a precedence path

Let $s_{l_1}, s_{l_2}, ..., s_{l_n}$ be the fair share values of links $l_1, l_2, ..., l_n$, respectively. If l_i is a direct precedent of l_{i+1}, then $s_{l_i} < s_{l_{i+1}}$, for all $1 \leq i \leq n-1$

Proof. By induction, it is sufficient to prove that the lemma holds for $n = 2$. Let k_1 and k_2 be the value of k at the iteration when links l_1 and l_2 are removed from the set of unresolved links (line 13 in the BPG algorithm), respectively. This implies $s_{l_1} = s_{l_1}^{k_1}$ and $s_{l_2} = s_{l_2}^{k_2}$, since the fair share of these links is no longer modified after they are removed from the set of unresolved links. Now assume l_1 is a direct precedent of l_2. It must be that $k_2 > k_1$, since from lines 15 and 16 of the BPG algorithm, at any arbitrary iteration k, a link l is only added to the set of direct precedents of another link l' ($\mathcal{D}_{l'}^k = \mathcal{D}_{l'}^k \cup l$) if link l has been resolved ($l \notin \mathcal{L}^k$) and link l' has not been resolved ($l' \in \mathcal{L}^k$). At iteration k_1 when link l_1 is resolved, we have $s_{l_1}^{k_1} < s_{l_2}^{k_1}$, since link l_1 and l_2 share at least one flow and $s_{l_1}^{k_1} = u_{l_1}^{k_1}$. Using Property 2 we have that $k_2 > k_1$ implies $s_{l_2}^{k_1} \leq s_{l_2}^{k_2}$. Thus, we have $s_{l_1} = s_{l_1}^{k_1} < s_{l_2}^{k_1} \leq s_{l_2}^{k_2} = s_{l_2}$.

□

A.2 Lemma 2.9: Polynomial time complexity

The time complexity of computing the set of bottleneck links $\mathcal{B}$ and the sets of direct $\{\mathcal{D}_l, \forall l \in \mathcal{B}\}$ and indirect precedents $\{\mathcal{I}_l, \forall l \in \mathcal{B}\}$ is $O(|\mathcal{L}| \cdot |\mathcal{F}| + |\mathcal{L}|^3)$. The time complexity of computing the set of bottleneck links $\mathcal{B}$ and the sets of direct precedents $\{\mathcal{D}_l, \forall l \in \mathcal{B}\}$ is $O(H \cdot |\mathcal{L}|^2 + |\mathcal{L}| \cdot |\mathcal{F}|)$, where H is the maximum number of links traversed by any flow.

Proof. We start by noting that since a link is removed from $\mathcal{L}^k$ at line 13 of the BPG algorithm, then lines 9, 10, 12, 13, 14, 15, 18 and 21 cannot be executed more than $|\mathcal{L}|$ times. We then analyze the complexity of these lines organizing them in three groups:

- Lines 9 and 10. The complexity of invoking once each of these two lines is $O(|\mathcal{L}|)$ and $O(|\mathcal{L}| \cdot H)$, respectively, where H is the maximum number of links traversed by any flow. Thus, the aggregated total execution time of these lines in one execution of the algorithm is $O(|\mathcal{L}| \cdot (|\mathcal{L}| + |\mathcal{L}| \cdot H)) = O(H \cdot |\mathcal{L}|^2)$.
- Lines 12, 13 and 14. The complexity of invoking once each of these three lines is $O(|\mathcal{F}|)$, $O(1)$ and $O(|\mathcal{F}|)$, respectively. Thus, the aggregated total execution time of these lines in one execution of the algorithm is $O(|\mathcal{L}| \cdot |\mathcal{F}|)$.
- Lines 15, 18 and 21. The complexity of invoking once each of these three *for* loops is $O(|\mathcal{L}|)$, $O(|\mathcal{L}|^2)$ and $O(|\mathcal{L}|)$, respectively. Thus, the aggregated total execution time of these lines in one execution of the algorithm is $O(|\mathcal{L}|^3)$.

Adding up the above three values, we obtain $O(H \cdot |\mathcal{L}|^2 + |\mathcal{L}| \cdot |\mathcal{F}| + |\mathcal{L}|^3)$. But since $H < |\mathcal{L}|$, we have that the time complexity of the BPG algorithm is $O(|\mathcal{L}| \cdot |\mathcal{F}| + |\mathcal{L}|^3)$. Finally, it is also easy to see that if we omit the calculation of indirect precedents, then the *for* loops in lines 18 and 21 are not executed and the time complexity is reduced to $O(H \cdot |\mathcal{L}|^2 + |\mathcal{L}| \cdot |\mathcal{F}|)$.

□

A.3 Lemma 2.3: Bottleneck convergence order

A link l is removed from the set of unresolved links $\mathcal{L}^k$ at iteration k in the BPG algorithm, $\mathcal{L}^k = \mathcal{L}^k \setminus \{l\}$, if and only if all of its direct and indirect precedent links have already been removed from this set at iteration $k-1$.

Proof. (Note that in what follows, the sentences "a link has converged" and "a link has been removed from the set of unresolved links" are considered equivalent.) We start with the sufficient condition. Assume that at an arbitrary iteration k all direct and indirect precedents of a link l_1 have converged. We will also assume that link l_1 does not converge at iteration k and arrive at a contradiction. It must be that $s_{l_1}^k \neq u_{l_1}^k$, otherwise from lines 11 and 13 of the BPG algorithm link l_1 would converge at iteration k. This implies the existence of at least one link l_r that shares a flow with l_1, $\mathcal{F}_{l_1} \cap \mathcal{F}_{l_r} \neq \{\emptyset\}$, such that at iteration k it has not converged yet, $l_r \in \mathcal{L}^k$, and $s_{l_r}^k < s_{l_1}^k$. Note from lines 18 and 19 such link is added to the relay set of link l_1, $l_r \in \mathcal{R}_{l_1}^k$. Assume link l_1 converges at iteration k_1, then since all of its direct precedents converged at iteration k, we have $s_{l_1}^{k_1} = s_{l_1}^k$. This also means that $s_{l_r}^{k_1} > s_{l_r}^k$, which necessarily implies the existence of at least one link l_2 that converges at an iteration k_2 such that $k < k_2 < k_1$ and $\mathcal{F}_{l_r} \cap \mathcal{F}_{l_2} \neq \{\emptyset\}$. Such link l_2 is a direct precedent of l_r and, since l_r is in the relay set of link l_1, at line 22 of the BPG algorithm, link l_2 must become an indirect precedent of l_1. Since link l_2 converges at iteration k_2 and $k_2 > k$, we arrive at a contradiction.

To address the necessary condition, suppose that a link l_1 converges at iteration k and that another link l_2 that has not converged yet at the same iteration is either a direct or indirect precedent of link l_1. By construction of the BPG algorithm, however, this is not possible since once link l_1 converges at iteration k, it is removed from the set of unresolved links $\mathcal{L}^k$, thus no further direct or indirect links can be added to it.

□

A.4 Lemma 2.4: 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 $\{l_1, l_2, ..., l_n\}$ 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. By induction, it is enough to show the case $n = 2$. We start with the sufficient condition. Assume that link l_1 is a direct precedent of link l_2, then from lines 15 and 16 of the BPG algorithm there must exist a flow f bottlenecked at link l_1 that traverses both links l_1 and l_2, $f \in \mathcal{F}_{l_1} \cap \mathcal{F}_{l_2}$. Note that any infinitesimal change on the capacity of link l_1 changes the rate of such flow, r_f, since this flow is bottlenecked at the same link (lines 9 and 12 in the BPG algorithm). Note also that such variation in the value of r_f propagates to link l_2 inducing a change in its fair share (line 9), which implies $\partial s_{l_2}/\partial c_{l_1} \neq 0$.

To address the necessary condition, note first that the performance of a link l is uniquely determined by the fair share equation (line 9), since this value determines the rate of all flows bottlenecked at link l:

$$s_l^k = (c_l - \sum_{\forall f \in C^k \cap \mathcal{F}_l} r_f)/|\mathcal{F}_l \setminus C^k|, \forall l \in \mathcal{L}^k \tag{1}$$

This equation depends on internal link parameters (such as its capacity c_l and the set of flows going through it $\mathcal{F}_l$) as well as external parameter (such as the set of flows bottlenecked at some other link and their rate values $\{r_f \mid \forall f \in C^k \cap \mathcal{F}_l\}$). Thus, s_l can only be externally influenced by

changing these rates. From lines 15 and 16, these rates correspond to flows that are bottlenecked at links that are direct precedents of link l. Thus, the fair share of a link can only change if the rate of one or more of its flows bottlenecked at one of its direct precedent links changes. Since the rates of these flows are equal to the fair share of the direct precedent links, this implies that in order to induce a change in the fair share of s_l, it is necessary to change the capacity of one or more of its direct precedent links.

□

A.5 Lemma 2.6: Bottleneck structure depth

The diameter of the BPG graph—which we will also refer as the depth or the number of levels of the bottleneck structure—is equal to the last value of the iterator k in the BPG algorithm.

Proof. Since the diameter of the BPG graph corresponds to the maximum length of any of its paths and since at every iteration the algorithm adds one more vertex to any longest path in the graph, the value of k at the end of the algorithm must be equal to the size of any longest path. □

A.6 Lemma 2.10: Minimum convergence time of a distributed congestion control algorithm

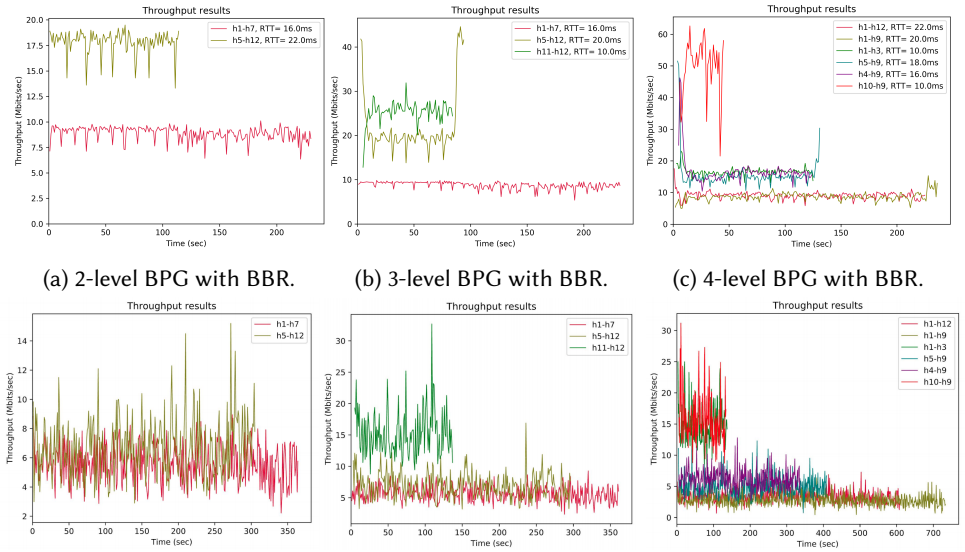
Let $\tau(l_i, l_j)$ be a weight assigned to each edge (l_i, l_j) of the BPG graph as follows: (1) If l_i is a direct precedent of l_j, then $\tau(l_i, l_j)$ is the time that it takes for a message to be sent from l_i to l_j; (2) If l_i is an indirect precedent of l_j, then $\tau(l_i, l_j) = max\{\tau(l_i, l_r)+\tau(l_r, l_j) \mid$ for any relay link l_r between l_i and $l_j\}$. Let $l_1 - l_2 - \ldots - l_n$ be a longest path terminating at link l_n according to these weights. Then the minimum convergence time for link l_n is $\sum_{1 \leq i \leq n-1} \tau(l_i, l_{i+1})$.

Proof. From Lemma 2.3, we know that a link l cannot resolve its fair share until all of its direct and indirect precedents have resolved their own fair share. Furthermore, we know that if all the direct and indirect precedents of a link l have been resolved at iteration k, then link l can converge immediately after at iteration $k + 1$. To derive the current lemma, we only need to take into account the time that it takes to communicate a message from the direct and indirect links of link l to link l itself. Because direct precedent links share a flow, communication can propagate directly. For the case of indirect precedence, communication has to go through the relay link. In particular, we need to select the relay link that imposes the longest distance, as that's the longest time it can take to propagate the state between a link and its indirect precedent. The $\tau()$ function introduced in the lemma captures the value of these communication times for the two possible relations between bottleneck links. □

A.7 Lemma 2.13: 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 $\{l_1, l_2, \ldots, l_n\}$ 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. Let f and f' be two arbitrary flows and assume that flow f' is bottlenecked at link l'. From lines 9 and 12 of the BPG algorithm, the rate of a flow corresponds to the fair share of its bottleneck. Thus, flow f can only influence flow f' if it can affect its bottleneck's fair share, $s_{l'}$. Assume that we apply an infinitesimal change to flow f's rate r_f. This could be achieved by applying a traffic shaper that slightly decreases its rate (negative infinitesimal) or by imposing a minimum rate constraint that assigns a rate slightly higher than the fair share of its bottleneck link s_l (positive infinitesimal). For any link l traversed by flow f, such small perturbation will lead to an infinitesimal change in its fair share s_l. Now from Lemma 2.4 we know that a link l



(a) 2-level BPG with BBR. (b) 3-level BPG with BBR. (c) 4-level BPG with BBR.

(d) 2-level BPG with Reno. (e) 3-level BPG with Reno. (f) 4-level BPG with Reno.

Fig. A1. Comparison of BBR and Reno's bottleneck structures: Unlike BBR, Reno is not able to identify the bottleneck structure of the network.

can only influence another link l' if there exists a directed path from l to l' in the BPG graph. Furthermore, Lemma 2.4 is a necessary and sufficient condition. Thus, both the necessary and sufficient conditions of this lemma must hold too if we set $l = l_1$ and $l' = l_n$. □

B RESULTS OF THE BOTTLENECK STRUCTURE WITH TCP RENO

Section 3.1 demonstrated that BBR generally performs better in capturing a given network's bottleneck structure than AIMD-based protocols such as Cubic and Reno. While the results of BBR and Cubic are presented in Figure 9, Figure A1 presents the same results for BBR and Reno.

C RESULTS OF THE LOW-HITTER FLOW EXPERIMENT FOR TCP CUBIC AND RENO

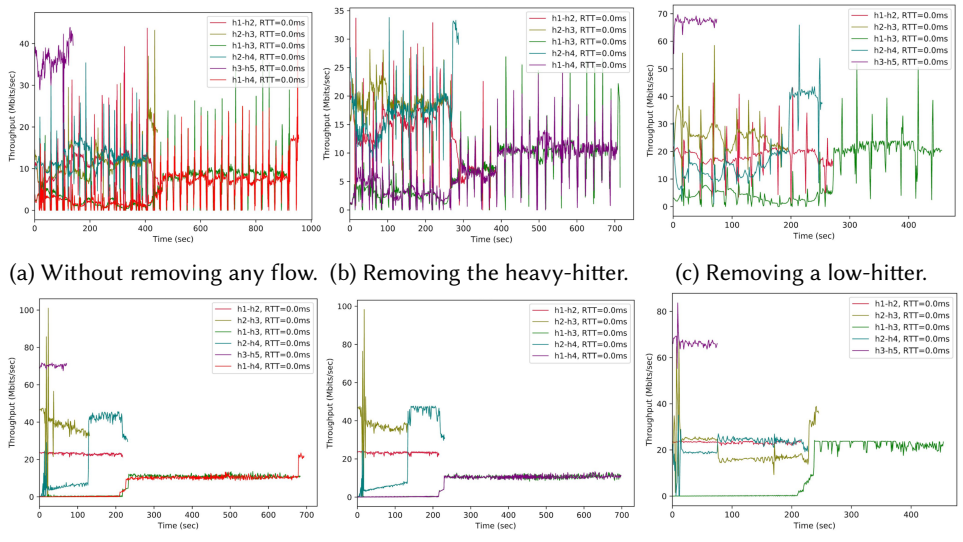
The plots in Figure A2 correspond to the experiment described in Section 3.3 using TCP Cubic and Reno instead of TCP BBR. Note that the same qualitative results are obtained in Cubic and Reno as with BBR—the biggest reduction in flow completion time is achieved when the low-hitter flow f_6 is removed.

D JAIN'S FAIRNESS INDEX

Jain's index [14] is a metric that rates the fairness of a set of values $x_1, x_2, ..., x_n$ according to the following equation:

$$\mathcal{J}(x_1, x_2, ..., x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} x_i^2} = \frac{\overline{\mathbf{x}}^2}{\overline{\mathbf{x}^2}}$$

The index value ranges from $\frac{1}{n}$ (worst case) to 1 (best case). As suggested in [14], for multi-link networks the value x_i must be normalized to an optimal fairness allocation. Throughout this paper, we normalize x_i as the ratio f_i/O_i, where f_i is the rate of flow f_i achieved through the experiments and O_i is its expected max-min fair throughput.



(a) Without removing any flow. (b) Removing the heavy-hitter. (c) Removing a low-hitter.

(d) Without removing any flow. (e) Removing the heavy-hitter. (f) Removing a low-hitter.

Fig. A2. The low-hitter flow is also a high-impact flow when using TCP Cubic and Reno. (a), (b), (c) are for Cubic; (d), (e), (f) are for Reno.

E NOTES ON USING THE FRAMEWORK IN PRODUCTION NETWORKS

In this section, we present practical considerations to enable the integration of the proposed Theory of Bottleneck Ordering into modern data networks. To construct the bottleneck structure, the BPG algorithm requires three inputs:

- *Flow information.* The set of flows: $\{f_i \mid \forall l_i \in \mathcal{L}\}$. This input requires knowing the flows' IP tuples (source and destination IP addresses and port numbers), which can be obtained from network monitoring protocols such as NetFlow [4] and sFlow [23], or from security monitoring tools such as Zeek [33] and Suricata [32]. Such information is commonly available in modern high-speed data networks, both for real-time consumption and for offline analysis.
- *Routing information.* The set of links traversed by each flow or, equivalently, the set of flows traversing each link: $\{\mathcal{F}_l \mid \forall l \in \mathcal{L}\}$. This information can be inferred from routing table information provided by protocols such as BPG-LS [9]. Since the flow information input provides the source and destination IP addresses of each flow, the routing information allows to reconstruct the set of links traversed by each flow.
- *Network topology.* The capacity of each link: $\{c_i \mid \forall l_i \in \mathcal{L}\}$. Such information can be obtained from software defined networking (SDN) components or from traditional protocols like SNMP [12]

If the network runs NetFlow or sFlow in all of its routers (as is often the case in advanced high-speed networks), then the bottleneck structure can be reconstructed by just reading flow information from these protocols together with the link capacity values. In such cases, routing information is not needed explicitly, as both of these protocols provide in each packet record the IP address of the router where the packet sample is taken and the IP address of its *next hop* router. This information allows to reconstruct flow path information.

Received August 2019; revised September 2019; accepted October 2019