Workgroup: ALTO
Internet-Draft:
draft-giraltyellamraju-alto-bsg-
requirements-03
Published: 23 September 2022
Intended Status: Informational
Expires: 27 March 2023
Authors: J. Ros-Giralt S. Yellamraju Q. Wu
Qualcomm Qualcomm Huawei
L.M. Contreras R. Yang K. Gao
Telefonica Yale University Sichuan University

Supporting Bottleneck Structure Graphs in ALTO: Use Cases and Requirements

Abstract

This document proposes an extension to the base Application-Layer Traffic Optimization (ALTO) protocol to support bottleneck structures as an efficient representation of the state of a network. Bottleneck structures are efficient computational graphs that allow network operators and application service providers to optimize application performance in a variety of communication problems including routing, flow control, flow scheduling, bandwidth prediction, and network slicing, among others. This document introduces a new abstraction called Bottleneck Structure Graph (BSG) and the necessary requirements to integrate it into the ALTO standard.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://giralt.github.io/draft-ietf-alto-gradient-graph/draft-giraltyellamraju-alto-bsg-requirements.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-giraltyellamraju-alto-bsg-requirements/.

Discussion of this document takes place on the WG Working Group mailing list (mailto:alto@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/alto/.

Source for this draft and an issue tracker can be found at https://github.com/giralt/draft-ietf-alto-gradient-graph.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 27 March 2023.

Copyright Notice

Table of Contents

1. Introduction

Bottleneck structures have been recently introduced in [G2-SIGCOMM] and [G2-SIGMETRICS] as efficient computational graphs that embed information about the topology, routing and flow information of a network. These computational graphs allow network operators and application service providers to compute network derivatives that can be used to make traffic optimization decisions. For instance, using the bottleneck structure of a network, a real-time communication (RTC) application can efficiently infer the multi-hop end-to-end available bandwidth, and use that information to tune the encoder's transmission rate and optimize the user's Quality of Experience (QoE). Bottleneck structures can be used by the application to address a wide variety of communication optimization problems, including routing, flow control, flow scheduling, bandwidth prediction, and network slicing, among others.

This document introduces a new abstraction called Bottleneck Structure Graph (BSG) and the necessary requirements to integrate it into the existing ALTO services (Network Map, Cost Map, Entity Property Map and Endpoint Cost Map) exposing the properties of the bottleneck structure to help optimize application performance. Use cases are also introduced to motivate the relevancy of bottleneck structures in the context of the ALTO standard and support the description of the integration requirements.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in

BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Brief Introduction to Bottleneck Structures

[G2-SIGMETRICS] and [G2-SIGCOMM] introduce a new mathematical framework to optimize network performance called the Quantitative Theory of Bottleneck Structures (QTBS). The core building block of QTBS is a computational graph called *bottleneck structure*, which allows to qualify and quantify the forces of interactions that flows and bottleneck links exert on each other. QTBS builds the bottleneck structure by assuming that flows in a network aim at maximizing throughput while ensuring fairness. This general principle holds for all the relevant congestion control algorithms used in IP networks (e.g., TCP Cubic, BBR, QUIC, etc.).

3.1. Example of Bottleneck Structure

Consider as an example the following network configuration:

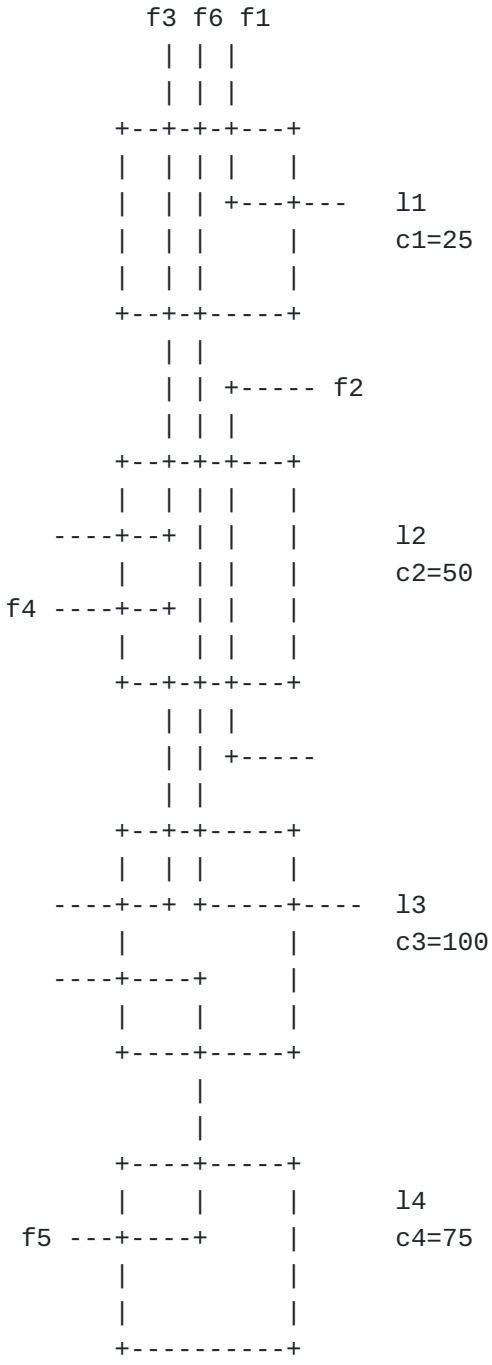

Figure 1: Network configuration example.

Each link li is represented by a squared box and has a capacity ci. For instance, link l1 is represented by the top most squared box and has a capacity of c1=25 units of bandwidth. In addition, each flow is represented by a line that passes through the set of links it traverses. For instance, flow f6 traverses links l1, l2 and l3.

The bottleneck structure of this network corresponds to the following digraph (see [G2-TREP] for details on how a bottleneck structure is computed):

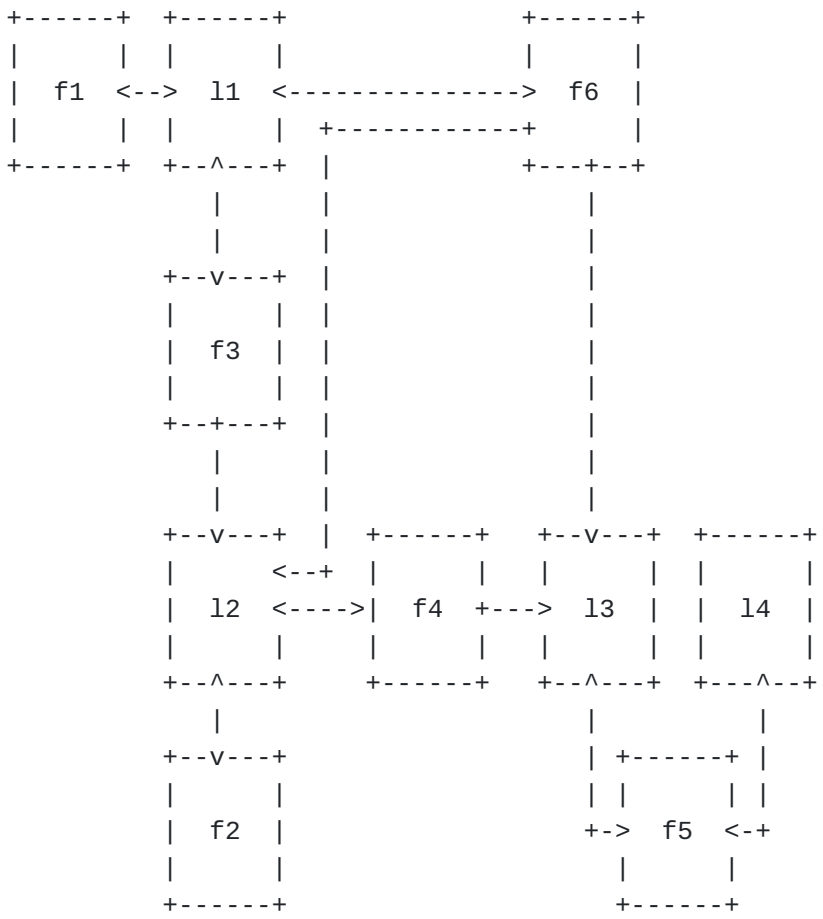

Figure 2: Bottleneck structure of the network in Figure 1.

The bottleneck structure is interpreted as follows:

- Links and flows are represented by vertices in the graph.
- There is a directed edge from a link l to a flow f if and only if flow f is bottlenecked at link l.
- There is a directed edge from a flow f to a link l if and only if flow f traverses link l.

For instance, in Figure 2 we have that flow f3 is bottlenecked at link l1 (since there is a directed edge from l1 to f3) and it traverses links l1 and l2 (since there is a directed edge from f3 to l1 and from f3 to l2). Note that when a flow is bottlenecked at a link, then the edge connecting them in the bottleneck structure must necessarily be bidirectional. This is because a flow that is bottlenecked at a link, must necessarily traverse that link too. Indeed, in Figure 2 we can see that all the directed edges from a link to a flow, are in fact bidirectional edges. This is important to ensure that bottleneck structures correctly model how

perturbations in a network propagate, as we explain in the next section.

3.2. Propagation Properties

Under the assumption of max-min fairness [GALLAGER], QTBS demonstrates the following two properties [G2-SIGMETRICS]:

* **Property 1. Flow perturbation**. An infinitesimal change in the transmission rate of a flow f will have an effect on the transmission rate of a flow f' if and only if the bottleneck structure has a directed path from flow f to flow f'.

* **Property 2. Link perturbation**. An infinitesimal change in the capacity of a link l will have an effect on the transmission rate of a flow f' if and only if the bottleneck structure has a directed path from link l to flow f.

The above two properties qualitatively relate to the classic question in chaos theory: Can the flap of a butterfly's wings in Brazil set off a tornado in Texas? [LORENZ] Obviously a butterfly alone cannot create a tornado, but every element is interconnected in a distributed system, and even the flap of a butterfly's wings in Brazil will have an effect in Texas. Bottleneck structures are graphs that characterize and quantify such type of effects in a communication network. In particular, a bottleneck structure reveals how a perturbation propagates through the network, describing which flows will be affected and by what magnitude.

3.3. Forces of Interaction among Flows and Links

Bottleneck structures are powerful computational graphs because they are able to capture the forces of interaction that flows and bottleneck links exert on each other. These forces of interaction are in general non-intuitive, even for a small simple network configuration like the one in Figure 1. For instance, from Property 2, the bottleneck structure reveals that a small variation in the capacity of link l2 (e.g., in a wireless network, a variation in the capacity of a link could be due to a change in the signal to noise ratio of the communication channel) will propagate through the network and have an impact on the transmission rate of flows f2, f4 and f5 (since from Property 2, the bottleneck structure has a directed path from link l2 to each of these flows). However, such a perturbation will have no effect on the transmission rate of flows f1, f3 and f6 (since there is no path from l2 to any of these other flows). Similarly, from property 1, a small perturbation on the rate of flow f4 (e.g., this could be due to the effect of a traffic shaper altering the transmission rate of flow f4), will have an

impact on the rate of flows f2 and f5, but it will have no effect on the rate of flows f1, f3 and f6.

3.4. Ripple Effects in a Communication Network

As another example, given the network in Figure 1, it is also not intuitive to foresee that flows f1 and f5 are related to each other by the forces of interaction inherent to the communication network, even though they do not traverse any common link. Specifically, flow f1 traverses link l1, while flow f5 traverses links l3 and l4. In between both flows, there is an additional hop (link l2) further separating them. Despite not being directly connected, the bottleneck structure reveals that a small perturbation on the performance of flow f1 (i.e., a change in its transmission rate), will create a ripple effect that will reach flow f5, affecting its transmission rate. In particular, the perturbation on flow f1 will propagate through the bottleneck structure and reach flow f5 via the following two paths:

f1 -> l1 -> f3 -> l2 -> f4 -> l3 -> f5

f1 -> l1 -> f6 -> l3 -> f5

It is also not intuitive to see that the reverse is not true. That is, a small perturbation on flow f5, will have no effect on flow f1, since the bottleneck structure has no direct path from vertex f5 to vertex f1. In [G2-SIGMETRICS], extensive empirical validation of these results is presented for a variety congestion-controlled IP networks.

3.5. Not all Bottleneck Links Are Born Equal

Bottleneck structures also reveal that not all bottleneck links have the same relevancy. In Figure 2, links at the top of the graph have a higher impact on the overall performance of the network than links at the bottom. For instance, consider link l1. A variation on its capacity will create a ripple effect that will impact the performance of all the flows in the network, since all flow vertices are reachable from vertex l1 according to the bottleneck structure. In contrast, link l3 has a much smaller impact on the overall performance of the network, since a variation of its capacity will affect flow f5, but have no impact on any of the other flows. This information is valuable to network operators and application service providers as it can be used to make informed network optimization decisions. For instance, in edge computing, an operator could choose to place a containerized service (e.g., for extended reality, XR) on compute nodes that would yield communication paths traversing bottleneck links with lower impact on the overall performance of the network (See the use case in Section 4.5 for more details).

Similarly, in network slicing (or capacity planning in general), operators could choose to allocate more bandwidth on those links that are more influential (i.e., those links that are at lower levels in the bottleneck structure) according to the expected traffic pattern in the network slice.

Overall, bottleneck structures provide a mechanism to rank bottleneck links according to their impact on the overall performance of the network. This information can be used in a variety of optimization problems, such as traffic engineering, routing, capacity planning, or resilience analysis, among others.

3.6. Quantifying the Ripple Effects

Bottleneck structures not only allow network operators to reason about the qualitative nature of the forces that flows and links exert on each other, but they also provide a mathematical framework to quantify such forces. In particular, the Quantitative Theory of Bottleneck Structures (QTBS) introduced in [G2-TREP] provides a mathematical framework that uses bottleneck structures as efficient computational graphs to quantify the impact that perturbations in a network have on all of its flows.

One of the core building blocks of the QTBS framework is the concept of *link and flow equations*, which mathematically characterize how a perturbation in a network propagates through each of the link and flow vertices in the bottleneck structure. (See [G2-TREP] for an exact mathematical formulation.) Because quantifying the effect of a perturbation on a system is nothing more than computing a derivative of the system's performance with respect to the parameter that's been perturbed, bottleneck structures can be used as efficient and scalable computational graphs to calculate flow and link derivatives in a communication network.

Consider for instance the computation of the following derivative:

dF()/dri

where F() represents the total throughput of the network (the sum of all flows' throughput) and ri is the transmission rate of flow fi. For example, this expression can be used to compute the effect of traffic shaping a flow (slightly reducing its rate) on the total throughput of the network. Computing this derivative using a traditional calculus approach is both very complex and costly, since it requires modeling the congestion control algorithm in the function F(), for which there is no closed form solution. Using bottleneck structures, however, the computation of this derivative is both simple and inexpensive. It is simple because it can be done by applying an infinitesimal change to the rate of flow fi and then

using the link and flow equations to measure how this perturbation propagates through the bottleneck structure [G2-TREP], [G2-SIGCOMM]. It is also very efficient because the computation is performed by applying delta calculations on the bottleneck structure, without involving links and flows that are not affected by the perturbation. For instance, in Figure 1, the computation of dF()/df4 only requires recomputing the transmission rates of flows f2 and f5, without the need to recompute the rates of f1, f3 and f6, since these other flows are not affected by the perturbation. In practice, QTBS provides a methodology to compute network derivatives two or three orders of magnitude faster than general purpose methods such as liner programming [G2-SC].

We finish this brief introduction to QTBS by stating the monotonic bandwidth allocation property that all bottleneck structures satisfy:

* **Property 3. Monotonic bandwidth allocation (MBA).** Let si be the transmission rate of the flows bottlenecked at link li. Then, for any path in the bottleneck structure of the form

  l1 -> f1 -> l2 -> f2 -> (...) -> ln -> fn

  we have that

  s1 < s2 < (...) < sn

The MBA property is relevant in that it states that bottlenecks located at higher levels in the bottleneck structure will have more bandwidth available than those located at lower levels. For instance, this property indicates that an application requiring high bandwidth should route its traffic through paths that involve links at higher levels in the bottleneck structure. We will be using the MBA property to reason about application performance in some of the examples described in this document.

3.7. Types of Bottleneck Structures

While QTBS introduces a core definition of bottleneck structure (see Section 3.1), there exist multiple types of bottleneck structures that can be computed depending on the level of granularity and information desired by the operator. Next, we introduce three types of bottleneck structures that will be used in this document and that are suitable to optimize application performance in the context of the ALTO standard:

* **Flow gradient graph (FGG).** This type of bottleneck structure corresponds to the base definition introduced in Section 3.1. The FGG has the finest level of granularity, including a vertex in

each graph for each link and flow in the network. Therefore, an FGG can be relatively large (e.g, with millions of vertices).

* **Path gradient graph (PGG)**. One technique to reduce the size of the bottleneck structure without affecting its accuracy is to collapse all the vertices of the flows that follow the same path into a single vertex called a *path vertex*. The resulting bottleneck structure is called the path gradient graph (PGG). A PGG usually has 2 or 3 orders of magnitude less vertices than the FGG.

* **QoS-Path gradient graph (Q-PGG)**. Some networks assign different types of traffic to different QoS classes. A Q-PGG can model QoS by collapsing all the vertices of the flows that follow the same path and have the same QoS class into a single vertex called a *Q-path vertex*. A Q-PGG is slightly larger than a PGG (with about |Q| times more vertices, where |Q| is the number of QoS classes supported by the network) but still significantly smaller than the FGG.

For most of the applications, it is recommended to use a PGG, or a Q-PGG if the network supports QoS classes, since these bottleneck structures are significantly smaller and faster to process than an FGG, and it is often the case that the operator does not need to know flow-level information in order to make proper application performance optimization decisions. Note also that the PGG and the Q-PGG provide the additional security advantage of hiding flow-level information from the graph. This can be important to operators that are sensitive about security and privacy.

3.8. Computing Optimized Network Reconfigurations

A central element to the theory of bottleneck structures is the ability to efficiently compute derivatives on a network. Derivatives are a core building block of the optimization framework, as they reveal the directions (gradients) in the feasible set that can help bring the network to a higher level of performance. In this document, we will refer to these directions in the feasible set as *network reconfigurations*, since that's what they effectively are in the physical world.

For instance, an example of network reconfiguration can be the action of rate limiting a flow carrying XR traffic to match the available bandwidth along its path with the goal to improve its QoE. Another example of network reconfiguration is the action of rerouting traffic through a new path in order to accelerate the transfer of a large backup data set between two cloud data centers. A third example can be the deployment of a new network slice in a 5G network in order to ensure the QoS of a V2X service. In each of

these actions, the network configuration is moved along a direction (a gradient, if the change maximally improves the performance objective) within the feasible set of possible configurations.

While derivatives describe how the performance of a network changes when a very small (infinitesimal) change is applied to its configuration, network reconfigurations can accept changes to the network that are arbitrarily large. For instance, traffic shaping a set of flows to reduce their rates by 10 Mbps is a network reconfiguration that is not infinitesimal. We note that bottleneck structures can also be used to compute optimized network reconfigurations consisting of non-infinitesimal changes in the network. This can be done by first computing derivatives using the bottleneck structure to find a direction (gradient) in the feasible set, and then reconfiguring the network by following that direction. This process can be repeated iteratively until a final optimized reconfiguration is achieved. (See for example [G2-SIGCOMM] and [G2-TREP] for examples of algorithms using this technique.)

In the next section, we summarize some of the network reconfigurations that can be optimized by using bottleneck structures.

3.9. Types of Network Reconfigurations

The following is a list of some of the network reconfigurations that can be efficiently computed and optimized using bottleneck structures:

* **Flow routing**. Both the operation of routing a new flow or rerouting an existing flow on a network can be modeled as a perturbation, whose impact can be efficiently measured using bottleneck structures. In particular, QTBS can be used to resolve the joint congestion control and routing optimization problem for individual flows (see Section 3.1 in [G2-TREP]).

* **Traffic shaping**. Traffic shaping a flow corresponds to the action of taking a derivative with respect to the rate of the flow. Bottleneck structures can be used by network operators and application service providers to compute such perturbations. For instance, to accelerate a large scale data transfer, an application can use bottleneck structures to identify optimal traffic shaping configurations (see Section 3.3 in [G2-TREP]).

* **Bandwidth enforcement**. In high-performance networks that target close to 100% link utilization such as Google's B4 network [B4-SIGCOMM], a centralized SDN controller is used collect the state of the network and compute an optimized multipath bandwidth allocation vector. The solution is then deployed at the edge of

the network using a technique known as bandwidth enforcement [BE-SIGCOMM]. By using bottleneck structures to efficiently compute changes in the bandwidth allocated to each flow path, operators can efficiently derive improved bandwidth allocation vectors.

* **Flow scheduling**. When a flow initiates transmitting data on a network, it uses bandwidth along its path, creating a ripple effect that impacts the performance of other flows in the network. Similarly, the termination of a flow frees bandwidth along its path, creating another perturbation that propagates through the network. Bottleneck structures can efficiently model and compute the effect of flow arrival and departure in a communication network by using simple delta calculations according to the link and flow equations (see Section 3.6 and [G2-SIGCOMM]). This information can be used by applications that need to perform bulk data transfer to decide when to schedule a flow. More in particular, it can be used to enhance the ALTO Cost Calendar service [RFC8896].

* **Service placement**. Deploying application services in a network requires deciding the location of the compute and storage resources needed to run the service. For instance, in edge computing, an extended reality (XR) server could be deployed at the distributed unit (DU), the central unit (CU), the mobile core (MC) or the central cloud [PETERSON]. Bottleneck structures can be used to measure the effect of placing a service on each of the candidate locations, helping the application service provider to make optimized decisions.

* **Multi-job scheduling**. Running a job on a network implies a number of flows will be initiated and terminated throughout the execution of the job. the ripple effects generated from the execution of a job can also be measured using bottleneck structures. This can be used to decide when to optimally launch one or more jobs. For instance, in a data center, bottleneck structure analysis can help the application decide how to optimally schedule multiple AI training or inference jobs that are sharing the same interconnect [G2-SIGCOMM].

* **Link capacity upgrades**. In capacity planning, operators often have a fixed budget and need to decide how to optimally add capacity to a network in order to maximize its performance. The effect of a link upgrade operation can be computed as a derivative with respect to a change (an increase) in the capacity of a link. Through the processing of historical flow information from the network (e.g., NetFlow logs), bottleneck structures can efficiently compute the effect of each link upgrade and identify those that yield maximal performance.

* **Path shortcuts**. Operators in wide area networks need to decide whether a communication path should be set up as purely optical (bypassing layer 3 routing) or undergo an optical-to-electrical-to-optical (OEO) conversion at certain routers in order to perform layer 3 routing [SH-SIGCOMM]. The trade-off is one of cost-efficiency versus better routing control of the network. Bottleneck structures can be used to search for paths that are optimally suitable for being offloaded to a purely optical path. These are also known in the literature as path shortcuts [SH-SIGCOMM].

4. ALTO Bottleneck Structure Service Use Cases

Applications of bottleneck structure analysis expand through a broad class of optimization problems that include traffic engineering, routing, flow scheduling, resiliency analysis, network slicing, service level agreement (SLA) management, network design and capacity planning, to name only a few. In this section, we briefly describe some of the use cases that relate to the objectives of the IETF ALTO Standard.

4.1. Application Rate Limiting for CDN and Edge Cloud Applications

In applications such as CDN, XR or gaming, it is important to throttle the transmission rate of flows to match the true available capacity along their communication path. Transmitting at a lower rate than the available bandwidth leads to lower quality of experience (QoE). Transmitting at a higher rate increases packet losses, which wastes network resources and also leads to a lower QoE.

Estimating the available bandwidth for a flow is complex because it depends on multiple factors including the network topology, the routing configuration and the set of dynamic flows using the network resources. Bottleneck structures capture in a single digraph these three factors, creating a model that allows to estimate the performance of each flow. See for instance Sections 3.1 and 3.2 in [G2-TREP] for examples on how bottleneck structures can be used to estimate the available bandwidth of an application.

An ALTO server could help the application service provider obtain the available bandwidth on a given path by exposing the bottleneck structure of the network. With this information alone, the provider could directly obtain the available bandwidth. Alternatively, the application service could query the ALTO server by passing the path for which the available bandwidth needs to be computed, and the ALTO server could return this value without the need to share the complete bottleneck structure.

4.2. Time-bound Constrained Flow Acceleration for Large Data Set Transfers

Bulk data transfer is an important application to both commercial and government supported networks. For instance, Google's B4 network supports large-scale data push synchronizing state across multiple global data centers [B4-SIGCOMM]. Another common use case is found in science networks, where massive data sets such as those originated from the Large Hadron Collider at CERN, Switzerland, need to be shared with scientific labs around the world. In this section, we show how bottleneck structures can be used to reconfigure a network towards accelerating a given data transfer with the goal to meet a certain time constraint.

To illustrate this use case, we will assume the simple bottleneck structure shown in Figure 3.

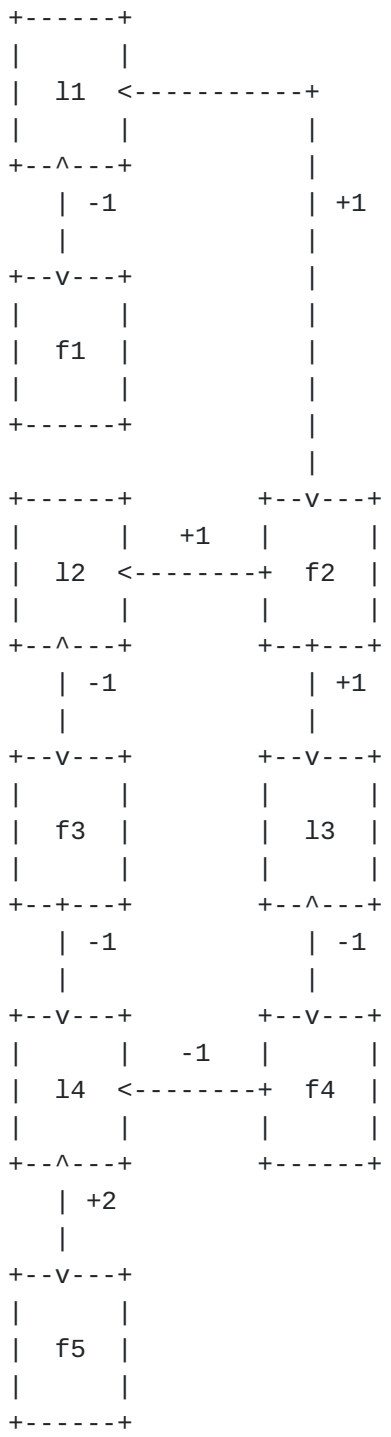

Figure 3: Reducing the rate of flow f1 maximally accelerates flow f5.

Suppose our goal is to accelerate flow f5. To achieve this objective, we will also assume that we are allowed to traffic shape (reduce) the rate of any of the other flows. Effectively, for each flow fi different than f5, we need to compute the following derivative

-dr5/d_ri

and then pick the maximum value. Note that in the above expression, we take the left-derivative (d_), since a traffic shaper reduces the rate of a flow. We also negate the derivative (-d), since we are interested in a positive impact induced on flow f5 when reducing the rate of another flow fi.

Such a calculation can be efficiently performed using the bottleneck structure. As an example, Figure 3 illustrates how the value of (-dr5/d_r1) is computed. First, we reduce the rate of flow f1 by 1 unit. This perturbation propagates through the bottleneck structure reaching flow f5 via two paths:

```
l1 -> f2 -> l2 -> f3 -> l4 -> f5
```

```
l1 -> f2 -> l3 -> f4 -> l4 -> f5
```

Using the link and flow equations (Section 3.6), each path simply flips the sign of the perturbation every time a link vertex is traversed. (The reason why the sign is flipped at each link vertex is explained by the link and flow equations that dictate how perturbations propagate through the bottleneck structure. Further mathematical descriptions to explain this effect are outside the scope of this document. For detailed mathematical derivations and additional examples, please see [G2-TREP]).

When reaching vertex f5, we find that each path contributes 1 unit of bandwidth. Thus we have:

```
-dr5/d_r1 = 1 + 1 = 2
```

In fact, it can be seen that this derivative is maximal. That is, traffic shaping any other flow would yield a smaller increase in the rate of f5. Thus, an operator can conclude that traffic shaping flow f1 yields an optimal strategy to maximally accelerate the rate of flow f5. Note also that in this case, there is a positive multiplier effect, since reducing flow f1's rate by 1 unit, leads to an increase on flow f5's rate by more than 1 unit. This is known as a power gradient [G2-SIGCOMM].

While left outside the scope of this document, bottleneck structures can also be used to efficiently compute the value of the optimal traffic shaper (i.e., in our example, to find by how much we should traffic shape flow f1) and to quantify the impact on the flow being accelerated. This information can also be used by the application to estimate the flow's completion time.

An ALTO server could help the application service provider identify an optimized traffic shaping strategy by exposing the bottleneck structure of the network. With this information alone, the provider could efficiently compute an optimized set of traffic shapers.

Alternatively, the application service could query the ALTO server by passing the set of flows that are allowed to be traffic shaped and the flow that needs to be accelerated, and the ALTO server could return the set of recommended traffic shapers.

4.3. Application Performance Optimization Through AI Modeling

A relevant and emerging area in the field of application performance is AI-based network modeling. Several global initiatives are been undertaken to apply AI to the field of understanding and predicting network performance. For instance, OpenNetLab [BE-ONL] provides a distributed networking platform with many collaborative nodes (universities and companies) and common benchmarking datasets for researchers to collect real networking data and train their AI models for various networking environments, including the Internet, cloud, and wireless networks. There also exist global benchmarks and challenges to foster innovation in this field, such as the ACM MMSys Challenge [MMSYS], which focuses on novel AI-based bandwidth estimation algorithms to enable superior overall QoE on a global production testbed built for real-time communications (RTC) of video and audio.

Modeling communication networks using purely AI frameworks such as deep learning is challenging as it requires very large production data sets that often times are not available. They also require many years of intense, global collaborative R&D. To address these challenges, it has been seen that hybrid models built by combining AI with a "physics" model of the network can outperform purely AI models, as they may require less training data to achieve the same or better performance. For instance, the top two winners of the ACM MMSys 2021 Bandwidth Prediction Challenge [MMSYS] were based on hybrid models.

Because bottleneck structures provide a "physics" model of the network that can both qualify and quantify the forces of interactions among flows and links, they can be used in combination with AI to enable better performance than purely AI-based models. For instance, this area is being discussed in the IETF ALTO WG (e.g., [BE-ONL]) as a potential use case in the ALTO Standard to help optimize the performance of RTC applications. In particular, a key building block to optimize the QoE performance of RTC applications is the bandwidth estimation module. This module runs on the endpoint of a real-time video application and aims at dynamically adapting the video bitrate to stay within the available network capacity. A limitation in the current algorithms, however, is their lack of network state visibility. This requires the algorithms to rely entirely on local indicators such as packet loss or latency, which leads to poor training and inference performance. Information provided by the bottleneck structure (which includes

topological, routing and flow information of the network in a single digraph) exposed via the ALTO service could help unlock a richer set of machine learning algorithms to optimize application performance.

An ALTO server could help the application service provider implement AI-assisted prediction algorithms by exposing the bottleneck structure of the network. Alternatively, ALTO could implement an AI-assisted prediction module with the help of bottleneck structures. The application would then query the ALTO server to obtain the predicted value.

4.4. Optimized Joint Routing and Congestion Control

In traditional IP networks, the problems of flow routing and congestion control are separately resolved by following a two-step process: first, a routing protocol is used to determine the path between any two nodes in a network; then, flows are routed according to such paths and their transmission rates are regulated using a congestion control algorithm. This layered and disjoint approach is known to be scalable but suboptimal because the routing algorithm identifies paths without taking into account the flow transmission rates assigned by the congestion control algorithm.

Suppose that an application is trying to launch a new flow between two endpoints with the goal to maximize the available bandwidth. One can be tempted to think that, to identify the path with maximal available bandwidth, it suffices to look at the current state of the network and find the least congested path offering the highest capacity. This approach, however, is naive since it does not take into account the fact that the placement of the new flow onto the network will itself create a perturbation in the network, potentially making the chosen path suboptimal or, even more troublesome, negatively affecting the performance of other priority flows.

The goal of the joint routing and congestion control problem between two given endpoints E1 and E2 consists in finding the path from E1 to E2 that will yield the highest throughput *after* the flow is placed on the network (i.e., taking into account the effect of placing the flow).

The solution to this problem is introduced in [G2-TREP] by employing a strategy that combines the strengths of both the Dijkstra algorithm and the insights revealed by the bottleneck structure. The algorithm can both compute the optimal path and measure the overall network-wide impact of deploying the new flow on the path. It also enables a framework to identify new good-performing paths that have a limited negative impact on the rest of the flows in the network. This allows network and application providers to identify paths that

can both provide good performance to the newly added application flow while preserving the performance of the existing high-priority flows.

An ALTO server could help the application service provider optimize the path selection decision by exposing the bottleneck structure of the network. With this information alone, the provider could efficiently compute the optimal path (e.g., using the algorithm introduced in [G2-TREP]). Alternatively, the application service could query the ALTO server by passing the information of the two endpoints that need to be connected, and the ALTO server could return a list of the top-N paths with the highest throughput and their expected performance.

4.5. Service Placement for Edge Computing

Determining the proper location to deploy an application service in the edge cloud is critical to ensure a good quality of experience (QoE) for its users. Yet the service placement problem is known to be NP-Hard [JSP-INFOCOM], requiring heuristics to compute good (albeit suboptimal) solutions.

In [G2-SIGCOMM], it is shown that Bottleneck structures can also be used as highly scalable network simulators to evaluate the performance of a network reconfiguration such as the placement of a new service on a edge cloud. In particular, bottleneck structures can very efficiently (1) compute the performance of each flow in the network and (2) quantify the effects of the arrival (departure) of new (existing) flows to (from) the network. This allows to simulate the full transmission of an application traffic pattern very efficiently, three or more orders of magnitude faster than traditional packet simulators.

Network and application providers can use this capability in two ways:

* Given a set of possible placement strategies, bottleneck structures can be used to simulate them in real time, helping the operator select the one that provides the best performance while guaranteeing the service level agreements (SLAs) of the other existing applications.

* Despite the server placement problem being intractable, bottleneck structures provide a framework to identify good candidate solutions. In particular, by capturing the topology, routing, and flow information in a single computational graph, they can be used to efficiently explore directions in the feasible set that yield incrementally better performance. By moving in these incremental directions, the placement algorithm

can progress within the enormous feasible set towards the optimal solution.

An ALTO server could help the application service provider optimize the placement decision by exposing the bottleneck structure of the network. With this information alone, the provider could compute the effect of placing the service in one location versus another. Alternatively, the application service could query the ALTO server by passing the information of the possible locations where it can be placed, and the ALTO server could return an ordered list of the locations and their expected performance.

4.6. Training Neural Networks and AI Inference for Edge Clouds, Data Centers and Planet-Scale Networks

Neural network training and inference using distributed computing systems are the subject of intense research and one of the leading target applications in today's communication networks. [TOPOOPT-MIT] [FLEXFLOW-STFORD] [SINGULARITY-MSFT]. To illustrate this use case, we will focus our discussion on three types of networks: edge clouds, data centers and planet-scale networks.

5G and Edge Clouds enable for the first time the ability to provide intelligence at the edge of the network. This capability is disruptive in that humans and machines will have access to unprecedented compute power to perform AI inference in real time. For instance, using augmented reality (AR), humans will be able to make better informed decisions as they navigate through an environment by leveraging AI-inference on video and audio signals captured in real time from their user equipments (UEs). Similarly, machines such as vehicles or factory robots will be able to use AI inference to optimize their actions.

Two resources are needed to perform inference: (1) Input data from the environment (e.g., image and audio signals captured from a video camera) and (2) compute (typically in the form of GPUs and CPUs). The input data needs to be transmitted from the location where it is captured (e.g., a micro-camera running on a human's glasses) to the location where it is to be processed for inference. The transmission of the input data requires communication resources, whereas the inference process requires computing resources. Since computing resources in the edge cloud (Figure 4) are distributed across the user equipment (UE), the radio unit (RU), the distributed unit (DU) and the central unit (CU) [PETERSON], the problem of efficiently performing AI-inference is one of optimizing the trade-off communication-compute as follows: compute (communication) power is more scarce (abundant) if the inference is performed closer to the UE, and more abundant (scarce) if performed closer to the CU. For instance, if an AR application running on a UE needs to perform an

inference task at a time when the communication path from the RU to the DU is highly congested, then it will have an incentive to perform such a task directly in the UE or in the RU. If instead the network offers an uncongested path to the DU and the CU, it will have an incentive to run the inference task on these other nodes since they offer more compute power.

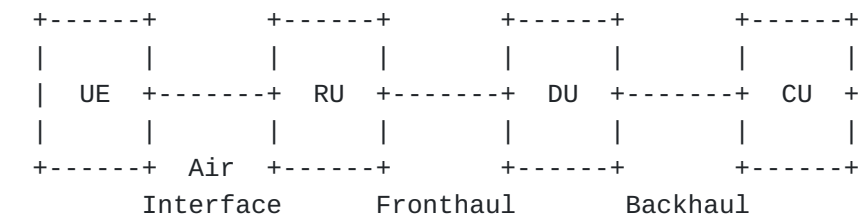

Figure 4: An AI-inference application in the edge cloud needs to place the inference task on a compute node location (UE, RU, DU or CU) that will perform well from both a compute and a communication standpoint.

Using ALTO path vector [I-D.ietf-alto-path-vector] and performance metrics [I-D.ietf-alto-performance-metrics] features, the application could retrieve the amount of compute resources located in the RU, DU and CU. By extending ALTO to support bottleneck structures, the application would also be able to estimate in real-time the available bandwidth for the paths UE-RU, UE-RU-DU, and UE-RU-DU-CU. Further, using bottleneck structure methods described in [G2-SIGCOMM], the application would be able to estimate the time to complete the inference task for each of the four possible scenarios (running in the UE, the RU, the DU or, or the CU) and choose the configuration with the fastest execution.

Similar joint compute-communication optimization problems appear when performing neural network training in large-scale data centers. Large-scale data centers with millions of compute nodes are used to train gigantic neural networks (with potentially trillions of parameters). Such a massive task needs to be broken down into smaller subtasks that are then executed on the nodes. Once again, compute and communication need to be jointly optimized (see [TOPOOPT-MIT] and [FLEXFLOW-STFORD]) in order to ensure regions in the network don't become bottlenecks. By exposing bottleneck structure information using ALTO, the AI-training application can make better subtask placement decisions that avoid potential network bottlenecks.

Finally, AI-training using planet-scale networks generalizes the same joint compute and communication problem to an Internet level [SINGULARITY-MSFT], with the need to implement a global scheduler that is responsible for placing workloads onto clusters of globally-distributed compute nodes. Here too enabling better network state

visibility using ALTO and bottleneck structure graphs could help the scheduler make better task placement decisions.

4.7. 5G Network Slicing

Bottleneck structures can also be used by network operators and application service providers to compute optimized network slices. In a simplified form, given a network topology consisting of a set of routers interconnected via links each with a given capacity, the problem of computing a network slice consists in identifying a subset of the routers, links, and a fraction of the capacity in each link, that can cover a certain demand of traffic generated by a given application service. The slice provides a virtual cut of the network, enabling isolation and ensuring a fix amount of resources to the application service. Examples of application services include a vehicle network service, the Internet of Things, an industrial logistical service, or the metaverse, to name a few.

[G2-SIGCOMM] shows how bottleneck structures can be used to compute optimized bandwidth allocations in data centers to optimally meet a certain traffic demand. Similar frameworks can be used to compute network slices in a virtualized networking environment.

5. Example: Application Layer Traffic Optimization using Bottleneck Structures

In this section we provide an example illustrating how bottleneck structures can be used to optimize application performance. This example will then be referenced in Section 6 to discuss and introduce the necessary requirements to integrate the BSG service into the ALTO standard. It is worth noticing that, as shown in Section 4, bottleneck structures have numerous applications. This section provides a complete example for just one of the use cases. In particular, the focus of the next example is on the joint routing and congestion control use case Section 4.4.

Figure 5 provides a view of Google's B4 network as presented in [B4-SIGCOMM], providing connectivity to 12 data centers distributed across the world (two in Asia, six in America and four in Europe).

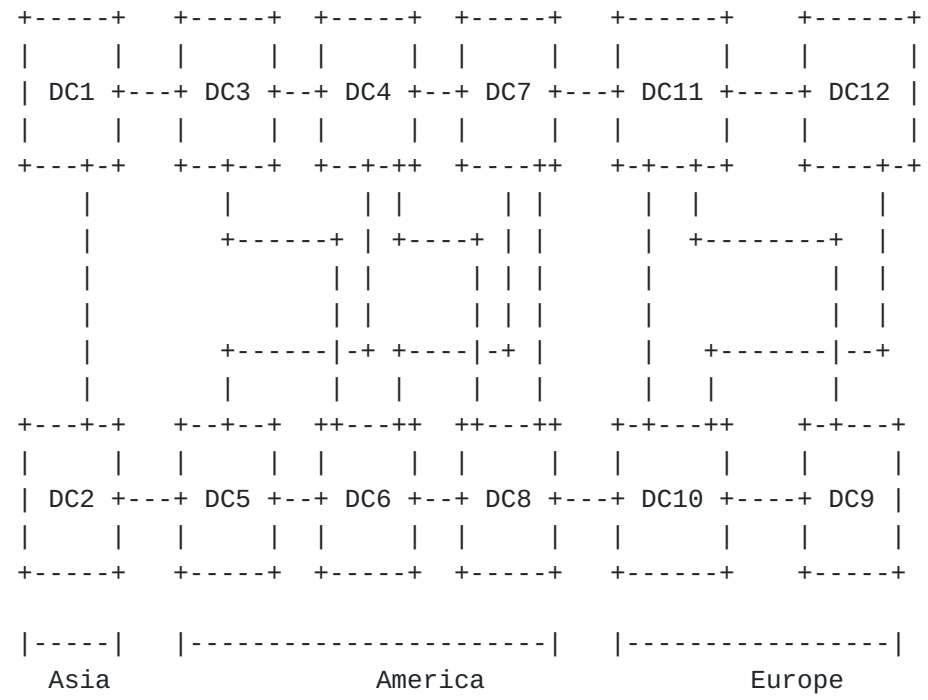

Figure 5: Google's B4 network introduced in [B4-SIGCOMM].

The 12 data centeres are connected via a total of 19 links, labeled l1, l2, ... l19. Table 1 presents the pair of data centers that each link is connected to.

Link	Adjacent data centers	Link	Adjacent data centers
l1	DC1, DC2	l11	DC10, DC12
l2	DC1, DC3	l12	DC4, DC5
l3	DC3, DC4	l13	DC5, DC6
l4	DC2, DC5	l14	DC11, DC12
l5	DC3, DC6	l15	DC4, DC7
l6	DC6, DC7	l16	DC4, DC8
l7	DC7, DC8	l17	DC7, DC8
l8	DC8, DC10	l18	DC9, DC11
l9	DC9, DC10	l19	DC10, DC11
l10	DC7, DC11		

Table 1: Link connectivity (adjacency matrix) in the B4 network.

For the sake of illustration, we will assume a simple configuration consisting of a pair of flows (one for each direction) connecting every data center in the US with every data center in Europe, with all flows routed along a shortest path from source to destination. Since there are six data centers in the US and four in Europe, this configuration has a total of 48 flows. All links are assumed to have a capacity of 10 Gbps except for the transatlantic links (DC7-DC11 and DC8-DC10), which are configured at 25 Gbps.

The next Figure presents the bottleneck structure of Google's B4 network with the assumed flow configuration. Please see Section 3.1 for a description of how to interpret the bottleneck structure. (See also [G2-SC], [G2-TREP] for details on the algorithm used to compute the bottleneck structure.)

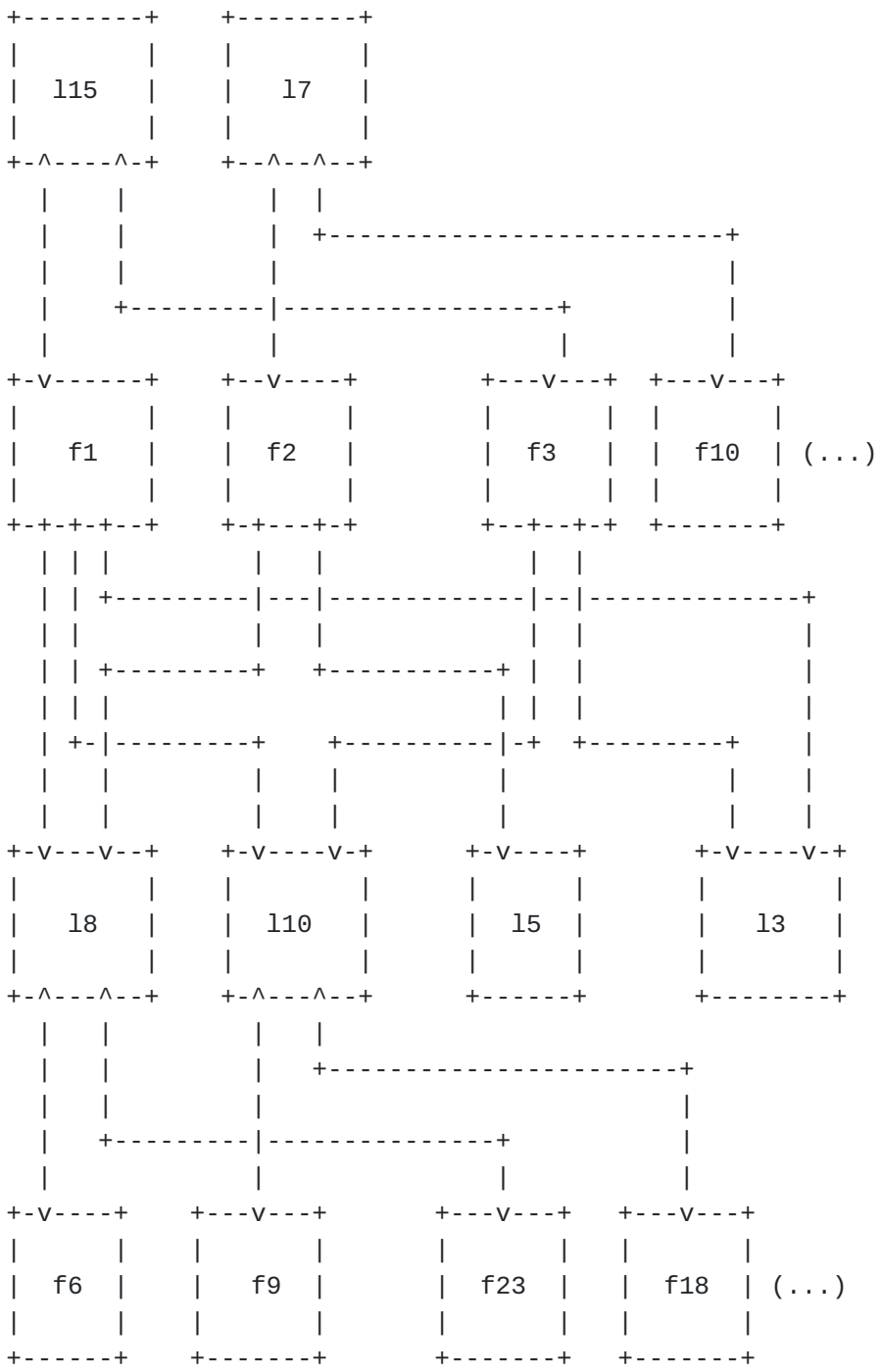

Figure 6: Bottleneck structure of Google's B4 network example.

For the sake of compactness, Figure 6 only includes the bottleneck links and a subset of the flow vertices that are part of the complete bottleneck structure. In particular, out of the 19 links that are part of B4, six links (l15, l7, l8, l10, l5, l3) are bottlenecks.

The bottleneck structure graph shows the existence of two bottleneck levels in our configuration example:

* The first level at the top of the bottleneck structure includes links l15 and l7. Flows f1, f2, f3, f10, etc. are bottlenecked at this level.

* The second level of the bottleneck structure includes links l8, l10, l5 and l3. Flows f6, f9, f23, f18, etc. are bottlenecked at this level.

From the MBA Property (Property 3), we know that flows bottlenecked by a link at level 2 will enjoy higher available bandwidth than flows bottlenecked at level 2. For instance, consider the following directed path in the bottleneck structure:

```
l15 -> f1 -> l8 -> f6
```

Using the MBA property, we have that since l15 precedes l8, it must be that s15 < s8, where s15 is the rate of flow f1 bottlenecked at l15 and s8 is the rate of flow f6 bottlenecked at l8.

Suppose now that an application needs to place a new flow on Google's B4 network to transfer a large data set from data center 11 (DC11) to data center 4 (DC4). The application needs to select and configure a path from DC11 to DC4 (for instance, this could be done by using SR [RFC8402]). The shortest path DC11 -> l10 -> DC7 -> l15 -> DC4 is often used as the default option. Doing so, however, implies that the flow will be bottlenecked at link l15 at the upper level of the bottleneck structure, leading to a lower transmission rate. If instead we choose the non-shortest path DC11 -> l19 -> DC10 -> l8 -> DC8 -> l16 -> DC4, now the flow will be bottlenecked at link l8 (at the lower level of the bottleneck structure), leading to a higher transmission rate.

Using QTBS, we can also numerically compute the transmission rate of the flow on each of the two path options. (See Section 3.1 in [G2-TREP] for a detailed description of how to compute the transmission rate assigned to the flow on each of these paths.) In particular, we obtain that when the application chooses the shortest path (bottlenecked at level 1 of the bottleneck structure), it gets a transmission rate of 1.429 Gbps. If instead the application chooses the slightly longer path (bottlenecked at level 2 of the bottleneck

structure), then it gets a transmission rate of 2.5 Gbps, an increase of 74.95% with respect to the shortest path solution.

[G2-TREP] introduces also a very efficient routing algorithm that uses the bottleneck structure to find the maximal throughput path for a flow in O(V+E*log(V)) steps, where V is the number of routers and E is the number of links in the network.

Overall, this example illustrates that, equipped with knowledge about the bottleneck structure of a network, an application can make better informed decisions on how to route a flow. In the next sections, we will use this example to support a discussion on the requirements for integrating the Bottleneck Structure Graph (BSG) service into the ALTO standard.

6. Requirements

This section provides a discussion on the necessary requirements to integrate the BSG service into the ALTO standard.

6.1. Requirement 1: Bottleneck Structure Graph (BSG) Abstraction

The first base requirement consists in extending the ALTO server with the capability to compute bottleneck structures. For instance, with this capability, given the network configuration in Figure 5, the ALTO server would be able to compute the bottleneck structure shown in Figure 6:

* Requirement 1A (R1A). The ALTO server **MUST** compute the bottleneck structure graph to allow applications optimize their performance using the BSG service.

We note that the alternative, which would consists in ALTO simply providing the necessary information for applications to compute their own bottleneck structures, would not scale due to the following issues:

* Suppose that 1 ALTO server is providing support to N ALTO clients. Then, requiring each application to compute the bottleneck structure would imply performing N identical and redundant computations. By computing the bottleneck structure in the ALTO server, a one-time computation can be leveraged by all N clients. We also note that [G2-SC] demonstrates that bottleneck structures can be efficiently computed in real time by the server even for large scale networks.

* A production-ready high-speed implementation of QTBS is relatively sophisticated. Requiring the applications to implement their own QTBS optimization library would impose unnecessary and (perhaps more importantly) out-of-scope requirements to the

application, which should focus on providing its service rather than implementing a network optimization math library.

The next requirement focuses on the type of bottleneck structure an ALTO server must compute:

* Requirement 1B (R1B). The ALTO server **MUST** at least support the computation of one bottleneck structure type from Section 3.7. Depending on the network information available (e.g., presence of QoS class information), the ALTO server **MAY** support all the three bottleneck structure types, in which case the ALTO client **MAY** be able to choose the bottleneck structure type for retrieval. Also, it is **RECOMMENDED** that the ALTO server supports the computation of the path gradient graph (PGG) as the default bottleneck structure implementation for retrieval by the ALTO clients.

6.2. Requirement 2: Information Received from the Network

To compute a bottleneck structure, two pieces of information are required:

* Topology Object (T). The T Object is a data structure that includes:

  (1) A Topology Graph (V, E), where V is the set of routers and E is the set of links connecting the routers in the network.

  (2) A Capacity Dictionary (C), a key-value table mapping each link with its capacity (in bps).

* Flow Dictionary (F). The F Dictionary is a key-value table mapping every flow with the set of links it traverses.

As shown in [G2-TREP], the above information is enough to compute the bottleneck structure. In fact, with only the F and C dictionaries, one can compute the bottleneck structure. The topology graph (V, E) is needed to perform optimal routing computations (for instance, to find new paths in the network that yield higher throughput, as illustrated in Section 5).

The above discussion leads to the following requirement:

* Requirement 2A (R2A). The ALTO server **MUST** collect information about (1) the set of routers and links in a network, (2) the capacity of each link and (3) the set of links traversed by each flow.

Information about the set of routers, links and link capacity is typically available from protocols and technologies such as SNMP, BGP-LS, SDN, or domain specific topology logs. This information is

enough to construct the T Object. Information about the set of links traversed by each flow can be obtained from protocols such as NetFlow, sFlow, IPFIX, etc. See [FLOWDIR] and [G2-SC] for examples of how requirement R2A is implemented in real-world production networks.

6.3. Requirement 3: Information Passed to the Application

The following requirement is necessary so that applications can optimize their performance using bottleneck structure information:

* Requirement 3A (R3A). The ALTO client **MUST** be able to query the ALTO server to obtain the current bottleneck structure of the network, represented as a digraph.

In addition, the current ALTO services can be extended with additional information obtained from the bottleneck structure:

* Requirement 3B (R3B). One or more ALTO services (the Network Map, the Cost Map, the Entity Property Map or the Endpoint Cost Map) **MUST** support reporting to ALTO clients additional network state information derived from the bottleneck structure to the ALTO client.

For example, the ALTO Cost Map Service can be extended with a new cost metric that corresponds to the estimated available bandwidth between two endpoints according to the bottleneck structure model.

6.4. Requirement 4: Features Needed to Support the Use Cases

Bottleneck structures offer a rich framework to optimize application performance for a variety of use cases. In addition to the base requirement to construct the bottleneck structure graph (see R1A), in this section we enumerate additional capabilities that must be supported by the ALTO BSG service to ensure it is effective in helping applications optimize their performance for each of the supported use cases.

* Requirement 4A (R4A). The ALTO BSG service **MUST** be able to compute the effect of network reconfigurations using bottleneck structure analysis and according to the types described in Section 3.9.

For example, an extended reality (XR) application might need to choose where to place a containerized instance of the XR service among a set of possible server racks located in various edge cloud locations. The application would query the ALTO BSG service to obtain the projected performance results of placing the new service instance on each possible location, allowing it to select the one that would yield the highest performance.

The following requirement is necessary to ensure that the information provided by the BSG service is not stale:

Requirement 4B (R4B). The BSG service **MUST** be able to update the bottleneck structure graph in near-real time, at least once a minute or less.

In [G2-SC] it is shown that bottleneck structures can be computed in a fraction of a session for a production US wide network with about 100 routers and 500 links. Thus, the above requirement should be achievable with a good implementation of the bottleneck structure algorithm [G2-TREP].

7. Security Considerations

Future versions of this document may extend the base ALTO protocol, so the Security Considerations [RFC7285] of the base ALTO protocol fully apply when this proposed extension is provided by an ALTO server.

The Bottleneck Structure Graph extension requires additional scrutiny on three security considerations discussed in the base protocol: Confidentiality of ALTO information (Section 15.3 of [RFC7285]), potential undesirable guidance from authenticated ALTO information (Section 15.2 of [RFC7285]), and availability of ALTO service (Section 15.5 of [RFC7285]).

For confidentiality of ALTO information, a network operator should be aware that this extension may introduce a new risk: As the Bottleneck Structure information may reveal more fine-grained internal network structures than the base protocol, an attacker may identify the bottleneck link and start a distributed denial-of-service (DDoS) attack involving minimal flows to conduct in-network congestion. Given the potential risk of leaking sensitive information, the BSG extension is mainly applicable in scenarios where:

* The properties of the Bottleneck Structure Graph do not impose security risks to the ALTO service provider, e.g., by not carrying sensitive information.

* The ALTO server and client have established a reliable trust relationship, for example, operated in the same administrative domain, or managed by business partners with legal contracts and proper authentication and privacy protocols.

* The ALTO server implements protection mechanisms to reduce information exposure or obfuscate the real information. Implementations involving reduction or obfuscation of the Bottleneck Structure information **SHOULD** consider reduction/

obfuscation mechanisms that can preserve the integrity of ALTO information, for example, by using minimal feasible region compression algorithms [NOVA] or obfuscation protocols RESA [MERCATOR]. We note that these obfuscation methods are experimental and their practical applicability to the generic capability provided by this extension is not fully assessed.

We note that for operators that are sensitive about disclosing flow-level information (even if it is anonymized), then they **SHOULD** consider using the Path Gradient Graph (PGG) or the QoS-Path Gradient Graph (Q-PGG) since these objects provide the additional security advantage of hiding flow-level information from the graph.

For undesirable guidance, the ALTO server must be aware that, if information reduction/obfuscation methods are implemented, they may lead to potential misleading information from Authenticated ALTO Information. In such cases, the Protection Strategies described in Section 15.2.2 of [RFC7285] **MUST** be considered.

For availability of ALTO service, an ALTO server should be cognizant that using Bottleneck Structures might have a new risk: frequently querying the BSG service might consume intolerable amounts of computation and storage on the server side. For example, if an ALTO server implementation dynamically computes the Bottleneck Structure for each request, the BSG service may become an entry point for denial-of-service attacks on the availability of an ALTO server.

To mitigate this risk, an ALTO server may consider using optimizations such as precomputation-and-projection mechanisms [MERCATOR] to reduce the overhead for processing each query. An ALTO server may also protect itself from malicious clients by monitoring the behaviors of clients and stopping serving clients with suspicious behaviors (e.g., sending requests at a high frequency).

8. IANA Considerations

Future versions of this document may register new entries to the ALTO Cost Metric Registry, the ALTO Cost Mode Registry, the ALTO Domain Entity Type Registry and the ALTO Entity Property Type Registry.

9. References

9.1. Normative References

[I-D.ietf-alto-path-vector] Gao, K., Lee, Y., Randriamasy, S., Yang, Y. R., and J. Zhang, "An ALTO Extension: Path Vector", Work in Progress, Internet-Draft, draft-ietf-alto-path-vector-25, 20 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-alto-path-vector-25>.

[I-D.ietf-alto-performance-metrics]
Wu, Q., Yang, Y. R., Lee, Y., Dhody, D., Randriamasy, S., and L. M. Contreras, "ALTO Performance Cost Metrics", Work in Progress, Internet-Draft, draft-ietf-alto-performance-metrics-28, 21 March 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-alto-performance-metrics-28>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, DOI 10.17487/RFC7285, September 2014, <https://www.rfc-editor.org/rfc/rfc7285>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8402] Filsfils, C., Ed., Previdi, S., Ed., Ginsberg, L., Decraene, B., Litkowski, S., and R. Shakir, "Segment Routing Architecture", RFC 8402, DOI 10.17487/RFC8402, July 2018, <https://www.rfc-editor.org/rfc/rfc8402>.

[RFC8896] Randriamasy, S., Yang, R., Wu, Q., Deng, L., and N. Schwan, "Application-Layer Traffic Optimization (ALTO) Cost Calendar", RFC 8896, DOI 10.17487/RFC8896, November 2020, <https://www.rfc-editor.org/rfc/rfc8896>.

9.2. Informative References

[B4-SIGCOMM] Jain et al, S., "B4: Experience with a Globally-Deployed Software Defined WAN", ACM SIGCOMM , 2013.

[BE-ONL] "Bandwidth Estimation on OpenNetLab", IETF Plenary 112, IETF ALTO WG , 2021, <https://datatracker.ietf.org/meeting/112/materials/slides-112-alto-bandwidth-estimation-service-00>.

[BE-SIGCOMM] Kumar et al, A., "BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing", ACM SIGCOMM , 2015.

[FLEXFLOW-STFORD] Jia et al, Z., "Beyond Data And Model Parallelism For Deep Neural Networks", n.d., <https://arxiv.org/pdf/1807.05358.pdf[>.

[FLOWDIR]
Pujol, E., Poese, I., Zerwas, J., Smaragdakis, G., and A. Feldmann, "Steering Hyper-Giants' Traffic at Scale", ACM CoNEXT , 2019.

[G2-SC] Amsel, N., Ros-Giralt, J., Yellamraju, S., von Hoffe, B., and R. Lethin, "Computing Bottleneck Structures at Scale for High-Precision Network Performance Analysis", IEEE International Workshop on Innovating the Network for Data Intensive Science (INDIS), Supercomputing , 2020.

[G2-SIGCOMM]
Ros-Giralt, J., Amsel, N., Yellamraju, S., Ezick, J., Lethin, R., Jiang, Y., Feng, A., Tassiulas, L., Wu, Z., and K. Bergman, "Designing data center networks using bottleneck structures", ACM SIGCOMM , 2021.

[G2-SIGMETRICS]
Ros-Giralt, J., Bohara, A., Yellamraju, S., Langston, H., Lethin, R., Jiang, Y., Tassiulas, L., Li, J., Tan, Y., and M. Veeraraghavan, "On the Bottleneck Structure of Congestion-Controlled Networks", ACM SIGMETRICS , 2020.

[G2-TREP]
Ros-Giralt, J., Amsel, N., Yellamraju, S., Ezick, J., Lethin, R., Jiang, Y., Feng, A., Tassiulas, L., Wu, Z., and K. Bergman, "A Quantitative Theory of Bottleneck Structures for Data Networks", Reservoir Labs (Qualcomm) Technical Report , 2021.

[GALLAGER] Gallager, R. and D. Bertsekas, "Data Networks", 1992.

[JSP-INFOCOM] Poularakis et al, D., "Joint Service Placement and Request Routing in Multi-cell Mobile Edge Computing Networks", n.d..

[LORENZ] Lorenz, E., "Does the flap of a butterfly's wings in Brazil set off a tornado in Texas?", American Association for the Advancement of Science, 139th Meeting , 1972.

[MERCATOR] Xiang, Q., Zhang, J., Wang, X., Guok, C., Le, F., MacAuley, J., Newman, H., and Y. Yang, "Toward Fine-Grained, Privacy-Preserving, Efficient Multi-Domain Network Resource Discovery", IEEE/ACM IEEE/ACM IEEE Journal on Selected Areas of Communication 37(8): 1924-1940, 2019, <https://doi.org/10.1109/JSAC.2019.2927073>.

[MMSYS] "Bandwidth Estimation for Real-Time Communications", 2021, <https://2021.acmmmsys.org/rtc_challenge.php>.

[NOVA] Gao, K., Xiang, Q., Wang, X., Yang, Y., and J. Bi, "An objective-driven on-demand network abstraction for adaptive applications", IEEE/ACM IEEE/ACM Transactions on Networking (TON) Vol 27, no. 2 (2019): 805-818., 2019, <https://doi.org/10.1109/IWQoS.2017.7969117>.

[PETERSON] Peterson, L. and O. Sunay, "5G Mobile Networks: A Systems Approach", Open Networking Foundation , 2020.

[SH-SIGCOMM] Singh et al, R., "Cost-effective capacity provisioning in wide area networks with Shoofly", ACM SIGCOMM , 2021.

[SINGULARITY-MSFT] Shukla et al, D., "Singularity: Planet-Scale, Preemptive and Elastic Scheduling of AI Workloads", n.d., <https://arxiv.org/pdf/2202.07848.pdf>.

[TOPOOPT-MIT] Wang et al, W., "TOPOOPT: Optimizing the Network Topology for Distributed DNN Training", n.d., <https://arxiv.org/pdf/2202.00433.pdf>.

Authors' Addresses

Jordi Ros-Giralt
Qualcomm

Email: jros@qti.qualcomm.com

Sruthi Yellamraju
Qualcomm

Email: yellamra@qti.qualcomm.com

Qin Wu
Huawei

Email: bill.wu@huawei.com

Luis Miguel Contreras
Telefonica

Email: luismiguel.contrerasmurillo@telefonica.com

Richard Yang
Yale University

Email: yry@cs.yale.edu

Kai Gao
Sichuan University

Email: kaigao@scu.edu.cn