Critical Reroute: A Practical Approach to Network Flow Prioritization using Segment Routing

Simon Redman
School of Computing
University of Utah
Salt Lake City, United States
sredman@cs.utah.edu

David Johnson
School of Computing
University of Utah
Salt Lake City, United States
johnsond@flux.utah.edu

Jacobus van der Merwe
School of Computing
University of Utah
Salt Lake City, United States
kobus@cs.utah.edu

***Abstract*—It is widely recognized that reliable communications are a key element of a successful response to a disaster situation. To address this need, local and regional governments in all parts of the world have deployed dedicated communications networks for first responders. These systems are often prohibitively expensive, voice-only, and are needed only on rare occasions. It would be more cost-effective to use already-existing networks or to deploy networks for shared use. However, due to the sudden increase in demand or physical failure caused by the disaster, shared networks may become overloaded. In such situations, it would be desirable to prioritize traffic flows belonging to public safety applications over others. Solutions such as priority queuing and differentiated services provide partial answers to that goal but leave other problems unsolved. This work presents a novel solution using Segment Routing and a Genetic Algorithm optimizer to minimize the impact of network overload on critical traffic flows. The results show that these methods can reroute flows using a single midpoint such that the total network overload is reduced compared to traditional shortest-path routing while avoiding unnecessarily long paths and taking priority of flows into account.**

***Index Terms*—segment routing, genetic algorithms, network reliability, metropolitan area networks**

I. Introduction

Many local and regional governments in all parts of the world have deployed dedicated communications networks for first responders [1]. However, these systems are often prohibitively expensive. They are only needed on rare occasions, so there is little motivation to invest the necessary resources. In the United States, GETS [2] and WPS [3] enable prioritized use of the commercial landline and cellphone networks when needed, thus avoiding the cost of dedicated infrastructure.

Voice-only calls are an important tool, but increasingly emergency services would like to add reliable data communications to their toolbox, which has led to the creation of FirstNet [4]. However, this requires the installation of dedicated infrastructure and is still under deployment. Public safety's takeup of FirstNet has been limited but is gaining momentum [5]. Just as with phone networks, developing the technology to reliably share already-deployed data networks would significantly and immediately benefit the ability of public safety to execute their mission.

This work is supported in part by the National Science Foundation grant number 1647264. 978-1-7281-1434-7/19/$31.00 ©2019 IEEE

We present a practical approach to prioritizing data traffic in the presence of network overload using single-midpoint Segment Routing. This solution operates on a single autonomous system with a relatively simple core network which supports segment routing and programmable traffic classifiers at the edges of the network for installing routing rules. Live traffic and topology information is captured from the network and used to optimize the network in real-time.

Midpoint selection is an NP-hard problem, as discussed in DEFO [6]. Therefore, it is difficult to establish any guarantees about computation time of an optimizer or how a solution compares to the optimal. Nevertheless, we have designed a genetic optimizer which can provide useful solutions while running quickly enough to be practical.

Real-world evaluation is ongoing, and initial results from the Emulab testbed environment [7] show that the system meets its design goal of real-time prioritization of network traffic. Calculation of priority-protecting routes happens on a timescale such that there would be minimal disruption to real-world applications.

This work contributes the following: 1) Design and implementation of an optimizer which computes routing rules to protect priority flows. 2) Design and implementation of a Segment Routing-enabled controller framework to enable real-world, practical flow prioritization.

In essence, our solution tries to reduce overload of links in the network by leveraging alternate paths, taking flow priority into consideration, implemented with Segment Routing

II. Background and Related Works

Traffic engineering is not a new goal, and the proliferation of works solving different aspects of this problem reflect its importance. We present some recent efforts in this area and compare each with our contributions. Segment Routing is a modern Software Defined Networking (SDN) tool which provides the ability to give fine-grained control over the route of a particular packet, which we use to assign midpoints to traffic flows by forwarding each packet of a flow through via a specified router.

1) Optimizing Restoration with Segment Routing [8]: Fundamentally, this work solves a similar problem as they work to eliminate overloaded links by describing the situation

as a linear programming problem. However, to arrive at a form suitable for a linear programming problem, the authors had to relax some constraints, such as allowing a flow to be divisible across infinitely-many midpoints. Such a relaxation is very difficult to implement in the real world. Our work requires that a single midpoint is assigned to each flow. We also do not assume that there is sufficient bandwidth in the network to allow for all flows to travel unimpeded. These constraints more accurately model our application to today's deployed networks.

2) Declarative and Expressive Forwarding Optimizer [6]: (DEFO) is a solution for applying Software-Defined Networking (SDN) to carrier-grade networks. They describe a general-purpose optimizer which can be run on individual flows to build segment-routed midpoints to achieve operator-specified goals. They use reasonable assumptions about a network so that, like our work, it could be applied today. Their evaluation shows that their solver is convincingly able to solve the targeted problem of MinMaxLoad on large networks. Our work solves for a different optimization goal in order to protect priority flows.

3) Expect the Unexpected: Sub-Second Optimization for Segment Routing [9]: This work solves the problem of link overload with special emphasis on small topologies and high-speed correction for change. The evaluation shows that their massively-parallel algorithm can quickly react to changing network demands on large topologies. As with DEFO [6], this work solves a slightly different problem than our work, since it does not prioritize protection of critical flows.

4) Priority Queue Forwarding: Priority queuing, such as DSCP [10], can be seen as the simplest form of network prioritization. In case all traffic in a network must pass through a single bottleneck, Priority Queuing can ensure the most important traffic gets through. However, in cases where multiple parallel routes are available, some more advanced solution should be used.

III. The Critical Reroute Design

Critical Reroute is an ecosystem which operates on a single autonomous domain to minimize the volume of critical traffic flow (the amount of traffic associated with a high-priority application) on overloaded links. This is achieved using segment routing by assigning a single routing midpoint to every flow. To realize its policy, Critical Reroute assumes that there is some programmable traffic classifier on the ingress routers which adds Segment-Routing Identifiers (SIDs) to incoming packets. The traffic classifiers receive commands from the controller describing traffic classes and which SIDs to install. The core of the network must only support segment routing and might otherwise be very simple.

By only requiring segment routing support, this low-complexity design allows the core network to remain simple while supporting many use-cases. For instance, consider a 4G LTE/EPC network operator who wants to prioritize the latency-critical traffic required for voice calls over the less sensitive data needed for buffered video streaming. Alternately, consider a municipal network which would like to support its first responders by prioritizing the video stream of an incident back to the operations center over the day-to-day traffic which is carried on the same network.

Critical Reroute can assign an SID to any packet which the Traffic Classifier can match. For the implementation discussed in Section IV, we assume a traditional source-destination pair, but that is not an inherent requirement of the design.

A. Critical Reroute Architecture

At the highest level, Critical Reroute has two components, shown in Figure 1. The Critical Reroute Controller is responsible for producing segment-routed paths for all flows, which minimize the amount of bandwidth on overloaded links, while the Critical Reroute Traffic Classifier is given directions by the controller to implement the policy.

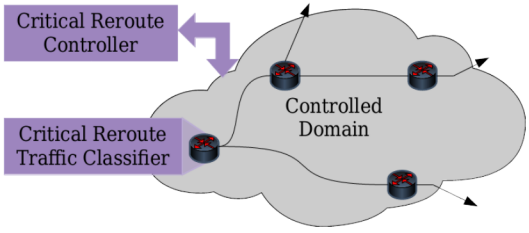

Fig. 1: Architecture Diagram

Figure 2 shows a logical view of the internal components of the Controller. In the southbound direction, the Controller's Network Manager communicates with the network to collect live topology information, useful when the network might be changing due to node or link failure. Once the Route Planner has computed optimized midpoints, those are processed by the Network Manager into actionable directives and pushed to the network. Flow information, such as actual bandwidth usage, is also collected by the network and delivered to the controller.

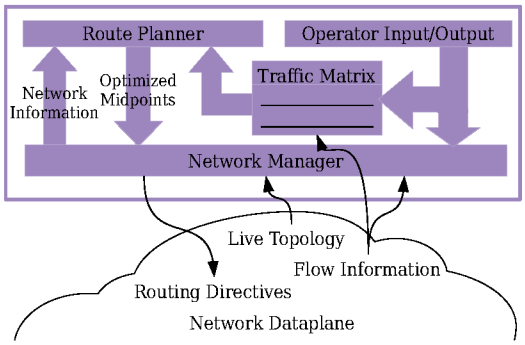

Fig. 2: Critical Reroute Controller Components

To compute optimized midpoints, the Critical Reroute Controller contains a Traffic Matrix of flows annotated with the priority and bandwidth allocation of each flow. Flows are populated by live information, with the option of the operator defining static flows with dedicated bandwidth allocations. This Traffic Matrix and the current topology information are the inputs to the Route Planner.

In the case that human intervention is required, such as defining flow priority match rules or inputting initialization information, the Controller has a northbound Operator Interface. This interface also provides information back to the operator about the state of the network.

B. Route Planner

The core component of the Critical Reroute system is the route planner. The route planner takes as inputs a graph of the physical topology and a traffic matrix annotated with flow bandwidth requests and priority and outputs a list of midpoints, one for each flow, which minimizes the amount of overload on links carrying critical traffic.

While the primary design goal of the Route Planner is to provide optimal routes, an important consideration is that it must run quickly enough that it can adapt to changing circumstances. For instance, if nodes are removed from the topology view or new flows are added to the traffic matrix, the route planner should be able to respond to those changes quickly, such that services using the network are disrupted as little as possible. It should also never return a worse solution than using pure shortest-path routing.

We elect to generate a solution with only a single midpoint per flow for the practical reason that the search space involving one midpoint per flow is vastly reduced compared to any additional midpoints, allowing the solver to run in a realistic amount of time. Simultaneously, one midpoint gives the controller good control over the path the traffic takes based on our experience as well as the experiences discussed in prior work [8].

The following subsections describe the different components of the route planner.

1) Cost Function: In order to define "better" or "minimum", we need to be able to quantify the usefulness of a solution. In optimization problems, this quantifier is called the cost function. Our cost function is three dimensional, with each dimension acting as a tiebreaker for the one before it. The factors considered are: 1) Total amount of critical traffic flow on overloaded edges 2) Sum of the latency of each flow multiplied by its priority value 3) Total number of extra edges used in a solution compared to shortest-path

Essentially, this cost function is a restatement of our goal of avoiding network overload while adding a latency term to capture the idea that we would like our priority flows to receive good service.

2) Priority Value: The priority value of a flow can be any integer or floating-point value greater than or equal to zero. If a priority of zero is assigned to a flow, the optimizer will treat the flow as totally unimportant, and any congestion it experiences would not be optimized, though it still contributes to the used bandwidth of a link. Priority values should be considered relative to other priority assignments in a solution, where higher priority means the flow is more important.

3) Optimizer: With this cost function definition, we can address the problem of how to solve for a "better" or "best" solution. The most important consideration is that the solution space is discrete. The midpoint for any flow can be exactly one core router and flows are not divided into fractional parts, as was done in [8].

Thus, we need an optimizer which works directly on discrete solution spaces. One class which works well in many cases is Genetic Optimizers [11].

4) Genetic Optimization Process: To start, a number of proposed solutions, called 'individuals' are generated. Then, at every step, the individuals are tested against the cost function to determine their 'fitness' with the fittest solutions being given the best chance to be selected to reproduce.

In this work, an allele is a single midpoint for a particular flow, and an individual is a list of such midpoints, one per flow. Each individual represents a valid solution.

5) Selection Algorithm: The first step in an iteration of the Genetic Optimizer is selecting parents which will be used to produce a new individual of the next generation. The literature has many selection algorithms, each with their own tradeoffs. This work uses Tournament Selection since it can avoid evaluating the fitness of every individual in the gene pool.

6) Crossover: Crossover is the step which builds the initial child by copying some section of alleles from each parent. The intuition behind crossover in any genetic optimizer is that the solution might be structured such that nearby alleles influence each other and keeping such groups together will result in a better solution. This might be true, for instance, if flows were sorted by source ID, so the selection of one allele might impact its neighbors.

7) Mutation: After crossover, the resulting child might be mutated. Mutation means that each allele in the child solution might be changed. When a particular allele is selected to be mutated, the midpoint that it represents is moved to a neighboring router uniformly at random.

Once a new individual has been generated, the entire process repeats from selecting new parents to generate another individual until the entire new population has been generated.

8) Initial Population: Over time, the genetic optimizer should converge towards the optimum regardless of how the population is initially seeded. In practice, having a good heuristic for initial population gives the optimizer a faster start. For this work, we build an initial individual by selecting the router which is the midpoint of the shortest path, then copy that individual enough times to fill the gene pool.

9) Termination: In general, it is not possible to determine whether a particular solution is the global optimum. Often, this problem is solved by detecting when the optimization algorithm stabilizes and assuming that the found optimum is "good enough". Another solution is, given a limited time budget, find the best solution possible. Given this work's goal of attempting to optimize a live network, we opt for a time-limited approach.

IV. The Critical Reroute Implementation

Route Planner: All components of the Route Planner are implemented in Python 3. The main work of the Route Planner

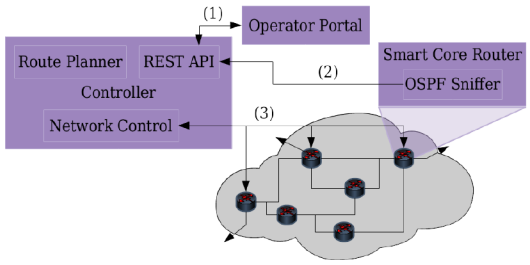

Fig. 3: Detailed Implementation Diagram

is the Genetic Optimizer which is implemented using the DEAP evolutionary computation framework [12]. We use the NetworkX [13] library for handling and manipulating network topology graphs, and we use the NetDiff library (part of OpenWISP, [http://openwisp.org/]) for handling the sharing of NetworkX graphs.

Topology Discovery: In order to do its job, the route planner needs a view of the network topology. We make no assumptions on the core of a network other than support for Segment Routing. For live topology detection, we insert a stand-alone routing protocol packets sniffer, inspired by the design in [14]. The sniffer observes OSPF Link-Stat Advertisements and builds a local graph of the network topology which is sent to the centralized controller. Our sniffer implements OSPF, but any other similar protocol could be used.

Although our sniffer does a good job at detecting broken routers and links, it is not able to detect metadata of the network such as how much bandwidth is available on a link or whether there exist invisible nodes such as layer-2 switches. Such data can be provided to the controller by adding a static topology map with extended information.

Live Traffic Detection: Live network information is a necessity for any traffic engineering endeavor, so there is a significant volume of prior work discussing how it can best be collected [15], [16], [17]. One easy, scalable, and practical solution is to use NetFlow data [18].

NetFlow [19] is a commonly-supported way of collecting network flow information by keeping per-flow statistics at each collector, then exporting it to some centralized location. For this work, we used the open-source Linux iptables Netflow module [20] for the netflow collectors, and the open-source SiLK collector toolkit [21].

SDN Controller: Once the optimizer has calculated optimized midpoints, the controller delivers match-action rules to the ingress routers to add Segment Routing headers to each packet of each modified flow.

The central controller elements are implemented in Python using the Ryu SDN controller framework. Ryu provides a built-in REST library which we have used to implement our REST API to support both the northbound user-facing applications as well as the OSPF collector.

V. Evaluation

We have evaluated Critical Reroute's route optimizer on synthetic and inferred topologies in a variety of bandwidth use cases. Instructions for running a demo of the optimizer can be found here: https://gitlab.flux.utah.edu/sredman/Critical-Reroute

We evaluate our results as compared to pure, un-optimized OSPF as well as against related works where such comparisons can be meaningfully made.

(a) A simple network with three paths

(b) A simple network with two paths

Fig. 4: A simple network based on the testbed being deployed in Ammon, Idhao. Each link has the same latency and can handle 10Gb/s of traffic. The OSPF default route from CB to N10 goes through N2.

Source	Destination	Bandwidth	Priority
CB	N10	7 Gb/s	1
CB	N10	7 Gb/s	1
CB	N10	7 Gb/s	10

TABLE I: Demands applied to Figure 4

A. Critical Reroute can Leverage Multiple Paths

One benefit of customizing paths per-flow is that, when multiple paths exist in a network, it should be possible to load balance across them. Critical Reroute can leverage such paths. Consider the network topology shown in Figure 4a with three paths and three flows, any two of which would cause a path to be overloaded. For this example, Critical Reroute assigns one flow to use N7 as a midpoint, another to use N8, and leaves one flow on the default OSPF path.

B. Critical Reroute Protects Priority Flows

For this example, we use the same topology shown in Figure 4b, but now consider one of the paths has failed. We would still like to route the same three flows through the network, but now overload is unavoidable. Critical Reroute notices that one flow is annotated with a higher priority than the other two, so it selects that flow to go on its own path, while the other two flows are assigned to use the same, now overloaded, path. Critical Reroute protects the higher-priority traffic, while the lower-priority flows get the best possible service given the situation.

C. Critical Reroute Scales to Realistic Networks

1) Experiment Setup: The topologies and demands used to evaluate this work come from the collection of test cases published alongside DEFO [6]. From that dataset, we have elected to use only the inferred topologies originally from the Rocketfuel project [22] since they best correspond to our real-world usecase. All experiments were run on a Xeon E5-2630 v3 CPU with 64GB RAM.

Because DEFO does not optimize with respect to priority values, the traffic matrices we use have no priority annotation.

Due to lack of better information, we have assigned each demand a priority value of 1 to represent non-critical flows and 10 to represent critical flows, with each flow being considered critical with 20% probability. Each traffic matrix is rerun 5 time to evaluate the variance of the optimizer.

2) Results: In many cases, the result presented is Overload Factor, since this is the quantity the optimizer is primarily trying to reduce. Recall that this is computed for a particular solution as the product of the highest priority value on each link, multiplied by the bandwidth usage which exceeds the link capacity, if any, summed over every edge in the network graph. For instance, if for a particular solution a single link in the entire network carrying traffic with a priority value of 10 were overloaded by 1000 bandwidth units, the overload factor would be 10000. Measuring a solution in this way should be thought of in comparison to other solutions on the same topology, rather than as a global measure. In all cases, an overload factor of 0 indicates that the optimizer managed to find a solution with no overload.

The other measure of interest is the latency factor, which is more readily understandable in the real world. Latency factor can be roughly understood in the same units as the link delay (often milliseconds) since it is the product of the delay of every link traversed by a flow multiplied by the flow's priority value. Once a flow has been optimized such that it is avoiding overloaded links, which have an unbounded negative effect on network performance, it becomes interesting to make sure that it is taking the shortest path possible.

Topology Name	Nodes	Edges	Demands
rf3967	79	294	6162
rf1239	315	1944	96057

TABLE II: Summary of tested topologies

3) Analysis: DEFO [6] seeks to optimize the most overloaded link such that, ideally, all links are below their theoretical bandwidth limit. For that work, the generated traffic demands always have some solution where bandwidth per link is below the threshold. In this environment, Critical Reroute can also find a solution such that no link is overloaded. However, this may take more time since Critical Reroute has to compute a more complicated cost function.

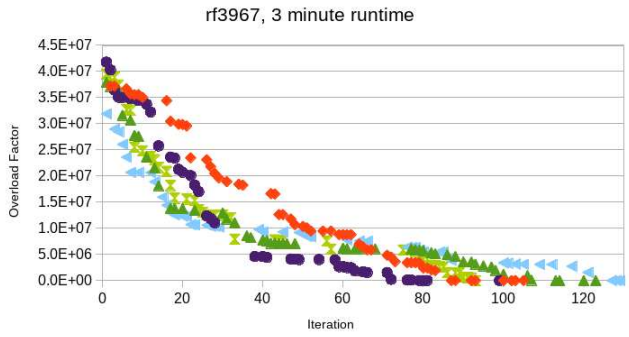

Fig. 5: Overload Factor vs. Iterations for the rf3967 topology

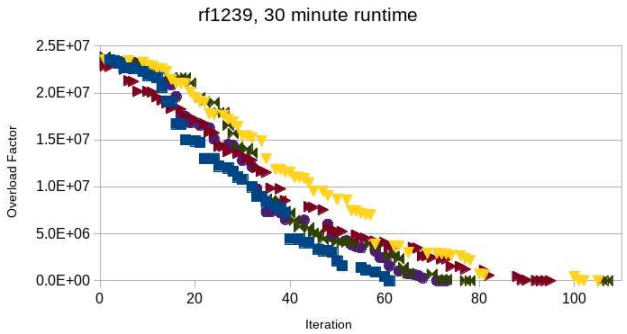

Fig. 6: Overload Factor vs. Iterations for the rf1239 topology

Figures 5 and 6 show example runtimes of the tested topologies and show the decrease in cost function over time for each of the five trials on a particular topology using the same traffic matrix and priority value annotations.

Figure 5 shows the relatively small 3697 Rocketfuel [22] topology. Because of its size, it does not take long for the optimizer to find an initial solution which eliminates overload.

Figure 6 shows runtimes of the largest topology we have evaluated against. On our test hardware, Critical Reroute needs roughly 30 minutes to converge to an initial solution which eliminates overload on this topology. This is a larger scale than Critical Reroute was originally designed to handle, but is a good example of the limits of the current approach and the ability of these methods to apply to more general problems.

It may seem that 30 minutes, or even 3 minutes, is too long to wait in a real-world scenario. However, this is just the initial setup time. In an operational deployment, we would be more concerned with how well Critical Reroute can adapt to changes in the network to keep traffic flowing smoothly. Section V-D simulates how the optimizer might respond to the dynamic nature of a live network.

Though DEFO [6] is able to optimize for its cost function in all listed topologies in 3 minutes or less. Critical Reroute's cost function is more complicated to take into account the priority and latency of individual flows; thus it is not able to solve larger problems as quickly.

Each test with the same inputs was rerun five times to test a total of 80 traffic matrices with different priority assignments for each topology. In all runs, the optimizer was able to find a solution which brought the total overload to 0. This is an indicator that the optimizer is not getting stuck in local optima or that at least those local optima result in useful solutions. This has the real-world implication that leaving the optimizer to run for the maximum available amount of time would result in the best solution, as opposed to restarting the optimizer to avoid locally-optimal solutions.

Table III shows the average factor increase in latency and path length for the tested topologies. This equates to a latency increase of about ten milliseconds and one additional link. When considering that the initial network was suffering from severe overload, causing routers to drop significant amounts

of traffic, network performance would have been terrible. An increase of one extra link is a low price!

Topology	Latency	Path Length
rf3967	1.12x	1.12x
rf1239	1.15x	1.20x

TABLE III: Average factor increase in latency and path length for the segment-routed solutions compared to OSPF

D. Critical Reroute Responds to Network Changes

Once the optimizer has converged to an acceptable solution, it is expected that the network will only change in small steps. Therefore, instead of being started from scratch, we can re-use the midpoint assignment from the previous solution as a starting point for a new solution.

For this simulation, we assume that Critical Reroute has been given enough time to find a solution which eliminates overload. Then, 1% of the nodes in the network are removed. Even as the network experiences continuing failure, Critical Reroute can respond in a reasonable amount of time to maintain service performance.

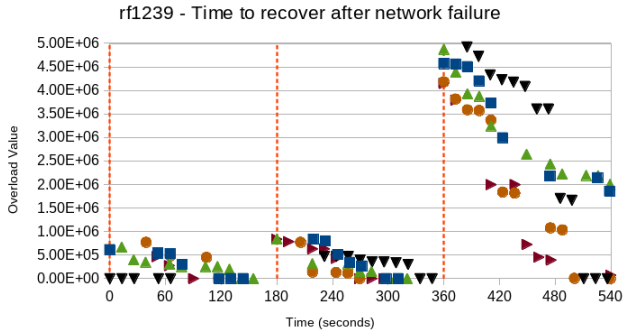

Fig. 7: Graph showing recovery time from network failure. The dashed vertical line indicates the point where 1% of the remaining nodes are removed from the network

Figure 7 shows the response time of Critical Reroute to a simulated failure in the larger rf1239 topology. It is not surprising that finding a solution gets harder at every iteration of this process since the same traffic has to share fewer paths. For the first two iterations, Critical Reroute can find a solution which eliminates overload in all test runs. In the third instance, only two test runs are able to completely eliminate overload.

VI. Conclusion

This work presents a practical approach to network flow prioritization using single-midpoint segment routing, automated by a genetic algorithm. Additionally, it describes the design and implementation of sufficient framework to run the system on a physical network. Our evaluations show that the optimizer can achieve its goals of protecting traffic from overload in an environment where traditional tools would not be sufficient.

References

[1] A. Kumbhar and I. Guvenc, "A Comparative Study of Land Mobile Radio and LTE-based Public Safety Communications," Apr. 2015.

[2] "Government Emergency Telecommunications Service (GETS)," https://www.dhs.gov/government-emergency-telecommunications-service-gets, Apr. 2013.

[3] "Wireless Priority Service (WPS)," https://www.dhs.gov/wireless-priority-service-wps, Apr. 2013.

[4] "FirstNet," https://www.firstnet.com, 2018.

[5] A. Ward, "FirstNet Momentum: More Than 2,500 Public Safety Agencies Subscribed — First Responder Network Authority," https://firstnet.gov/news/firstnet-momentum-more-2500-public-safety-agencies-subscribed, Aug. 2018.

[6] R. Hartert, S. Vissicchio, P. Schaus, O. Bonaventure, C. Filsfils, T. Telkamp, and P. Francois, "A Declarative and Expressive Approach to Control Forwarding Paths in Carrier-Grade Networks," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 15–28.

[7] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An Integrated Experimental Environment for Distributed Systems and Networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, ser. OSDI '02. Boston, MA: USENIX Association, Dec. 2002, pp. 255–270.

[8] F. Hao, M. Kodialam, and T. V. Lakshman, "Optimizing restoration with segment routing," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, Apr. 2016, pp. 1–9.

[9] S. Gay, R. Hartert, and S. Vissicchio, "Expect the unexpected: Sub-second optimization for segment routing," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[10] K. Nichols, S. Blake, F. Baker, and D. L. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers," https://tools.ietf.org/html/rfc2474, Dec. 1998.

[11] W. Darrell, "A Genetic Algorithm Tutorial," http://bipad.cmh.edu/ga_tutorial1994.pdf, 1994.

[12] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, Jul. 2012.

[13] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, 2008.

[14] A. Shaikh and A. Greenberg, "OSPF Monitoring: Architecture, Design and Deployment Experience," p. 15.

[15] G. Balestra, S. Luciano, M. Pizzonia, and S. Vissicchio, "Leveraging Router Programmability for Traffic Matrix Computation," in *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, ser. PRESTO '10. New York, NY, USA: ACM, 2010, pp. 11:1–11:6.

[16] A. Davy, D. Botvich, and B. Jennings, "An Efficient Process for Estimation of Network Demand for Qos-aware IP Network Planning," in *Proceedings of the 6th IEEE International Conference on IP Operations and Management*, ser. IPOM'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 120–131.

[17] K. Papagiannaki, N. Taft, and A. Lakhina, "A Distributed Approach to Measure IP Traffic Matrices," in *Proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '04. New York, NY, USA: ACM, 2004, pp. 161–174.

[18] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 265–280, Jun. 2001.

[19] Cisco, "Introduction to Cisco IOS NetFlow - A Technical Overview," *Cisco*, pp. 1–16, May 2012.

[20] "Netflow iptables module for Linux kernel," Apr. 2019.

[21] CERT/NetSA at Carnegie Mellon University, "SiLK (System for Internet-Level Knowledge)," https://tools.netsa.cert.org/silk, 2019.

[22] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP Topologies with Rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2004.