YATES: Rapid Prototyping for Traffic Engineering Systems

Praveen Kumar
Cornell University

Chris Yu
Carnegie Mellon University

Yang Yuan
Cornell University

Nate Foster
Cornell University

Robert Kleinberg
Cornell University

Robert Soulé
Università della Svizzera italiana

ABSTRACT

This paper presents the design and implementation of YATES, a software framework that seeks to dramatically lower the cost of experimenting with different traffic engineering approaches. YATES offers a suite of tools that make it possible to rapidly prototype and evaluate the performance of traffic engineering systems including tools for modeling topologies, routing schemes, demands, prediction algorithms, and failures. YATES comes with two backends: a network simulator that calculates congestion, throughput, loss, latency, etc., and an SDN-based implementation that can be used to validate results obtained via simulation and also provides an easy path to deployment. We evaluate YATES by prototyping 17 TE systems of varying complexity.

CCS CONCEPTS

• **Networks** → *Traffic engineering algorithms*; *Network simulations*; *Network experimentation*;

KEYWORDS

Traffic Engineering, Wide-Area Networks, Simulation, Tools.

ACM Reference Format:
Praveen Kumar, Chris Yu, Yang Yuan, Nate Foster, Robert Kleinberg, and Robert Soulé. 2018. YATES: Rapid Prototyping for Traffic Engineering Systems. In *SOSR '18: ACM SIGCOMM Symposium on SDN Research, March 28–29, 2018, Los Angeles, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3185467.3185498

SOSR '18, March 28–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5664-0/18/03…$15.00
https://doi.org/10.1145/3185467.3185498

1 INTRODUCTION

Imagine you are a network operator and you are asked to develop a new traffic engineering (TE) solution for your wide-area network (WAN). As recently as five years ago, you quite likely would have followed a conventional approach: build a system that tunes link weights so distributed routing protocols such as CSPF and ECMP compute a good set of forwarding paths. But the emergence of software-defined networking (SDN) has made it possible to consider using a much wider variety of solutions than in the past. For example, systems like SWAN [18] and B4 [21] use centralized controllers and constraint programming to compute forwarding paths that provide optimal levels of performance.

But although many interesting TE approaches have been proposed over the years [1, 18, 21, 22, 27, 38], there is not a single best choice that outperforms the others in every possible scenario. In practice, performance varies wildly depending on factors like the network topology and the conditions in which they are deployed. Furthermore, each approach was designed with a different objective in mind—e.g., some approaches optimize for minimizing congestion while others optimize for fault tolerance. To make an informed choice, we would need a way to model various design choices, build prototypes, and perform what-if-style reasoning about trade-offs across many possible alternatives.

In principle, it should be possible to use simulation frameworks such as `ns-2`, `ns-3`, and Mininet [26, 29] to evaluate these trade-offs. But actually doing this would be difficult: because most simulation frameworks model the network at a low level of abstraction, one would need to write a lot of code that is unrelated to the behavior of the TE systems. For example, interactions between individual traffic flows do not typically have a major impact on the performance and robustness characteristics of WANs, since they deal with traffic aggregates comprising millions of flows. What is needed is a framework that abstracts away from low-level details of TE systems while still capturing macroscopic behavior.

To this end, we present YATES (Yet Another Traffic Engineering System), a software framework that is designed to

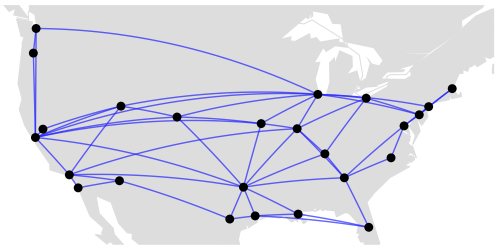


Figure 1: Example topology: AT&T WAN (from Internet Topology Zoo [24]).

dramatically lower the cost of experimenting with different TE approaches. Our design goals for YATES include:

- **Domain-specific:** YATES focuses specifically on WAN-TE and abstracts away from packet-level interactions while still providing enough knobs to accurately capture macroscopic behavior.
- **Modular:** YATES is based on a modular design that makes it possible to express complex TE behaviors as the composition of simpler components.
- **Libraries:** YATES offers a rich library of algorithms and tools that can be used to rapidly assemble TE prototypes.
- **Backends:** YATES provides multiple backends including a software simulator and an SDN backend, which offers a level playing field for benchmarking different TE systems as well as an easy path to deployment.

Contributions: This paper shows how we can use YATES to easily prototype conventional distributed (§2) as well as contemporary centralized (§3) TE systems, and how we can evaluate their behavior under a variety of operational conditions, including in the presence of failures (§4). We present the detailed design and implementation of the YATES framework (§5) along with over a dozen TE systems.

2 CONVENTIONAL TE

To start, let us see how YATES can be used to model a conventional TE system based on the constrained shortest path first (CSPF) algorithm. Although modeling existing systems is arguably not its purpose, this example will illustrate the main features of YATES in a familiar setting.

Consider the topology shown in Fig. 1, and suppose we need to tune the link weights of an existing deployment to better balance the load. However, before we actually modify the weights, we want to ensure that doing so will not degrade the steady-state performance under the current workload.

2.1 YATES Algorithm Modules

YATES provides a simple interface for modeling TE systems in terms of *demands*, *routing schemes*, and *algorithms*, as shown in Fig. 2. A *demand* maps each source-destination pair to a floating-point number that encodes the required bandwidth between those nodes. Demands may either be measured or

```
(* Map from src-dst pairs to traffic demands *)
type demands = float SrcDstMap.t

(* Map from src-dst pairs to path distributions *)
type scheme = (float PathMap.t) SrcDstMap.t

module type Algorithm = sig
  val initialize : scheme -> unit
  val solve : topology -> demands -> scheme
end
```

Figure 2: YATES Interface

```
(* Algorithm : ECMP *)
let solve (topo:topology) (dem:demands) : scheme =
  SrcDstMap.fold dem ~init:SrcDstMap.empty
    ~f:(fun ~key:(u, v) ~data:_ acc ->
      (* Compute all u-v paths *)
      let paths = all_shortest_paths topo u v in
      (* Compute weight in uniform distribution *)
      let prob = 1. /.
        Float.of_int (List.length paths) in
      (* Construct uniform distribution *)
      let uniform = List.fold_left paths
        ~init:PathMap.empty
        ~f:(fun acc path ->
        PathMap.add acc ~key:path ~data:prob) in
      (* Add u-v and distribution to scheme *)
      SrcDstMap.add acc ~key:(u, v) ~data:uniform)
```

Figure 3: ECMP implementation in YATES.

predicted. A *routing scheme* is a mapping from node pairs to probability distributions on paths between those nodes. For example, given an input (u, v) a shortest path routing scheme would return a single path (i.e., the shortest path from u to v) with probability 1.0, whereas a more complicated scheme might return a set of paths, each weighted by a different probability [18, 34, 38]. A TE *algorithm* computes a routing scheme given a topology and a demand. In simple cases, the algorithm may depend only on the topology (e.g. shortest path routing), but more generally it may maintain state, or adapt to changing conditions as encoded in the demands. Hence, a TE algorithm may be invoked repeatedly, in response to changes in demands or the topology.

2.2 ECMP and CSPF

ECMP spreads traffic uniformly across all shortest paths from a source to a destination. These shortest paths, used to carry traffic, can be manipulated by changing link weights. To prototype ECMP in YATES, we implement a TE algorithm module that: (i) given a topology, computes all shortest paths for every node pair and (ii) generates a routing scheme that assigns uniform probability to the shortest paths for each node pair. Fig. 3 shows that the corresponding YATES implementation is just a few lines of OCaml code.

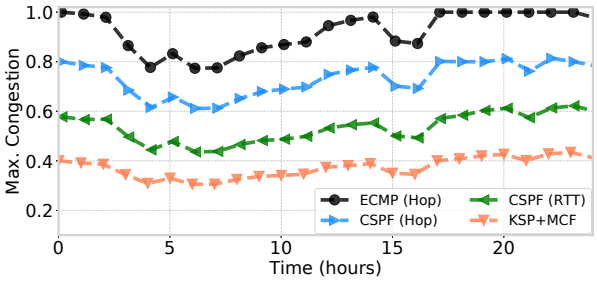


Figure 4: Timeseries of max. congestion on AT&T topology.

Similarly, CSPF is a distributed routing algorithm that routes traffic over shortest paths that have sufficient available bandwidth. Many deployments reserve headroom on each link to handle unexpected surge in traffic. Let us assume that all link weights are set to 1 so that length of a path is its hop-count, and that 20% of the capacity on each link is reserved as headroom. Returning to the top-level example, we wish to decide if modifying the link weights to be proportional to their RTT would improve performance.

To evaluate this choice using YATES, we first provide the topology and a timeseries of demands (generated using gravity model described later) to the system. YATES simulates traffic based on the demands and routes it based on the specified TE algorithm to provide a detailed analysis of network performance. For instance, Fig. 4 shows the maximum link congestion in the network for the TE approaches mentioned above on the sample topology. It is easy to see that the RTT-based implementation of CSPF outperforms the implementation based on hop counts in this setting.

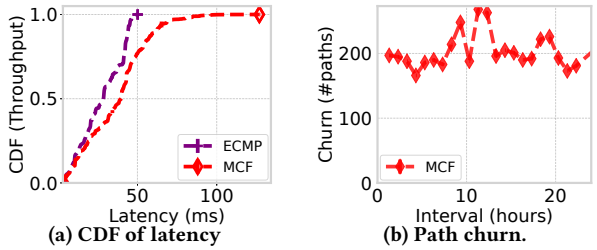
3 CENTRALIZED TE

Next we will see how the same programming interface used to model conventional distributed TE algorithms, such as ECMP, can also be used to implement centralized algorithms, which have risen in popularity in recent years as more and more networks adopt SDN-style management [18, 21].

To illustrate, consider two representative algorithms, which would typically only be implemented on a centralized architecture: (1) MCF: an optimal approach that minimizes maximum link congestion [31] and (2) KSP + MCF: a simplified version of SWAN [18] which load balances traffic over k-shortest paths. We can implement the former algorithm in YATES by writing a single module that formulates the routing problem as a multi-commodity flow (MCF) problem, encodes it as a linear program (LP) using YATES libraries, and uses an off-the-shelf LP-solver, such as Gurobi [15], to compute the optimal routing scheme. More interestingly, we can implement the latter by combining a module that computes k-shortest paths (KSP), with another that optimizes the probability distribution over the paths computed by KSP using

```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm
(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  let empty_scheme : scheme = SrcDstMap.empty in
  let empty_demands : demands = SrcDstMap.empty in
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands
(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in
(* Helper: simulate one traffic matrix *)
let simulate_step (d:demands) : unit =
  (* Compute current routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
  ...
(* Simulate all traffic matrices *)
let simulate_all (ds:demands list) : unit =
  List.iter ~f:simulate_step ds
```

Figure 5: Pseudocode for simplified SWAN (KSP + MCF).



(a) CDF of latency **(b) Path churn.**

Figure 6: Overheads of congestion-optimal MCF-based TE.

the SemiMCF module, which solves the multi-commodity flow problem restricted to just those paths. Fig. 5 gives pseudocode showing how this behavior can be obtained simply by composing components of YATES modules. Hence, YATES enables specifying complex TE algorithms as the modular composition of simpler components. As demands evolve over time or as failures occur, these modules would be repeatedly invoked to meet the current operating conditions.

Fig. 4 shows the improvement in performance when using a centralized approach like KSP+MCF in contrast to conventional approaches. Moreover, YATES makes it easy to examine a large set of additional performance metrics such as latency, solver time, churn etc. To illustrate, Fig. 6a shows that MCF, which is optimal in terms of minimizing maximum congestion, may select paths with higher latency. Further, the routing scheme generated by MCF is not stable under changing demands—i.e., the set of paths used for forwarding traffic can change significantly even if demands change slightly. This leads to churn in network state as shown in Fig. 6b. Thus, YATES enables studying trade-offs between TE systems on a level playing field.

```
type failure = EdgeSet.t
val recovery :
      (* Initial routing scheme and topology *)
      scheme -> topology ->
      (* Failure scenario *)
      failure ->
      (* Post-recovery routing scheme *)
      demands -> scheme
```

Figure 7: Generic failure recovery type.

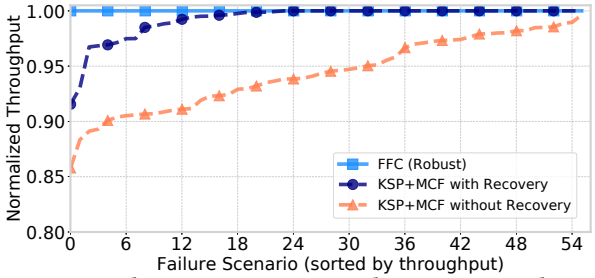


Figure 8: Combining TE systems with recovery mechanism to improve robustness. (Path budget = 4)

4 ROBUST TE

TE systems are expected to gracefully handle unexpected failures, such as fiber cuts or router malfunctions. One mechanism that is widely deployed in practice is Fast Reroute (FRR) [3], which protects MPLS-based distributed TE systems against failures. FRR works by rerouting traffic over a precomputed backup path when a failure affects the primary routing path. Using the high-level programming interface exposed by Yates, it is straightforward to implement failure recovery in a generic way by implementing a function with the type shown in Fig. 7.

Such a function can be composed with an existing TE system, perhaps one designed without built-in fault-tolerance, to make it more robust. For example, Yates comes equipped with a generic failure recovery method called *normalization recovery*, which normalizes the path distributions in the routing scheme after removing all paths affected by the failure.

To demonstrate this, let us take KSP+MCF and augment it with normalization recovery to improve its robustness. As an example of a robust-by-design TE system, we also implement a prototype of FFC [27], a TE system that is designed to be resilient to a configurable number of failures (a single link in our implementation). Fig. 8 shows the total throughput (normalized to total demand) achieved by these TE systems in an experiment where we systematically test every possible single link failure scenario. As expected, FFC uses a diverse set of paths which can tolerate any single link failure, and thus it always achieves throughput of 1 while KSP+MCF's throughput under failures improves significantly with normalization recovery.

5 DESIGN AND IMPLEMENTATION

So far, we have seen how Yates's high-level and modular programming interface makes it easy to implement various TE algorithms in a unified setting. This section explores the design and implementation of Yates in further detail. Yates provides three main tools for evaluating TE approaches: (i) a simulator that enables quantitative comparison of approaches in a common framework under a variety of scenarios (§5.1), (ii) a set of tools for generating topologies, demands, and predictions using multiple statistical models (§5.2), and (iii) a backend for deploying TE approaches in an SDN-enabled network, allowing for empirical validation (§5.3).

Our implementation comprises approximately 12k lines of C, C++, and OCaml code. We have made all Yates code publicly available on GitHub under an open-source license [43]. To evaluate the ease of prototyping TE systems with Yates, we have implemented 17 different TE systems ranging from simple approaches like OSPF, ECMP etc. to more complex ones like MCF [31], FFC [27], SMORE [25] etc.

5.1 Simulator

The Yates simulator models how the state of the network evolves in response to changing demands and failures. It has four required input parameters: (i) the network topology, (ii) a list of *actual* traffic matrices (TMs), (iii) corresponding *predicted* TMs, and (iv) a list of algorithms to evaluate. The simulator also supports a number of parameters for modeling different scenarios such as failures and traffic bursts.

The simulator iterates over the sequence of algorithms and TMs. In each iteration, it uses the selected algorithm and predicted TM to compute the routing scheme, and the corresponding actual TM to simulate traffic. To maintain accuracy of aggregate statistics of interest, such as link utilizations, while ensuring scalability of the simulator, Yates relies on the *fluid model* [30]. To enforce link capacities, Yates associates each link with a queue. At each time step, sources push traffic onto the queue associated with the appropriate ingress link. Likewise, each link forwards traffic to next hops for each flow that it is handling by pushing traffic to the queue for the next link. The simulator allocates each flow its *max-min fair share* of bandwidth at each link, and records any excess traffic as dropped. Based on the specified failure model, the simulator can fail certain network elements at run time, and notify the TE algorithm of such changes in topology to allow it to react by updating the routing scheme.

Runtime Parameters. The performance of a TE algorithm is affected by a number of runtime and deployment conditions. Yates's simulator provides several parameters that can be systematically tuned to model these conditions:

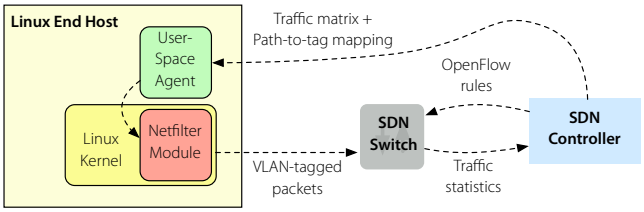


Figure 9: SDN backend architecture.

- *Budget:* Network devices are often constrained in terms of the number of forwarding rules. To evaluate the impact of such resource constraints, YATES allows imposing specified limits on the number of paths that an algorithm can use per source-destination pair.
- *Failure Model:* To measure robustness of TE algorithms, YATES allows simulating failure scenarios based on different approaches such as failing: (i) random links, (ii) shared risk link groups (SRLG), and (iii) links based on a empirically measured probability distribution, and so on.
- *Prediction Error:* Some TE approaches that use predicted TMs to compute routing schemes, such as MCF, are sensitive to inaccuracies in TM prediction. In addition to error margin [1], YATES provides other realistic ways to model prediction error, as described later.
- *Path-split quantization:* Current routers support flexible splitting of traffic over multiple paths by mapping next-hop groups [21] to paths in proportion to the splitting ratio. As the number of next-hop groups is limited, path splitting ratio cannot be arbitrary, but should be a multiple of the path-split quantum supported by the router.

5.2 Workload Generation

To evaluate TE approaches under varying workloads, YATES provides tools for modeling topologies, generating demands, and predicting demands using a variety of statistical models.

Topologies. YATES accepts topologies expressed in standard graph formats, which can be easily generated using a variety of tools—e.g., NetworkX [16]. In our experiments, we have used a large public set of WAN topologies provided by Internet Topology Zoo [24] and Rocketfuel [37] as well as some proprietary topologies shared by ISPs and content providers.

Demands. YATES implements the gravity model [36] to generate synthetic but realistic TMs. This model ascribes to each node i a non-negative weight, w_i, and posits that the amount of traffic flowing from i to j is proportional to the product $w_i \cdot w_j$ for all pairs i, j. YATES computes w_i based on empirically determined TMs from real WANs.

As demands vary over time, YATES uses two techniques to model these variations depending on timescale. For diurnal and weekly patterns, it introduces noise to the Fourier coefficients of the time-series of total flow measurements. For hourly timescales, YATES samples TMs from a Markov chain, whose stationary distribution is the gravity model based TM, using Metropolis-Hastings algorithm. This algorithm updates w_i in consecutive time steps by randomly sampling an adjusted value for w_i based on a proposal distribution. YATES defines a proposal distribution for additive adjustment (Δw) that incorporates gradual variation over time ($\mathcal{N}(0, w^2/4)$ with probability 0.99) accompanied by rare discrete jumps ($\mathcal{U}([-w, -0.8w] \cup [0.8w, w])$ with probability 0.01).

Flash Bursts: To model unexpected bursts of traffic, YATES supports inducing spikes in demand to a sink followed by a heavy-tailed decrease back to the stationary distribution. The burst amount parameters, the half-life of the decreasing tail, and the selection of sink nodes are all configurable.

Prediction. YATES provides two kinds of algorithms for predicting the next TM in a sequence of TMs: (i) machine learning methods including linear regression, lasso/ridge regression, logistic regression, random forest prediction etc. and (ii) algebraic methods including FFT fit and polynomial fit, which are based on approximating the time series with a Fourier-sparse or low-degree-polynomial, respectively.

Prediction error: To model inaccuracies in TM estimation, YATES perturbs each w_i in the gravity model by multiplying it with $\mathcal{U}(\{1-\epsilon, 1+\epsilon\})$ and then generating the TMs using the perturbed weights. Predicting TMs in this way allows direct modulation of the error parameter (ϵ) in order to evaluate robustness of TE algorithms to prediction inaccuracy.

5.3 SDN Backend

YATES enables easy deployment of TE approaches by providing a OpenFlow-based SDN backend. We use SDN to demonstrate the fact that the same implementation of TE systems can be used for simulation within the framework as well as to control real networks. YATES's modular design enables us to replace the default backend with other mechanisms to control network devices. The default backend generates traffic based on specified TMs and routes packets using a path identifier, similar to source routing. Fig. 9 shows the architecture of the SDN backend, which has four main components: (i) an SDN controller, (ii) OpenFlow switches, (iii) an end-host agent in user space, and (iv) an end-host kernel module.

SDN controller. The controller performs the following functions: (i) compute the forwarding paths based on the TE algorithm and assign a physical network path identifier to each forwarding path, (ii) install appropriate forwarding rules, (iii) send the path-to-identifier mappings to each of the user-space end-host agents in the network, and (iv) periodically gather traffic statistics from switches.

OpenFlow-enabled switch. The switches route traffic by examining the identifier tag in a packet, and forwarding the packet out on the port decided by the matching flow rule. Our prototype uses VLAN tags to store the identifier, although

we could have also used MPLS labels. Switch counters collect statistics about the amount of traffic on each link.

End-host user-space agent. The agent serves as an intermediary between the controller and the end-host kernel module. It listens on a designated port for messages from the controller, containing path-to-identifier mappings and a periodically updated routing scheme. The agent communicates this information to the kernel module through `/proc`.

End-host kernel module. The kernel module intercepts outgoing packets using `netfilter` hooks and tags each packet with the appropriate path identifier. It ensures that packets in the same flow are sent along the same paths by tracking flows in a hash table. Flows are evicted lazily from the hash table based on an idle timeout. For randomized schemes, it assigns path to a new flow by sampling from the probability distribution over paths specified by the routing scheme.

Calibration. We used this backend to emulate Abilene [19] on a hardware testbed. We generated traffic using measured TMs [28] and benchmarked the simulator's accuracy using detailed measurements on the testbed. Overall, we found the simulator results to be consistent with the measured results on the testbed. We also calibrated Yates using data collected from the network of a major content-provider [25].

5.4 Limitations

Although Yates provides a number of powerful algorithms and tools for modeling and evaluating TE algorithms, it also has several important limitations. One such limitation is the choice to use the fluid model. While Yates attempts to achieve high accuracy for macroscopic performance metrics like throughput, it is not designed to reason about precise per-packet behavior in the network. Consequently, Yates is not ideal for micro-benchmarking such as evaluating TCP congestion control algorithms or studying the effects of packet-reordering, queueing delays etc. Similarly, the simulator cannot capture the effects of hashing algorithms used to map traffic onto paths. Many alternatives exist for such measurements [26, 29, 40], and they provide complementary features to Yates. Moreover, using the SDN backend, Yates could be deployed in a hardware testbed or on top of emulators or virtualized environments such as Mininet, EmuLab etc.

Yates is designed with a focus on centralized TE algorithms, and evaluating de-centralized ones requires a different infrastructure. Although it is possible to approximate them in Yates, as demonstrated in §2, it comes with drawbacks—e.g, no ability to reason about convergence of distributed protocols or performance during transient states.

6 RELATED WORK

TE has been an area of extensive research for decades [8, 9, 18, 21, 22, 27, 38, 42, 44]. The traditional approach is to carefully tune link weights in distributed routing protocols, such as OSPF and ECMP, so they compute a near-optimal set of forwarding paths [10, 11]. The optimal approach is based on solving the MCF problem with LP techniques [15], or relaxing optimality and using approximation algorithms [2, 13, 33].

TE approaches often try to optimize certain metrics such as congestion, throughput [18], latency, robustness [38], fairness [21] etc. Several recent systems have exploited the global visibility offered by SDN to distribute traffic across paths in near-optimal ways [18, 21]. Another line of work has explored robust approaches in the presence of limited network information [1, 34, 39]. Many of these projects claim impressive performance results, but the operational conditions that were used to evaluate them are not always easy to replicate. Yates allows them all to be put on a level playing field.

Numerous simulators and emulators have been developed over the years [4, 5, 17, 20, 23, 26, 29, 32, 35, 40]. While these are powerful research tools, they are difficult to scale for WANs. Recently, robust validation [6] techniques have been used for worst-case analysis of TE systems. To get more accurate behavior, operators might use a live testbed, such as Emulab [41], or PlanetLab [7] but building realistic testbeds is difficult. REPETITA [14] is closest to our work. Like Yates, it provides a framework for experimenting with TE algorithms.

Yates abstracts away from packet-level interactions and focuses on macroscopic performance of WANs. Yates is designed specifically to evaluate TE algorithms, and to enable rapid deployment and validation on SDN networks. Additionally, Yates can be extended with verification tools like ProbNetKAT [12] to reason about network properties.

7 CONCLUSION

WAN operators face competing requirements, such as performance versus robustness, and a wide range of operational conditions when implementing TE solutions. Comparing different strategies is a difficult task, compounded by the fact that many TE solutions are tailored to specific assumptions about network behavior. We believe that innovation has been hindered owing to lack of realistic and credible ways to compare systems through careful experiments.

This paper presents Yates—a domain-specific, open-source TE framework. Yates provides a unified set of high-level and modular abstractions to allow users to quickly implement, evaluate, and deploy different TE algorithms. The research community has recognized the importance of sharing data and artifacts, as can be seen by recent initiatives such as artifact evaluation. Yates is a tool designed to facilitate such efforts and encourage a scientific approach to TE research.

Acknowledgments. We wish to thank the SOSR reviewers for their helpful feedback. This work is supported by NSF grant CCF-1637532 and ONR grant N00014-15-1-2177.

REFERENCES

[1] David Applegate and Edith Cohen. 2003. Making Intra-domain Routing Robust to Changing and Uncertain Traffic Demands: Understanding Fundamental Tradeoffs. In *ACM SIGCOMM*.

[2] Sanjeev Arora, Elad Hazan, and Satyen Kale. 2012. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Transactions on Computers* 8, 1 (2012), 121–164.

[3] Alia Atlas, George Swallow, and Ping Pan. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC 4090. (May 2005).

[4] Lee Breslau, Deborah Estrin, Haobo Yu, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, et al. 2000. Advances in Network Simulation. *Computer* 5 (2000), 59–67.

[5] Xinjie Chang. 1999. Network simulations with OPNET. In *ACM Conference on Winter Simulation*.

[6] Yiyang Chang, Sanjay Rao, and Mohit Tawarmalani. 2017. Robust Validation of Network Designs under Uncertain Demands and Failures. In *USENIX NSDI*.

[7] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: An Overlay Testbed for Broad-coverage Services. In *ACM SIGCOMM CCR*, Vol. 33. 3–12.

[8] Emilie Danna, Subhasree Mandal, and Arjun Singh. 2012. A Practical Algorithm for Balancing the Max-Min Fairness and Throughput Objectives in Traffic Engineering.. In *IEEE INFOCOM*.

[9] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, and Jennifer Rexford. 2000. NetScope: Traffic engineering for IP networks. *IEEE Network* 14, 2 (2000), 11–19.

[10] B. Fortz, J. Rexford, and M. Thorup. 2002. Traffic Engineering with Traditional IP Routing Protocols. *IEEE Communications Magazine* 40, 10 (2002).

[11] B. Fortz and M. Thorup. 2000. Internet Traffic Engineering by Optimizing OSPF Weights. In *IEEE INFOCOM*.

[12] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. 2016. Probabilistic NetKAT. In *ESOP*.

[13] Naveen Garg and Jochen Könemann. 2007. Faster and Simpler Algorithms for Multicommodity Flow and Other Fractional Packing Problems. *SIAM J. Comput.* 37, 2 (2007), 630–652.

[14] Steven Gay, Pierre Schaus, and Stefano Vissicchio. 2017. REPETITA: Repeatable Experiments for Performance Evaluation of Traffic-Engineering Algorithms. *CoRR* abs/1710.08665 (2017).

[15] Gurobi Optimization, Inc. 2016. Gurobi Optimizer. http://www.gurobi.com. (2016).

[16] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *SciPy*.

[17] Thomas R Henderson, Mathieu Lacage, George F Riley, C Dowell, and J Kopena. 2008. Network Simulations with the ns-3 Simulator. *SIGCOMM demonstration* 14 (2008).

[18] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving High Utilization with Software-Driven WAN. In *ACM SIGCOMM*.

[19] Internet2. 2003. Historical Abilene Data. http://noc.net.internet2.edu/i2network/live-network-status/historical-abilene-data.html. (2003).

[20] Teerawat Issariyakul and Ekram Hossain. 2011. *Introduction to Network Simulator NS2*. Springer Science & Business Media.

[21] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, Jonathan Zolla, Urs Hölzle, Stephen Stuart, and Amin Vahdat. 2013. B4: Experience with a Globally Deployed Software Defined WAN. In *ACM SIGCOMM*.

[22] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. 2005. Walking the Tightrope: Responsive Yet Stable Traffic Engineering. In *ACM SIGCOMM*.

[23] Srinivasan Keshav. 1988. *REAL: A network simulator*. University of California.

[24] Simon Knight, Hung X Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. 2011. The Internet Topology Zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[25] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-Oblivious Traffic Engineering: The Road Not Taken. In *USENIX NSDI*.

[26] Bob Lantz, Brandon Heller, and Nick McKeown. 2010. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *ACM HotNets*.

[27] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. 2014. Traffic Engineering with Forward Fault Correction. In *ACM SIGCOMM*.

[28] Matthew Roughan. 2017. Internet Traffic Matrices. http://www.maths.adelaide.edu.au/matthew.roughan/project/traffic_matrix. (2017).

[29] Steven McCanne and Sally Floyd. 1995. NS network simulator. (1995).

[30] Vishal Misra, Wei-Bo Gong, and Don Towsley. 2000. Fluid-based analysis of a network of AQM routers supporting TCP flows with an application to RED. In *ACM SIGCOMM CCR*, Vol. 30. 151–160.

[31] Debasis Mitra and KG Ramakrishnan. 1999. A Case Study of Multiservice, Multipriority Traffic Engineering Design for Data Networks. In *IEEE GLOBECOM*.

[32] Michal Piorkowski, Maxim Raya, A Lezama Lugo, Panagiotis Papadimitratos, Matthias Grossglauser, and J-P Hubaux. 2008. TraNS: Realistic Joint Traffic and Network Simulator for VANETs. *ACM SIGMOBILE MC2R* 12, 1 (2008).

[33] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. 1995. Fast Approximation Algorithms for Fractional Packing and Covering Problems. *Mathematics of Operations Research* 20, 2 (April 1995), 257–301.

[34] Harald Räcke. 2008. Optimal Hierarchical Decompositions for Congestion Minimization in Networks. In *ACM STOC*.

[35] George F Riley. 2003. The Georgia Tech Network Simulator. In *ACM SIGCOMM MoMeTools*.

[36] Matthew Roughan, Albert Greenberg, Charles Kalmanek, Michael Rumsewicz, Jennifer Yates, and Yin Zhang. 2002. Experience in Measuring Backbone Traffic Variability: Models, Metrics, Measurements and Meaning. In *ACM SIGCOMM Workshop on Internet measurment*.

[37] Neil Spring, Ratul Mahajan, and David Wetherall. 2002. Measuring ISP topologies with Rocketfuel. In *ACM SIGCOMM CCR*, Vol. 32. 133–145.

[38] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. 2011. Network Architecture for Joint Failure Recovery and Traffic Engineering. In *ACM SIGMETRICS*.

[39] L. Valiant. 1982. A Scheme for Fast Parallel Communication. *SIAM J. Comput.* 11, 2 (1982), 350–361.

[40] András Varga et al. 2001. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference*.

[41] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. 2002. An Integrated Experimental Environment for Distributed Systems and Networks. In *USENIX OSDI*.

[42] Xipeng Xiao, Alan Hannan, Brook Bailey, and Lionel M Ni. 2000. Traffic Engineering with MPLS in the Internet. *IEEE Network* 14, 2 (2000).

[43] YATES authors. 2018. YATES Implementation. http://github.com/cornell-netlab/yates. (2018).

[44] Baobao Zhang, Jun Bi, and Jianping Wu. 2013. Making Intra-domain Traffic Engineering Resistant to Failures. In *ACM SIGCOMM*.