

# REPLEX — Dynamic Traffic Engineering Based on Wardrop Routing Policies

Simon Fischer<sup>\*</sup>  
RWTH Aachen, Germany  
fischer@cs.rwth-aachen.de

Nils Kammenhuber<sup>†</sup>  
TU München, Germany  
hirvi@net.in.tum.de

Anja Feldmann  
Deutsche Telekom Laboratories  
Anja.Feldmann@telekom.de

## ABSTRACT

One major challenge in communication networks is the problem of dynamically distributing load in the presence of bursty and hard to predict changes in traffic demands. Current traffic engineering operates on time scales of several hours which is too slow to react to phenomena like flash crowds or BGP reroutes. One possible solution is to use load sensitive routing. Yet, interacting routing decisions at short time scales can lead to oscillations, which has prevented load sensitive routing from being deployed since the early experiences in Arpanet.

However, recent theoretical results have devised a game theoretical re-routing policy that provably avoids such oscillation and in addition can be shown to converge quickly. In this paper we present REPLEX, a distributed dynamic traffic engineering algorithm based on this policy. Exploiting the fact that most underlying routing protocols support multiple equal-cost routes to a destination, it dynamically changes the proportion of traffic that is routed along each path. These proportions are carefully adapted utilising information from periodic measurements and, optionally, information exchanged between the routers about the traffic condition along the path.

We evaluate the algorithm via simulations employing traffic loads that mimic actual Web traffic, i. e., bursty TCP traffic, and whose characteristics are consistent with self-similarity. The simulations quickly converge and do not exhibit significant oscillations on both artificial as well as real topologies, as can be expected from the theoretical results.

## Categories and Subject Descriptors

C.2.2 [Computer-Communication Networks]: Network Protocols—Routing protocols

## General Terms

design, experimentation, measurement, performance, reliability

<sup>\*</sup>Supported by DFG-Schwerpunkt 1126 grant Vo889/1-3.

<sup>†</sup>Supported by DFG-Schwerpunkt 1126 grant Fe570/1-3.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

CoNext'06, December 4–7, 2006, Lisboa, Portugal  
Copyright 2006 ACM 1-59593-456-1/06/0012 ...\$5.00.

## Keywords

Dynamic routing, real-time adaptive multipath routing, load balancing, Wardrop equilibria, convergence time, traffic engineering

## 1. INTRODUCTION

Internet Service Providers (ISPs) want to use their network infrastructure as efficiently as possible to maximise their revenues. This encompasses the task of designing the network and optimising the routing system which is responsible for delivering the traffic entering the network to its destination. In the context of IP networks, the latter task is referred to as *traffic engineering* (TE). TE adopts the parameters of the routing system to the imposed traffic demands, in order to achieve performance gains.

Over time, various TE methods have been proposed. Most popular among operators are approaches that either utilise MPLS tunnels [47] or solutions with optimised IGP link weights [45, 16, 41, 22, 18, 17, 12]. However, the disadvantage of these approaches is that they are quasi-static: The optimal routing is calculated offline for a predicted or previously measured traffic demand matrix. Obviously such schemata are not able to react to unpredicted traffic changes or network events. To arrange for unforeseen sudden traffic shift, network providers account for single link failures and some traffic shifts in their optimisations [45, 18, 1]. In addition they often combine traffic engineering techniques with overprovisioning of their network infrastructure, or purely rely on overprovisioning.

Recently, TeXCP [21] and MATE [9] have been proposed as promising dynamic TE solutions. These systems take advantage of alternative paths in a network (e. g., MPLS multipaths). They optimise network utilisation by adjusting the distribution of traffic among the paths with the same ingress/egress nodes. These systems are not routing protocols per se; they rather operate on top of an existing MPLS infrastructure. Accordingly they are referred to as *traffic engineering protocols* [21].

This paper presents a new traffic engineering protocol, REPLEX, which is applicable in a broader context than TeXCP or MATE. In contrast to TeXCP and MATE, it is not restricted to MPLS-like solutions but can be deployed on top of virtually any routing protocol. Moreover communication between the routers is favourable, but not necessary.

Our protocol is derived from an algorithm that can be proven to be stable and converge quickly in a game-theoretic model. The algorithm assumes that the only action that a router can perform is to change the traffic distribution among a set of equal-cost<sup>1</sup> routes between an ingress/egress pair. Similar to TeXCP or MATE, these

<sup>1</sup>Throughout this paper, the term *cost* refers to quasi-static costs as defined by an underlying routing protocol, e. g., OSPF link weights. It does not encompass costs that change with traffic conditions.

paths are provided by an existing routing architecture.

Our algorithm specifies how a router should react to changes in its traffic load, be they externally or the effect of another routers' decision. In order to not reorder packets within a flow, which is known to cause bad TCP performance [26], traffic splitting is done on a flow-by-flow basis utilising the standard hashing technique [7]. To help the router with its decision about how to distribute the traffic among the path, the router can gather measurement data about the path itself, or, in addition, periodically exchange information with other routers. While experiments indicate that this information is helpful, in cases where such communication is not desired, e. g., in an interdomain context, the algorithm still yields improvements. In general, the communication overhead is low, as we use a distance vector-like approach for distributing this information. The method is therefore scalable even for large networks.

Our proposed system features a number of novel properties. Theoretical analyses in a game-theoretic setting promise a quick convergence speed and no oscillations. Our simulations support these expectations under realistic conditions. Contrary to almost all other papers proposing dynamic routing solutions, we impose a realistic workload whose characteristics are consistent with self-similar traffic, and which is thus difficult to handle due to the resulting traffic bursts. The traffic demands are the results of heavy-tailed arrival processes of Web-like downloads with heavy-tailed distributions of the number of bytes and inter-download times. The Web-like workload moreover implies that we do neither ignore the dynamics imposed by TCP's congestion control, nor its inherent feedback to congestion in the network which interacts with any small-time scale traffic engineering. The simulations furthermore confirm that the communication overhead is low and that the system works even if no communication is possible.

One of the convenient properties of the proposed system is that it is generic with regard to the underlying routing architecture that provides the routing alternatives, may they be OSPF, IS-IS, MPLS, etc. Here, an interesting question is if the number of offered alternative paths is sufficient to enable the algorithm to do its job, or how many alternative paths are needed. Our experiments using topologies provided by Rocketfuel [37] indicate that the available number of equal-cost paths when using hop-count as distance metric is sufficient.

The remainder of this paper is structured as follows. We discuss related work in Section 2. The underlying game theoretic algorithm and its analysis together with the derivation of the proposed algorithm is presented in Section 3. The setup for our network simulations featuring realistic workload is outlined in Section 4. Section 5 presents simulation results for simple topologies, which are used to understand the principle behaviour of the algorithm as well as to determine suitable parameter values. Section 6 analyses the algorithms performance on topologies of actual autonomous systems as determined by Rocketfuel. We summarise our contributions and discuss future work in Section 7.

## 2. RELATED WORK

A great variety of publications exist on traffic engineering and related areas [16, 47, 17, 45] such as traffic matrix estimation [34, 49, 40, 50]. Most of the TE methods frequently used within the network operator community are offline methods: Network traffic intensities are measured over some period of time (usually a few hours); then these measurements are collected at a central point, where a new routing (e. g., readjusted OSPF weights; [17, 18, 41] and many others) is calculated. This new routing is then used the

network, and the network remains in the new state—unless a link or a router fails, or the TE is re-run.

Such TE schemes are used by quite a number of operational networks, since they achieve good performance gains. By running TE mechanisms multiple times per week or even per day, Internet providers can react to long-term changes in traffic demands (e. g., day-of-week effects and to some extent circadian effects). Hao and Ito even propose a mechanism for dynamically adjusting OSPF weights [19]. However, changing weights too often increases the number of times that the intradomain routing protocol has to converge, and has an impact on the interdomain routing. During the interdomain routing convergence time, albeit in the order of seconds or milliseconds, routing loops and thus massive service degradation can occur. Moreover changing IGP weights can result in changes to the BGP routing [39] and thus affect other providers and potentially be harmful to the global BGP stability.

Obviously, offline methods that are run once every few hours cannot react to sudden traffic changes in real time. Such changes can, e. g., be caused by BGP reroutes, flash crowds, malicious behaviour, and to some extent by diurnal traffic variations. Furthermore, while current TE can deal with network failures by pre-computing alternative routings, it usually does so only for a limited set of failures (e. g., explicitly via MPLS backup paths or implicitly [18, 1]). It thus may fail to anticipate certain failures, which can lead to bad performance even when the network actually provides enough capacity. Even a guaranteed worst-case behaviour [45] still leaves room for improvement.

In the past, various methods for online traffic engineering, in other words, routing protocols that react to changes in traffic demands, have been proposed. The earliest approaches to such load-adaptive routing were used in the Arpanet [23]. However, those protocols have a strong tendency to oscillate; this is due to interactions between the decisions made by the various routers. In today's Internet, one can presume an even higher danger of oscillations when deploying a load-adaptive routing protocol: A large share [8] of today's traffic consists of TCP connections, which themselves use end-to-end feedback loops in order to avoid network congestion [30]. Thus a routing protocol reacting to load changes may affect a large number of TCP connections, which results in interactions between the feedback loop of the routing protocol with the one used by the affected TCP connections. Hence, by reacting to changes in the traffic demands, the traffic demands themselves may change.

With all these issues in mind, network operators are sceptical about employing load-adaptive routing protocols. Instead, they use traditional offline TE techniques, combined with vast overprovisioning of their network backbone infrastructure. While this “brute-force” approach can indeed protect against potentially disrupting changes in traffic patterns, it brings about the question whether a currently overprovisioned network backbone can be used in a more efficient way, thus allowing the operator to increase its revenues without the need for costly upgrades to its hardware.

The reluctance of network operators to employ load-adaptive routing in their network is amplified by the fact that most load-adaptive schemes that have been proposed in theory (for an overview, we refer to [36]) require them to entirely replace their current routing architecture with a new load-adaptive one. To remedy this aspect, approaches that allow automatic online traffic engineering on top of an existing traditional routing infrastructure have been proposed. Among these are MATE [9] and TeXCP [21]. Both systems represent approaches similar to ours, in that they are not routing protocols, but traffic engineering protocols [21]. They split traffic amongst multiple paths that an underlying routing architecture

has established between the same pair of edge routers (the authors suggest MPLS tunnels), and adjust the splitting ratio dynamically depending on the traffic. In contrast to TeXCP and MATE, the system we propose is not restricted to path-centred (i. e., MPLS-like) scenarios. Not only is there a large number of ISPs who do not use MPLS, but also there are known issues with MPLS paths across AS boundaries [29]; therefore these TE protocols are not suited as interdomain TE protocols. Moreover our approach allows greater scalability for large networks, since the communication cost incurred at each router is only linear with the number of interfaces times the number of destination prefixes, while for MPLS-like solutions it is quadratic with the number of edge routers. Finally, an evaluation of their systems’ behaviour under realistic traffic load, containing TCP feedback loops and bursts caused by the self-similar nature of Internet traffic, has not been performed yet.

The Traffic Engineering extensions for OSPF routing (OSPF-TE [22]) allow the routers to inform the network not only about availability and costs for each link, but also provide a framework to broadcast traffic conditions for each link. Considering that OSPF is a link-state protocol, one can expect a sizable increase in communication due to periodically broadcasting traffic conditions for each single link. The system that we propose is expected to scale better than OSPF-TE, since it aggregates information for paths in a distance-vector like fashion. Moreover our approach does not prescribe the operator to use OSPF; rather it can even be deployed on top of routing protocols such as BGP/MIRO [48] across AS boundaries for the purpose of interdomain traffic engineering.

The theoretical background of our algorithm is described in [13, 14, 15]. Therein, a very similar policy is analysed in what has become known as the Wardrop model [4, 35, 46]. Ragnathan et al. analyse the performance of STARA routing protocols and propose various Wardrop-based improvements, most notably PSTARA [31, 32, 36]. The authors focus on mobile networks where network sizes are small; realistic traffic workload and TCP issues are not considered. Liu and Reddy analyse a similar adaptive mechanism [27] in a multihoming scenario [27].

Adaptive routing policies have also been studied from the perspective of online learning, where one aims at minimising the *regret* which is defined as the difference between a user’s average latency over time and the latency of the best path in hindsight (see, e. g., [2, 6]). Analyses of the convergence time for selfish load balancing problems in discrete models have been performed mainly for simple networks consisting of parallel links [10, 5].

### 3. METHODOLOGY

In this section we introduce our load-adaptive multipath rerouting algorithm. We start from a simple rerouting policy in a game theoretical model which can be proven to quickly converge to a stable state. This model assumes that a set of agents is located at the ingress of the network, each of which manages the path for a small amount of traffic headed towards some destination. Each agent has full control over which path its traffic takes and uses this control to minimise the latency that its traffic will sustain.

Subsequently, we adopt this algorithm from an artificial setting to an algorithm that is deployable in networks with a traditional IP routing infrastructure. We assume that path alternatives exist (such as in OSPF equal-cost multipath), but that the router may not necessarily have control over the path the packets will take beyond the current router. The optimisation TE goal may also differ from minimising latencies.

#### 3.1 A Game Theoretic View

The algorithm we present in this section is built upon the founda-

tion of a well-known game theoretic model, the Wardrop model. A rigorous analysis in this model proves convergence even with stale information and yields polynomial upper bounds on the time to reach approximate equilibria [13, 15]. Note, that the algorithm presented in this section cannot be directly used as a routing policy in a real-world IP network. Rather, it serves as a starting point for the scheme we devise in Section 3.2.

In the Wardrop traffic model [46], one assumes an infinite number of selfish agents each of which wants to send an infinitesimal amount of traffic (called *flow*) through a network  $G = (V, E)$ . Each agent belongs to a *commodity*  $i$  from a set  $[k] = \{1, \dots, k\}$  specified by a source  $s_i$ , a sink  $t_i$  and a total flow demand  $d_i$  that is to be routed from  $s_i$  to  $t_i$ . The total flow demand  $d_i$  can be interpreted as the number of agents belonging to commodity  $i$ . We normalise the demands such that  $\sum_{i \in [k]} d_i = 1$ . Each agent can choose the path for their flow from a set  $\mathcal{P}_i$  of paths connecting  $s_i$  and  $t_i$ . The path an agent picks is also referred to as their *strategy*, and  $\mathcal{P}_i$  is therefore the *strategy space* of commodity  $i$ . By  $\mathcal{P} = \cup_{i \in [k]} \mathcal{P}_i$  we denote the set of all possible routing paths. An assignment of agents, or, equivalently, traffic, to paths induces a flow vector  $(f_P)_{P \in \mathcal{P}}$ . This flow assignment in turn induces latencies on the edges.

For  $e \in E$ , let  $f_e = \sum_{P \ni e} f_P$  denote the flow on edge  $e$ . Then the latency of edge  $e$  is given by the value of the respective non-decreasing *latency function*  $\ell_e(f_e)$  and the latency of a path  $P$  is then  $\ell_P(f) = \text{agg}_{e \in P} \{\ell_e(f_e)\}$  where *agg* is some aggregation function. In the Wardrop model, *agg* is typically assumed to be the sum which is appropriate if  $\ell$  denotes latency. If  $\ell$  denotes any other metric like link utilisation, *agg* can also be the maximum or some other function. In the following, when  $f$  is clear from the context, we omit  $f$  as an argument to  $\ell$ .

In picking a routing path  $P \in \mathcal{P}$ , the agents strive to minimise the latency they sustain. From the game theoretic perspective, one is usually interested in equilibrium solutions, i. e., an assignment of agents to paths such that no agent has an incentive to deviate from the path it is assigned to. This notion is formalised by the solution concept of *Nash equilibria*, also referred to as *Wardrop equilibria* in this model.

**DEFINITION 1.** *A flow vector  $f$  is at a Wardrop equilibrium if for each commodity  $i \in [k]$  and each path  $P \in \mathcal{P}_i$  with  $f_P > 0$  it holds that  $\ell_P(f) \leq \ell_{P'}(f)$  for all  $P' \in \mathcal{P}_i$ .*

Wardrop equilibria are meaningful in a competitive scenario where each agent strives to minimise their own latency selfishly. In contrast, the operator controlling an autonomous system (AS) is usually interested in obtaining a system optimal flow minimising the total or average cost  $\bar{\ell}(f) = \sum_{e \in E} f_e \ell_e(f_e)$  without any need for “competition” among its network components. Let us remark that system optimal flows are Wardrop equilibria with respect to modified latency functions. More precisely, a flow at Wardrop equilibrium for so-called *marginal cost* [4] latency functions  $h_e(x) = (x \cdot \ell_e(x)) \frac{d}{dx}$  is known to be system optimal for the original latency functions  $\ell_e$ . Note, that for linear latency functions without offset, i. e.,  $\ell_e(x) = a_e \cdot x$ , we have  $h_e(x) = 2a_e \cdot x = 2\ell_e(x)$  implying that Wardrop equilibrium and optimum coincide.

We are considering the Wardrop model in a dynamic, round-based variant. Here, agents are activated every  $T$  seconds and are then allowed to change their path simultaneously. An interesting question is then whether agents can distributedly and jointly learn a Wardrop equilibrium quickly. The main problem here is to avoid oscillation. To illustrate this, consider the natural policy where agents migrate to a path with minimal latency whenever they are activated. Such policies are generally referred to as *best response* policies. However, best response policies lead to greatly increased

congestion on the optimal path at the end of a round and cause oscillation effects subsequently. Hence, our rerouting policy must be more careful. The so-called  $(\alpha\text{-}\beta)$ -exploration-replication policy presented in [13] is designed to avoid oscillation and finds approximate equilibria quickly. This policy is simple, easy to implement, and requires only local knowledge.

Informally, our policy as performed by all agents in parallel, can be described as follows: At regular intervals an agent samples another path applying one out of two sampling techniques: For *uniform sampling*, which is executed with only a small probability, every path is sampled with uniform probability. This sampling step guarantees that every path has strictly positive sampling probability and is consequently responsible for an *exploration* of the strategy space. For the second sampling technique, *proportional sampling*, the probability of sampling a path is proportional to the fraction of agents already using it, i. e., the popularity of a path is taken as an indicator of its quality, and agents using good paths are more likely to be imitated. Proportional sampling is thus responsible for *replication* of paths with small latency and will thus “boost” successful strategies. It can indeed cause the flow on such a path to grow exponentially fast. Having sampled a path  $Q$ , the agent must eventually decide whether or not they want to migrate from their old path  $P$  to  $Q$  in a *migration* step. Again, the decision is randomised. Indeed, the migration probability is sensitive to the relative latency gain that can be achieved by switching from  $P$  to  $Q$ . More precisely, the migration probability is  $\max\{0, \frac{\ell_P - \ell_Q}{\ell_P + \alpha}\}$  where  $\alpha$  is some parameter that can be interpreted as a positive offset added to all latency functions to prevent the migration probability to become overly large if  $\ell_P$  (and also  $\ell_Q$ ) are close to zero.

More formally, our policy can be described in the following way. In every round, every agent is activated with probability  $\lambda$ . Each activated agent performs the following two steps (consider an agent in commodity  $i$ , currently assigned to path  $P \in \mathcal{P}_i$ ):

1. *Sampling*. With probability  $\beta$  perform step (a) and with probability  $1 - \beta$ , perform step (b).
  - (a) *Uniform Sampling*. Pick a path  $Q \in \mathcal{P}_i$  with probability  $1/|\mathcal{P}_i|$ .
  - (b) *Proportional Sampling*. Pick a path  $Q \in \mathcal{P}_i$  with probability  $f_Q/d_i$ .
2. *Migration*. If  $\ell_Q(f) < \ell_P(f)$ , migrate from  $P$  to  $Q$  with probability  $\frac{\ell_P - \ell_Q}{\ell_P + \alpha}$ .

The adaptive migration rule ensures that small latency gains only cause a small migration rate. This is necessary to avoid oscillation. However, the migration rate also depends on the choice of the parameter  $\lambda$  which consequently determines whether or not oscillation can occur. Its value has to respect the steepness of the latency functions. This is made precise by the following theorem which is (in a slightly stronger version) proved in [13].

**THEOREM 1.** *If the latency functions are polynomials of degree  $d$  and  $\lambda \leq c/d$  for a suitable constant  $c > 0$ , the  $(\alpha\text{-}\beta)$ -exploration-replication policy converges towards a Wardrop equilibrium if  $\beta \leq \frac{\min_{P \in \mathcal{P}} \ell_P(0) + \alpha}{L \cdot \max_{e \in E} \max_{x \in [0, \beta]} \ell_e(x)}$  where  $L$  is the maximum length of a path.*

This theorem can actually be generalised to latency functions with finite first derivative. While this ensures convergence in the long run, in order to give a bound on the time of convergence we need to define *approximate equilibria*. Let  $\bar{\ell}(f)$  denote the average latency of all paths and let  $\bar{\ell}_i(f)$  denote the average latency of paths in commodity  $i$ . We say that a flow is at a  $(\delta, \varepsilon)$ -equilibrium iff for

every commodity  $i \in [k]$ ,  $\varepsilon_i$  agents use paths with latency at least  $\bar{\ell}_i(f) + \delta \cdot \bar{\ell}(f)$  and  $\sum_{i \in [k]} \varepsilon_i \leq \varepsilon$ . It can be shown [13] that for suitable choices of parameters, the  $(\alpha\text{-}\beta)$ -exploration-replication policy reaches a  $(\delta, \varepsilon)$ -equilibrium in time

$$\mathcal{O}\left(\frac{d}{\varepsilon^2 \delta^2} \log \frac{d \max_f \bar{\ell}(f)}{\min_f \bar{\ell}(f)}\right),$$

where  $d$  is again (a generalisation of) the degree of the polynomial latency functions. We believe that these positive results are a strong indication that a load-adaptive rerouting algorithm based on this selfish rerouting policy can perform well in practice. In the following section, we develop such an algorithm.

## 3.2 Wardrop routing in an IP network

The Wardrop model is suitable when the number of agents is infinite, the agents have full control over their traffic and easy access to the current latency of the paths. Obviously, none of these is true in real-world networks such as the Internet. In the following we address all of the aforementioned issues and show that a variant of our Wardrop policy can yield a practical algorithm.

### 3.2.1 Delegation of Decisions

The first major problem is that agents normally do not have control over the path their packets take in real-world communication networks. Usually, the end points of a communication stream are located at the periphery of the network, and are typically connected to the network by only a single link, e. g., Ethernet or DSL. Rather, it is the routers that are responsible for choosing the path along which the data packets are sent. In most cases however, not even the routers can choose a path as a whole: Apart from cases where tunnels such as MPLS are being used, a router can only determine the next-hop node on the way to the destination. Moreover, in normal IP routing this decision is solely based on the destination of a packet while other header attributes (e. g., the source) are neglected. In this section we describe how agents (end hosts) delegate rerouting decisions to adjacent routers, which make partial decisions and, in turn, delegate further decisions they are unable to carry out themselves along the path.

Consider a router  $r$  lying on a path  $s \rightsquigarrow r \rightsquigarrow t$  from  $s$  to  $t$ . There may exist several paths from  $r$  to  $t$ . Applying our Wardrop rerouting policy, router  $r$  aims at distributing the traffic from  $s$  to  $t$  evenly among these paths. However,  $r$  does not know about all of these paths, but merely knows a set of possible next hops of routes for destination  $t$  denoted by  $N(r, t)$ . In practice, these routes can be manually configured, or they are obtained from an underlying routing protocol such as OSPF as equal-cost routes.

In order to control the traffic balance, the router maintains a set of dynamically changeable weights  $w(r, t, v_i)$  for every target  $t$  and possible next-hop router  $v_i \in N(r, t)$ . For every target  $t$  we normalise the weights such that we obtain  $\sum_{v_i \in N(r, t)} w(r, t, v_i) = 1$ . For the time being, let us interpret  $w(r, t, v_i)$  as the exact fraction of traffic routed from  $r$  to  $t$  via  $v_i$ .

From the perspective of an agent at source  $s$ , the routing decision made at intermediate node  $r$  influences only the performance of the path section from  $r$  to  $t$  whereas the performance of the section from  $s$  to  $r$  is fixed. In order to make the routing decision,  $r$  must gather traffic information about the set of paths between  $r$  and  $t$ .

On the other hand,  $r$  cannot control which one of the possible paths a packet will take once it has been forwarded to one of the  $v_i$ 's. Thus,  $r$  cannot exploit exhaustive information about *all* possible paths. Consequently, we use only aggregated information about the possible paths from next-hop  $v_i$  to destination  $t$ . We now describe how router  $r$  can gather this information simply by perform-

ing measurements of the adjacent links  $(r, v_i)$  and exchanging messages with its peer routers  $v_i$ . This information exchange resembles a distance vector routing protocol.

Since  $r$  cannot influence a packet's path beyond any neighbour  $v_i$ , the decision whether to send the packet via link  $r \rightarrow v_i$  is based on the *expected latency*  $L(r, t, v_i)$  for the path  $r \rightarrow v_i \rightsquigarrow t$ . Router  $v_i$  can keep  $r$  informed about the expected latency of its paths  $v_i \rightsquigarrow t$  by periodically sending messages. But how can  $r$  use these values to compute  $L(\cdot)$ ? Let us consider the case that  $\text{agg}$  is the sum, and let  $\mathcal{P}(x, y)$  denote the set of paths between node  $x$  and  $y$ . Then  $r$ 's valuation  $L(r, t, v_i)$  of the next-hop node  $v_i$  for destination  $t$  is

$$L(r, t, v_i) := \ell_{rv_i} + \sum_{P \in \mathcal{P}(v_i, t)} w_P \cdot \ell_P$$

where  $w_P$  is the weight of path  $P$ , i. e., for  $P \in \mathcal{P}(u_i, u_j)$ , we have  $w_P = \prod_{i=1}^{l-1} w(u_i, t, u_{i+1})$ . The measurement value of the next-hop edge  $\ell_{rv_i}$  can be evaluated at  $r$ . Let us now specify the information that router  $v_i$  sends to  $r$ . We define

$$A(v_i, t) := \sum_{P \in \mathcal{P}(v_i, t)} w_P \cdot \ell_P .$$

We then obtain

$$\begin{aligned} A(v_i, t) &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \sum_{P \in \mathcal{P}(u_j, t)} w_P \cdot (\ell_{v_i u_j} + \ell_P) \\ &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \left( \ell_{v_i u_j} + \sum_{P \in \mathcal{P}(u_j, t)} w_P \cdot \ell_P \right) \\ &= \sum_{u_j \in N(v_i, t)} w(v_i, t, u_j) \cdot L(v_i, t, u_j) , \end{aligned}$$

where the first equality is the definition of  $\ell_P$ , the second holds because  $\ell_{v_i u_j}$  does not depend on  $P$  and the weights sum up to 1, and the third equality holds as it uses the definition of  $L(\cdot, \cdot, \cdot)$ .

To summarise, the value  $A(v_i, t)$  is computed at  $v_i$  and sent to  $r$  at regular intervals, after which  $r$  can update  $L(r, t, v_i) = \ell_{rv_i} + A(v_i, t)$ .  $A(v_i, t)$  can be interpreted as condensed information about the performance of subsequent path sections. Note that we are using the fact that  $L(v_i, t, u_j)$  is already defined for all  $v_i \in N(r, t)$  and  $u_j \in N(v_i, t)$  when computing  $L(r, t, v_i)$ . This corresponds to the information being propagated backwards along a path.

For every  $t$ , the size of such a message contains the identifier of  $t$  which is typically an IPv4 prefix of  $32 + 5$  bits plus the value of  $A(v_i, t)$  itself. Allowing 11 bits for this value, the total size of the update message is 6 bytes per best-cost destination in the routing table of  $v_i$ . For IPv6 prefixes, this value triples to  $128 + 7 + 11$  bits  $\leq 19$  bytes. Even if one assumes an extreme "worst-case" scenario where information about 200,000 IPv6 prefixes is exchanged every two seconds, the resulting communication is only 1.8 Mbytes/s. Compared to today's link capacities of 10 Gbit/s and more, this value is small ( $< 0.2\%$ ). It can be reduced even further by identifying prefixes for which no multipaths exist.

Based on the values stored in  $L(r, t, \cdot)$ , router  $r$  can adapt its weights  $w(r, t, \cdot)$  over time and distribute the traffic of this demand (from various sources to the same sink  $t$ ) as evenly close to the weights as possible. The next section explains how this split can be performed in practice.

### 3.2.2 Randomising vs. Hashing

A natural way of distributing traffic according to some weights  $w(r, v_i, t)$  is to use simple randomisation, where each weight is interpreted as a probability. However, this probabilistic routing approach has the drawback that packets headed towards the same des-

tinuation may take different paths—a bad decision. Even though the *expected* latencies along each path may be the same, the *actual* latencies can differ. Thus packets can overtake each other, which results in packet reordering at the destination. Since TCP treats packet reordering in the same manner as it treats packet losses, probabilistic routing can seriously harm TCP performance [26].

Therefore we "roll the dice" not per-packet but rather per-flow by applying the well-known technique [7] of using a hash function  $h : V \times V \mapsto [0, 1]$  mapping each packet based on its source and destination address to some value. For every destination  $t$ , we partition the interval  $[0, 1]$  into sections of size  $w(r, t, v_i)$  and label each interval with the corresponding next-hop node  $v_i$ . Whenever a packet from source  $id_s$  and destination  $id_t$  arrives, it is forwarded to the node associated with the interval containing  $h(id_s, id_t)$ .

As long as the weights are constant, we can thus be sure that no packet reordering occurs. Whenever weights are shifted, this causes a fraction of the traffic to be rerouted, possibly causing packet reordering. Hence, the time interval at which weight shifts occur should not be smaller than the time it typically takes for TCP to recover from packet losses. Note that if there is a range of hash values that are systematically more popular than others, be it by accident or due to an adversary, this causes our weight shifting procedure to decrease the corresponding weight accordingly.

### 3.2.3 Measuring Performance

In Section 3.2.1 we have assumed that every router  $r$  is able to measure  $\ell$  for all outgoing edges  $\{r, v_i\}$ . With an acceptable computational effort we can measure the following values during each time interval: the number of packets sent and dropped (due to link overload), the number of bytes sent and dropped, and the average queue length. From this information we can, e. g., compute the following two metrics:

1. The *latency* as the sum of the (constant) propagation delay of the edge plus the (variable) queueing delay. Obviously, latency values are additive along a path.
2. The *link utilisation*. Let  $b$  denote the bit rate of an interface,  $T$  the length of the measurement interval, and let  $n$  denote the number of bytes sent within this interval. Then we compute the link utilisation as  $\frac{8 \cdot n}{T \cdot b}$ . We define the utilisation of a path to be the maximum utilisation of a link in the path, i. e., the aggregation function  $\text{agg}$  is the maximum.

Different applications require optimisation of different parameters. Applications like voice over IP require small latency, accessing files requires large throughput, and a typical goal of an ISP doing traffic engineering is to minimise the maximum utilisation of an edge in the network. In this work, we focus on the latter. Let us remark that in this case, Wardrop equilibria with respect to link utilisation are also optimal from the ISP's point of view. Note, however, that our algorithm can cater for other metrics as well.

Internet traffic is known to be very bursty [28]. Therefore, traffic measurements made within a short time interval are not very reliable. To this end, instead of using the measured values directly, we use an exponential moving average (EMA). More precisely, if  $\tilde{\ell}(r, v)$  is our current metric value for link  $(r, v)$ , and we measure a value of  $\ell_{rv}$  then we update the value of  $\tilde{\ell}(r, v)$  in the next round to be  $\tilde{\ell}(r, v) \leftarrow \eta \cdot \ell_{rv} + (1 - \eta) \tilde{\ell}(r, v)$ .

## 3.3 The REPLEX Algorithm

Combining the techniques presented in the previous subsection, we now present the protocol which is run on each individual router that participates in a network optimised by our algorithm. Since

---

**Algorithm 1** Main loop of the REPLEX-algorithm

---

```
1: initialise an empty message  $M$ 
2: for every destination  $t$  in the routing table do
3:   for all next-hop nodes  $v \in N(r,t)$  do
4:     measure performance  $\ell_{rv}$ 
5:      $\tilde{\ell}(r,v) \leftarrow \eta \cdot \ell_{rv} + (1 - \eta)\tilde{\ell}(r,v)$  // calculate EMA
6:      $L(r,t,v) \leftarrow \text{agg}(\ell_{rv}, A(v,t))$ .
7:   end for
8:    $avg \leftarrow \sum_{v \in N(r,t)} w(r,t,v) \cdot L(r,t,v)$ 
9:   Append  $(t, avg)$  to  $M$ 
10: end for
11: send  $M$  to all neighbours  $v$  who store it in  $A(r,\cdot)$ 
12: call procedure ADAPTWEIGHTS (Algorithm 2)
```

---

---

**Algorithm 2** Procedure ADAPTWEIGHTS

---

```
1: for every destination  $t$  in the routing table do
2:    $w'(r,t) \leftarrow w(r,t)$ .
3:   for every pair of next-hop routers  $v_1, v_2 \in N(r,t)$  do
4:     if  $L(r,t,v_1) > L(r,t,v_2) + \epsilon$  then
5:        $\delta \leftarrow \lambda \left( (1 - \beta)w(r,t,i) + \frac{\beta}{|N(r,t)|} \right) \frac{L(r,t,v_1) - L(r,t,v_2)}{L(r,t,v_1) + \alpha}$ 
6:        $w'(r,t,v_1) \leftarrow w'(r,t,v_1) - \delta$ 
7:        $w'(r,t,v_2) \leftarrow w'(r,t,v_2) + \delta$ 
8:     end if
9:   end for
10:   set  $w(r,t) \leftarrow w'(r,t)$ .
11: end for
```

---

it is derived from the exploration-replication policy, we name our algorithm REPLEX which sounds better than EXREP. Recall that a protocol implementation has to address several tasks: Foremost, it needs to calculate the weights  $w(\cdot)$ . For this, measurements  $\ell$  on the router's link queues are necessary. These are combined with the information  $A(\cdot)$  that is received from neighbouring routers. Finally, it has to compute the information  $L(\cdot)$  which it then sends to its neighbours. In order to perform above computations, each router  $r$  maintains the following arrays::

$\tilde{\ell}(r,v)$ : The EMA of the measurements for link  $(r,v)$ .

$L(r,t,v)$ : metric value for destination  $t$  using next hop node  $v$

$A(v,t)$ : The average measurement value that next hop router  $v$  has announced for destination  $t$ . This array is updated whenever update messages from neighbouring routers are received. It is initialised with values that are neutral w. r. t.  $\text{agg}(\cdot, \cdot)$ .

$w(r,t,v)$ : Current weight of route  $r \rightarrow v \rightsquigarrow t$  for destination  $t$

In the router's main loop, which is executed every  $T$  seconds, these values are updated and sent to the neighbouring routers, as described in Section 3.2.1. Algorithm 1 describes the procedure in more detail.

At the heart of the algorithm we have the actual weight adaption procedure described in Algorithm 2. For every pair of next hop nodes, this procedure computes the migration rates (line 2) similar to the policy described in Section 3.1. The computed rates are then migrated by shifting the corresponding weights.

### 3.3.1 Parameters

In the following we describe the static parameters of our algorithm.

**update period length  $T$** : The length of the interval at which the main loop is executed.

**weigh shift factor  $\lambda$** : This parameter determines the weight shifted in one round. The ratio  $\lambda/T$  controls the convergence speed of the algorithm. Oscillation effects and congestion control feedback loops of affected TCP connections limit the maximum convergence speed we can achieve.

**EMA weight  $\eta$** : Decreasing the weight of the exponential moving average makes the algorithm less sensitive to the effects of bursty traffic whereas choosing it too small increases the time until the algorithm realises the effects of rerouting decisions.

**virtual latency offset  $\alpha$** : This virtual offset is added to all path latencies  $L(r,t,v)$  making the algorithm less sensitive to small differences when the metric is close to 0. (For a detailed discussion, see [13].)

**improvement threshold  $\epsilon$** : The parameter  $\epsilon$  can be considered a damping factor. Projected weight shifts that are smaller than  $\epsilon$  are not carried out; thus we can ensure that weights are only shifted if the change is substantial.

**exploration ratio  $\beta$** : This parameter determines the ratio between exploration and replication. If  $\beta$  is too small, currently unused paths may not be detected by our algorithm. On the other hand,  $\beta$  should not be too large since this can result in excessive growth of a flow if  $f_p$  is close to zero. (For a detailed discussion, see [13].)

Our simulations show that the performance is largely independent of the choice of  $\alpha$ ,  $\epsilon$ , and  $\beta$  as long as these are chosen within reasonable intervals. We find  $\alpha = \beta = \epsilon = 0.1$  to offer a good trade-off between the effects described above. The more sensitive parameters that influence the performance most are  $T$ ,  $\lambda$ , and  $\eta$ . Accordingly, we explore this parameter space in Section 5.1 in more detail.

## 4. SIMULATION SETUP

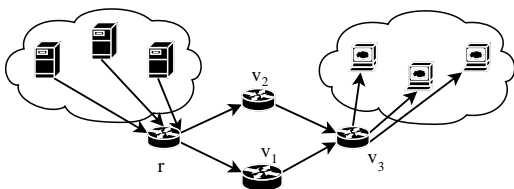
For our simulations we use the toolkit SSFNet [33] as it encompasses the features that we need: a full TCP implementation, flexible workload generators, complex topologies, and support of multiple routing protocols (including OSPF, MPLS [24], BGP). Furthermore it is possible to run complex simulations lasting for multiple hours of simulated time with hundreds of clients and tens of routers in reasonable computation time (less than 1/2 hours) and with thousands of clients and hundreds of routers in not quite reasonable computation time (multiple days).

The purpose of the simulations are twofold. On the one hand we use them to determine good parameters for the algorithm. On the other hand we want to understand the behaviour of the system both in simple scenarios as well as more complex ones. Accordingly, we consider various topologies ranging from simple four-node graphs to complex intra-autonomous system topologies such as those provided by Rocketfuel [37]. The specifics of the topologies are discussed when we present the results. However, the principle workload and routing setup is the same across all simulations and summarised below.

Within each simulation run we distinguish two phases: a startup period which lasts 500 seconds and an evaluation period during which we study the performance of the algorithm. The startup period is used to setup the underlying routing system and to enable the workload generator to reach a "stable" state.

### 4.1 Routing

As a traffic engineering protocol, our algorithm relies on an underlying routing system. For simple examples, we choose to install the routing manually; for more complex scenarios, we use OSPF routing. The computation of the routes only takes a few seconds.



**Figure 1:** A set of HTTP servers is connected to a set of HTTP clients via two different routes  $(r, v_1, v_3)$  and  $(r, v_2, v_3)$ .

Information contained in the Rocketfuel maps also comprises inferred IGP weights. These weights are presumably the result of a traffic engineering process. As we do not know the traffic matrix underlying the TE process, we cannot use these IGP weights. Instead we use uniform OSPF weights, resulting in hop count as the distance metric. Using hop count increases the number of equal-cost paths when compared to routing based on, e. g., weights inferred from Rocketfuel measurements. Therefore it provides the algorithm with more flexibility.

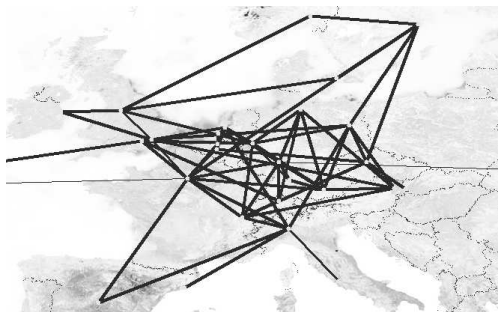
## 4.2 Realistic workload

To account for the bursty nature of traffic on the Internet [28] and the fact that most traffic on the Internet is still using TCP [8], we choose to generate our workload using the Web workload generators of [42] and [33]. Both use a workload model similar to the one imposed by SURGE [3] or NSWeb [43], which generate TCP traffic with realistic properties [11]. To increase burstiness, we disable the use of persistent HTTP connections [25]. Both generators simulate Web users that read a Web page, then are idle for some time before reading another Web page, etc. Therefore one has to allow the system to reach its equilibrium during a *startup period*, where all clients are started at random times (to avoid synchronisation effects) after the route computations have finished and before the startup period ends. Like the timeouts, the simulated file sizes are also heavy-tailed and thus yield many short flows, as well as a significant number of long-lasting flows.

We note that alternative approaches, such as replaying a pre-recorded trace, do not suffice, as they do not account for interacting feedback loops. After all, the self-adjusting traffic engineering protocol *as well as* other control loops, foremost TCP, both react to congestion. Therefore it is crucial to investigate potential interactions of TCP’s congestion control feedback loop with the actions of our TE protocol.

## 4.3 Client/Server location

For the Rocketfuel-based simulations, we install workload clouds containing some number of clients and servers at every border and



**Figure 2:** The Rocketfuel topology for EBONE (AS 1755, .r0 map) consists of 172 routers.

every access router. The number of clients and servers in each workload cloud is linear to the degree of the respective router. This model assumes that well-connected POPs with many routers can be expected to account for a larger share of the traffic than less well-connected POPs. To be more specific, we install 30 clients and 3 servers for each link.

In order to add variability to the experienced RTT values [11], we randomise the link delay for each client and each server. They are chosen uniformly from the interval  $[0, 60]$  ms for clients and  $[0, 25]$  ms for servers. The bandwidth of each end-host is 40 Mbit/s. Since Rocketfuel estimates only link latencies but not link bandwidths, we use the former in our simulations while the latter have to be chosen manually. As TE is most effective in a network where the bottleneck is in the backbone and not at the periphery, we simulate an *underprovisioning* network and set the link bandwidths of the links between routers to 10 Mbit/s.

## 4.4 Traffic Demands

In all of our simulations, the request distribution between Web clients and Web servers and therefore the traffic demands are uniformly distributed, i. e., at all times each client chooses the Web server hosting the next Web page to be downloaded uniformly, regardless of their location. Our workload demands are thus constructed to reflect the traffic matrix model of Zhang et al. [50].

In order to determine the traffic load imposed by such traffic demands in an extremely fast and overprovisioned “ideal” network, we examine the results of a simulation in which all pairs of workload clouds are connected directly by vastly overprovisioned links with minimal delays, and not via a network backbone. Naturally this shifts the network bottleneck from the network core to the network edge. In this scenario, the application layer bandwidth consumed by each client is 21 kBytes/s when averaged over the entire simulation time; the median throughput when downloading Web objects however is 47 MBytes/s. This large discrepancy indicates the burstiness of the generated traffic.

## 4.5 Traffic Engineering Algorithm

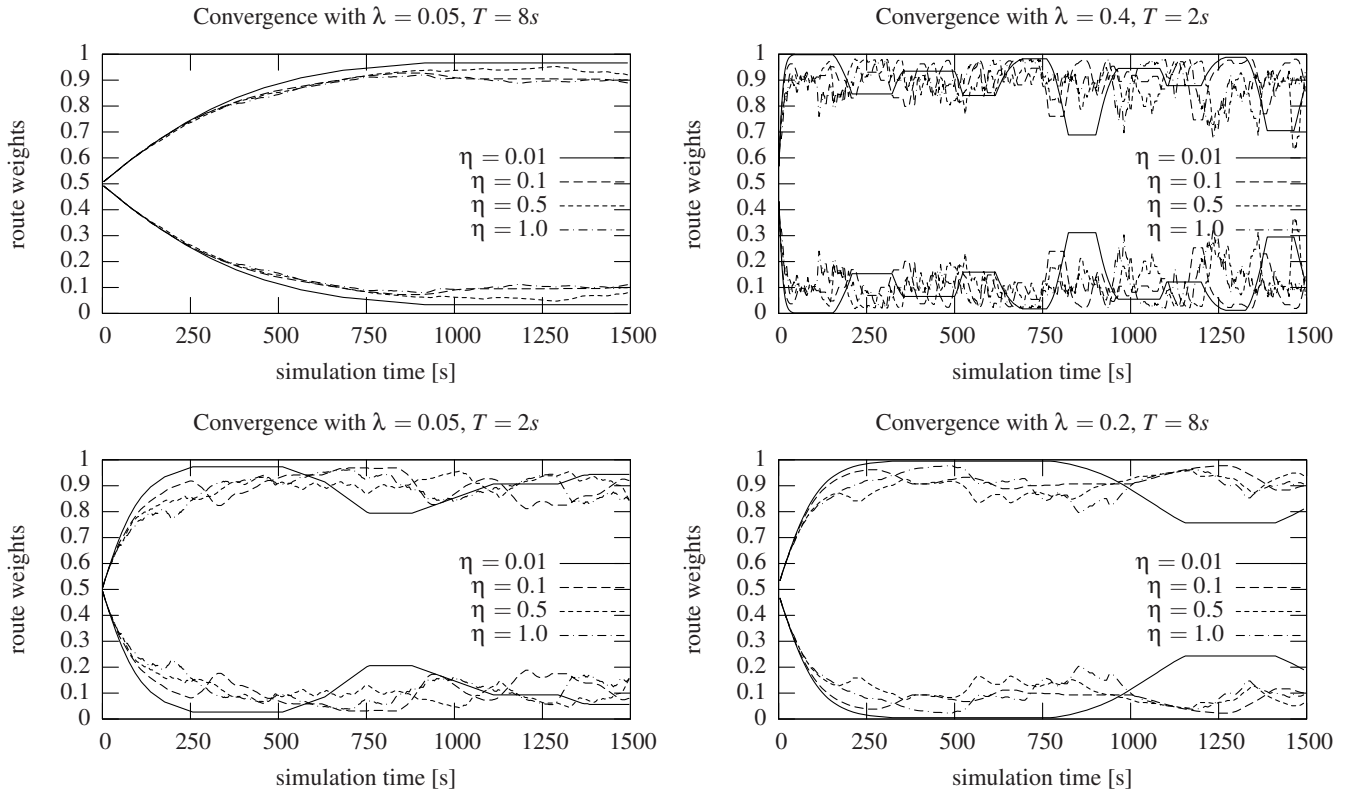
After 500 s, the startup period ends and the *evaluation period* begins. To simplify our notation, we completely ignore the startup period and define the time at  $t = 0$  to be the beginning of the evaluation period. The evaluation period is used to study the behaviour of the TE algorithm. Accordingly, the TE instances at each router are now started. Similar to the Web clients, the TE agents start at random times to avoid synchronisation effects. This happens during  $t \in [0, 1]$  s distributed uniformly across the interval. If communication between routers is enabled, they establish TCP connections to their neighbours in order to start exchanging traffic information. Initially, all route weight settings are neutral; this guarantees an equal traffic distribution among the available equal-cost routes.

When the algorithm starts, the route weights are still at their unbiased uniform setting (i. e., each inversely proportional to the number of same-cost routes to its destination), and have not yet been optimised. Thus the time at which the TE instances are started can be seen as a situation in which a drastic traffic change previously has caused a sudden large-scale readjustment of (almost) all weights—which our algorithm now needs to mend.

In all our simulations we chose link utilisation as the metric to optimise, since it is presumable the most popular traffic engineering goal. Note that our software allows to optimise other goals as well (e. g., minimise latencies).

## 5. ARTIFICIAL TOPOLOGIES

The theoretical background of the algorithm indicates fast con-



**Figure 3:** The above plots show how weights are adapted for two routes with speeds 10Mbit and 100Mbit over time. The optimal weights for this scenario are  $1/11$  and  $10/11$ , respectively. For various values of  $\lambda$ ,  $T$ , and  $\eta$  the above plots show the weights for two different routes over time. For every choice of parameters, we have two plots of weights, one for each route. We observe that oscillation vanishes as  $\tilde{\lambda} = \lambda/T$  decreases.

vergence. But as this analysis cannot take real world factors into account, e. g., interactions with the feedback loop of TCP congestion control, we now evaluate its performance under realistic workload conditions but on simple topologies. This helps us to tune the various parameters of the algorithm as well as evaluate the impact of communication on the behaviour of the algorithm. Furthermore we check how the algorithm reacts to traffic shifts.

### 5.1 Choosing the Rerouting Speed

To avoid oscillation effects, we have to judiciously choose the time interval  $T$ , which specifies how often a router can shift traffic, and the amount of flow  $\lambda$  that the router can shift at any one time. Furthermore, we need to pick the exponential moving average weight  $\eta$  in a way that the algorithm can react quickly enough but does not overshoot.

We start with the topology depicted in Fig. 1 consisting of four routers and focus on the traffic from  $r$  to  $v_3$  with its two alternative paths, one via  $v_1$  the other via  $v_2$ . The router  $r$  distributes traffic sent by a set of HTTP servers connected to it to a set of HTTP clients that are connected to our sink router  $v_3$ . Router  $r$  has two possible routes  $(r, v_1, v_3)$  and  $(r, v_2, v_3)$  to distribute its traffic to  $v_3$ . Our algorithm splits the traffic between the routes by assigning each one a weight. To study the capabilities of the algorithm to move from a uniform traffic distribution which corresponds to equal weights (the initial setting) to an unbalanced (but optimised) traffic distribution, we configure different bandwidths for each path. Links  $(r, v_1)$  and  $(v_1, v_3)$  have bandwidth  $S_1$  whereas links  $(r, v_2)$  and  $(v_2, v_3)$  have bandwidth  $S_2$ . Therefore, the optimal weights for the two routes are  $S_1/(S_1 + S_2)$  and  $S_2/(S_1 + S_2)$ . Note that in this scenario, we

can safely disable communication between the routers: Router  $r$  is able to measure link utilisation on its directly connected links, and moreover there is no cross traffic.

We start with  $S_1 = 10\text{Mbit}$  and  $S_2 = 100\text{Mbit}$  and a workload imposed by 40 HTTP servers (on end-hosts connected to  $r$ ) and 40 clients (connected to  $v_3$ ). Obviously, the optimal weights for this scenario are  $1/11 \approx 0.091$  and  $10/11 \approx 0.909$ . Fig. 3 shows how our algorithm adjusts the weights at  $r$  during the evaluation period, for different choices of  $\lambda$ ,  $T$ , and  $\eta$ . We observe that for all choices of  $\lambda$  and  $T$ , the weights (averaged over time) computed by our algorithm is close to the optimal values. Yet, for large values of  $\lambda$  and small values of  $T$ , the weights oscillate rather heavily around their optimal weights. The critical parameter value is  $\tilde{\lambda} := \lambda/T$ . As this value decreases, both amplitude and frequency of the oscillation decrease until the oscillation vanishes (almost) completely. The simulations are in good conformance with theory.

The top row of Fig. 3 shows two extreme cases: For  $T = 8\text{s}$  and  $\lambda = 0.1$ , the  $\tilde{\lambda}$  is  $0.0125\text{s}^{-1}$ . Convergence to the optimal weights is very smooth, but rather slow. For the other extreme case,  $T = 2\text{s}$  and  $\lambda = 0.4$ , the resulting  $\tilde{\lambda}$  is  $0.2\text{s}^{-1}$ . Here we observe heavy oscillation of the weights around their optimal values. This is expected, as the load and therefore the losses vary due to the variability in the imposed traffic load.

The bottom row of Fig. 3 shows the results with  $\tilde{\lambda} = 0.025\text{s}^{-1}$  (i. e.,  $\lambda = 0.05$  and  $T = 2\text{s}$  for the left-hand side plot and  $\lambda = 0.2$  and  $T = 8\text{s}$  for the right-hand side plot). Here, we see significant reduction in the oscillation and fast convergence. With a longer measurement interval  $T$  and larger value of  $\lambda$ , the amplitude of the oscillation increases slightly. We conclude that one should choose

$T$  as small as possible such that reasonable measurements can be taken.  $\lambda$  should be chosen as large as possible but small enough to avoid oscillation. The plots presented in this section use link delays of 10ms; similar results are obtained for other values of link delays (0.1 ms to 100ms) and workload parameters, which we omit for brevity.

Fig. 3 also reveals the impact of the EMA weight parameter  $\eta$ : Small values for  $\eta$  smooth peaks and values and thus stabilise the weights that the algorithm is assigning to each alternative path. Values such as 0.01 appear to be too small as they cause the algorithm to be inert—it can no longer react to the effects of its rerouting decision.

Choosing the right parameter is a tradeoff between convergence speed and oscillation. We suggest to choose  $T = 2$  s,  $\lambda = 0.05$ , and  $\eta = 0.1$  as a good compromise. These parameter settings achieve convergence within 200s with almost no oscillation.

## 5.2 Quality of Adaption

To study how the algorithm is capable of utilising the capacity on the links, we examine how the relative utilisation of the two routes changes over time: If the algorithm works well, the load on both routes has to be balanced over the two links, i. e., if the total imposed load is 11 Mbit and one link has capacity  $S_1 = 10$  Mbit while the other has capacity  $S_2 = 100$  Mbit, one expects the former to carry 1 Mbit and the latter 10 Mbit; thus each route should see a utilisation of 10%. Using the same topology (Fig. 1), we assign different bandwidths to one link  $S_1 \in \{10 \text{ Mbit}, 34 \text{ Mbit}, 100 \text{ Mbit}\}$  while keeping the other at  $S_2 = 100$  Mbit. The optimal weights, ignoring traffic variability, are consequently 0.09/0.91, 0.25/0.75, and 0.5/0.5. Fig. 4 shows that the algorithm finds the optimal weights (top graph) and that the link utilisation is balanced once the algorithm has converged to the optimal weights (bottom graph).

## 5.3 Necessary Number of End Hosts

One of the insights pointed out in Section 3.2.2 is that derandomising using hash functions can only perform well if the range of input values to the hash function is sufficiently large. Since we use source and destination IP as well as port numbers as inputs, and since all packets belonging to one TCP connection thus form the same input, the variable in question is thus the number of distinct connections.

We can increase the number of distinct connections by increasing the number of clients or servers, or by changing the workload parameters. So as not to change the established parameters of our workload generators, we choose to only vary the number of clients and servers. For these experiments we again use the topology shown in Fig. 1 with link bandwidths 10 Mbit and 100 Mbit. The routers  $r$  and  $v_3$  are connected to a set of  $n$  end hosts, running HTTP servers and clients, respectively. We experiment with different values of  $n$ . Note, that while there are only  $n^2$  client-server pairs, the set of connections and therefore the input to the hash function is even larger, as end hosts pairs can maintain multiple connections in parallel on different ports.

For these experiments we use very conservative parameters with regards to convergence speed by choosing  $T = 4$  s,  $\lambda = 0.05$ ,  $\eta = 0.2$ . Fig. 5 shows how the weights assigned of the two routes change over time. We observe that for one, two, and five end hosts on every side, the algorithm is unable to find the optimal weights in the beginning. However, for  $n \geq 10$  clients and servers, the algorithm reliably finds the optimal weights. While we still observe some oscillation for  $n = 10$ , the solution becomes stable as  $n$  increases further.

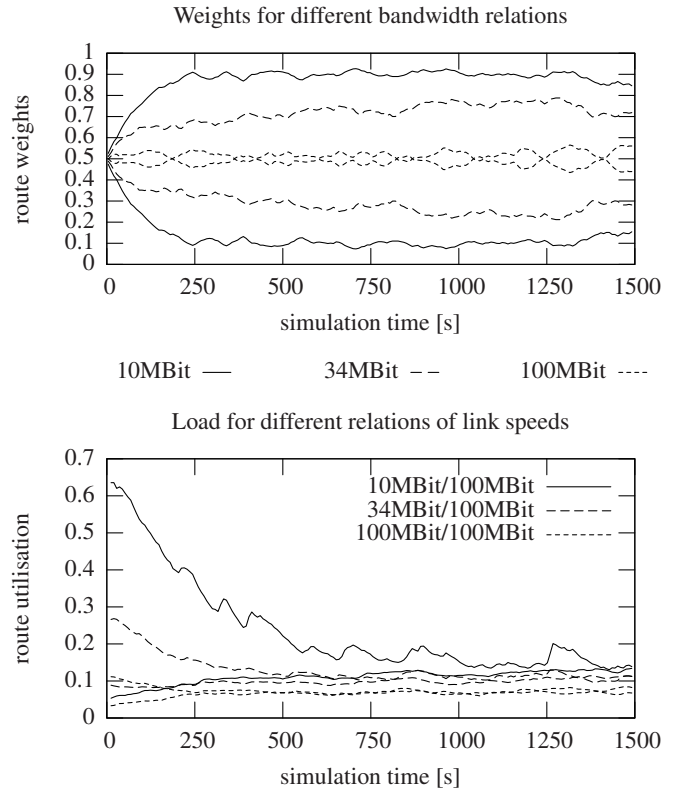


Figure 4: The weights assigned to the routes for different relations of bandwidths. The upper plot shows the weights whereas the lower plot shows the link utilisation of the two routes.

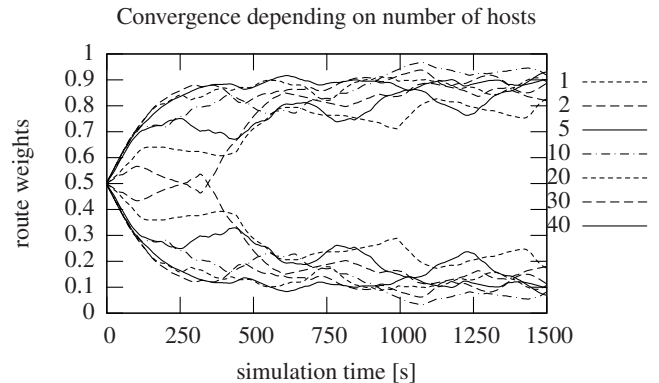


Figure 5: Dependence of the stability of the solution dependent on the number of hosts  $n$ .

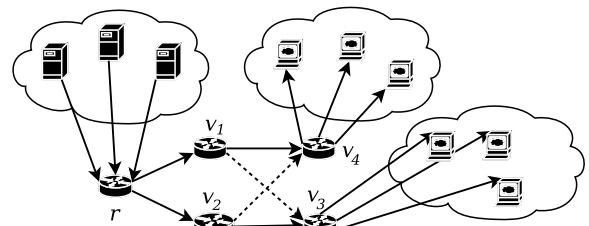


Figure 6: The Web servers are connected to two client subnets, each reachable via two different routes. Bottleneck edges are dashed.

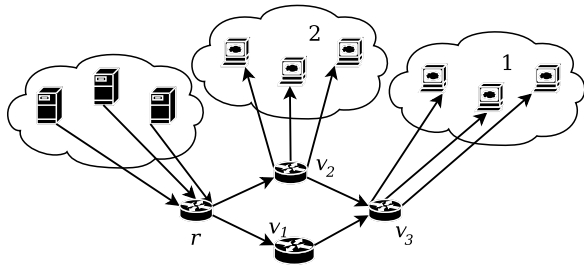


Figure 8: A set of  $D$  Web clients is connected to router  $v_3$  and a set of  $D/k$  clients waking up at time  $T_1 = 500$ s is connected to router  $v_2$ .

## 5.4 Benefiting from Communication

So far none of the topologies can significantly benefit from communication between the routers, as the route utilisation is more or less equivalent to the link utilisation at the router, which can be measured directly. Therefore we now study a topology that can benefit from communication, see Fig. 6. The router  $r$  is connected to two routers  $v_1$  and  $v_2$  via 100Mbit links. Each of these routers is connected to two other routers  $v_3$  and  $v_4$ , who are each connected to a subnet of end hosts. If  $(v_1, v_3)$  as well as  $(v_2, v_4)$  are 100Mbit links and links  $(v_1, v_4)$  as well as  $(v_2, v_3)$  (shown as dashed lines) are assigned smaller bandwidth (in our simulations we used 10Mbit, 34Mbit, and 100Mbit) these become the bottlenecks in this topology. We see that  $r$  can reach the two subnets of end hosts either via  $v_1$  or via  $v_2$ . However, it cannot directly measure which of the paths has the least utilisation, since the bottlenecks are behind  $v_1$  and  $v_2$ , respectively. To enable such measurements we need to enable communication. If the two routers  $v_1$  and  $v_2$  are sending messages every  $T = 8$ s about the two client networks which are modelled as two prefixes, this causes a communication overhead of only  $4 \cdot 6 \text{ bytes} / 8 \text{ s} = 3 \text{ bytes/s}$ .

Router  $r$  now has to maintain four weights, two for each subnet. The optimal weights can be computed as above. From Fig. 7 we see that our algorithm is able to quickly find these weights, and it converges fast. Further simulations show that the algorithm is not able to find the optimal weights for this topology if communication is disabled; in this case both route weights vary around 0.5. This means that if the algorithm has no information about which route is better, it reverts to the default behaviour: it just distributes the traffic equally across all best equal-cost paths.

## 5.5 Drifting Traffic Demands

Up to now, we have assumed that the traffic demands are static, even though the workload itself is bursty. However, the ultimate reason for deploying a traffic engineering protocol is to adapt the routing quickly to traffic changes. To examine such a situation, we use the topology shown in Fig. 8. Here, the HTTP clients at end hosts in subnet 2 are inactive at first and start waking up after time  $T_1 = 500$ s within some time interval  $T_2$ . These HTTP clients are then responsible for additional traffic on link  $(r, v_2)$ . Let the total traffic demand imposed by the sets of HTTP clients in subnet 1 and 2 be  $D$  and  $D/k$ , respectively. Before the additional HTTP clients are woken up, the traffic is supposed to be equally balanced between the two routes  $P_1 = (r, v_1, v_3)$  and  $P_2 = (r, v_2, v_3)$  (with respective weights of 0.5 and 0.5). After the additional clients have woken up, an additional traffic of volume  $D/k$  is added to link  $(r, v_2)$ . Hence, the traffic is balanced if the weights  $w$  and  $1 - w$  for  $P_1$  and  $P_2$  satisfy  $w \cdot D = (1 - w) \cdot D + \frac{D}{k}$ , or, equivalently,  $w = \min\{\frac{1}{2} + \frac{1}{2k}, 1\}$ .

Fig. 9 shows the simulation results for  $k = 1, 2$ , and 4. In this

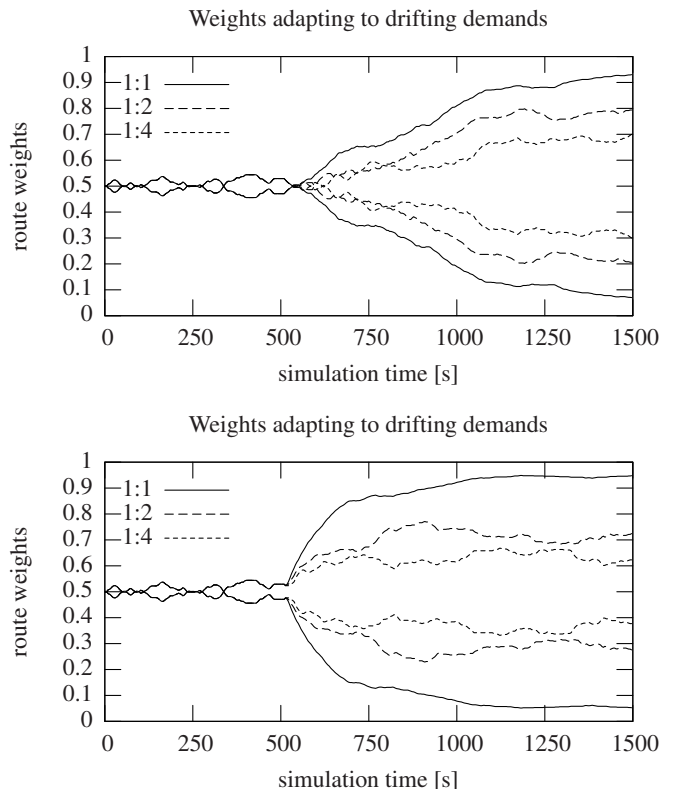


Figure 9: Weights following a change of traffic demands within 500s (top) and 10s (bottom). Each plot shows simulations for different ratios of additional traffic: 1, 1/2, and 1/4.

case the optimal weights for  $P_1$  are 1, 0.75, and 0.625, respectively. We observe that our algorithm indeed readjusts the weights to these desired values. How quickly this happens, depends on the length of the interval  $[T_1, T_2]$  during which the additional HTTP clients are woken up. We observe that for  $T_2 - T_1 = 500$ s, the weights assigned by our algorithm follow the intermediate flow demands smoothly, whereas for  $T_2 - T_1 = 10$ s, the algorithm needs its regular convergence time to adapt to the sudden change in flow demands.

## 6. ROCKETFUEL TOPOLOGY

In this section, we analyse the performance of REPLEX in simulations involving a realistic topology. We focus on the EBONE network topology from the 2002 Rocketfuel [37] data shown in Fig. 2, as it features a sizable number of routers and links. Moreover its good connectivity offers good path diversity, which our algorithm can take advantage of.

In total, we connect 5,400 Web clients and 417 Web servers to the 61 access and border routers of this topology. Since each of the 61 resulting “workload clouds” is assigned its own network prefix, the communication overhead on each link amounts to  $\frac{61 \cdot 6 \text{ Bytes}}{2.5 \text{ s}} = 1.2 \text{ kbit/s}$  in the simulations when communication is enabled. As parameters, we choose  $\lambda = 0.3, \epsilon = 0.15, T = 2.5$ s.

We start with analysing the oscillation behaviour of our algorithm. As the topology consists of 367 links, it is not feasible to show weight changes for individual links. Instead, we sum up all route weight changes at all routers for each 5-second interval. As already indicated by the experiments in Section 5, we expect the weights to be subject to small fluctuations even in the converged state, due to the bursty nature of the generated realistic workload

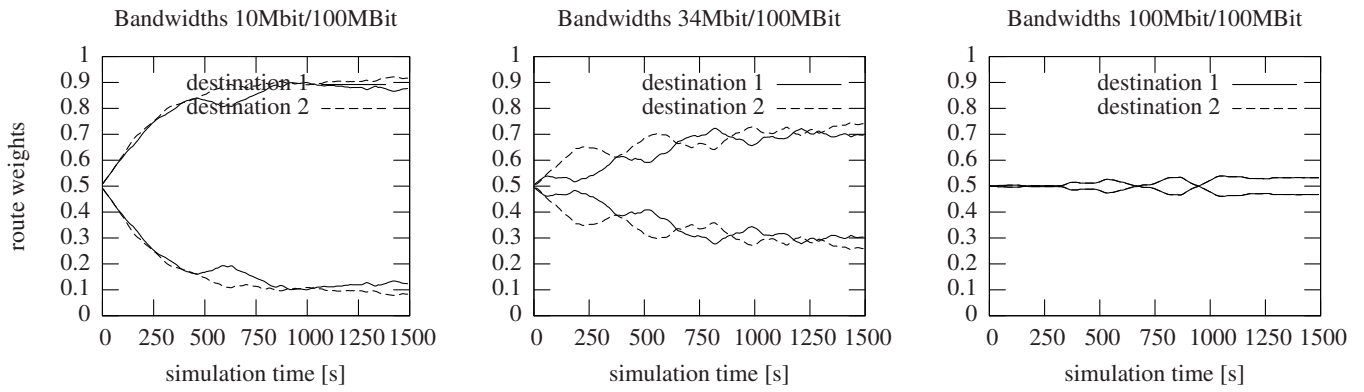


Figure 7: Simulation results for the topology depicted in Fig. 6 using bandwidths 10Mbit, 34Mbit and 100Mbit, respectively for the bottleneck edges and 100Mbit for the other edges.

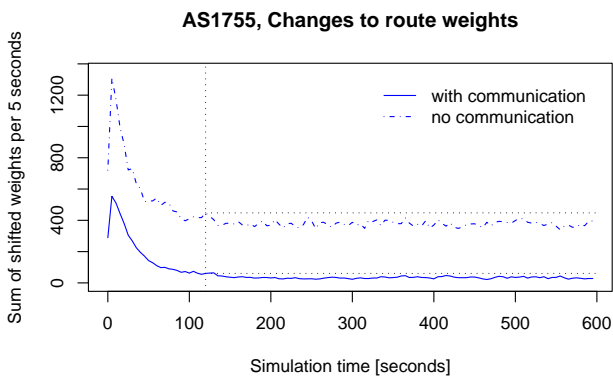


Figure 10: Development of weight shifts during the first 10 minutes. Large weight shifts occur only within the first 2 minutes (vertical line).

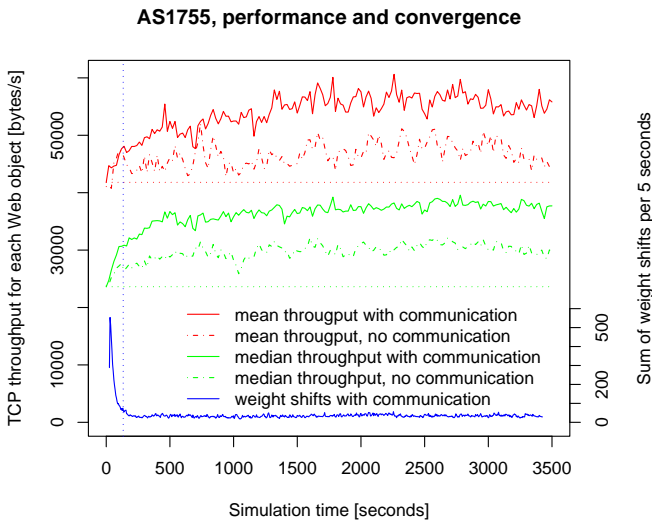


Figure 11: Network performance during 1 hour of simulation time. Application layer throughput increases even after the main weight shift has occurred (vertical line) and only ceases increasing well after 10 minutes.

traffic.

Fig. 10 shows the development of the weight changes for scenarios with (continuous line, bottom) and without communication (dash-dotted line, top). We see that in the former scenario, our protocol converges quickly as indicated by the theory: Significant weight shifts occur only within the first 120 s (marked by the vertical line) of the simulation, thereby showing a swift decline. If we disable communication between the routers such that they have to rely on their own measurements, algorithm converges at roughly the same time, although significantly more weight shifts occur.

While convergence certainly is a pleasant property, it remains to show that REPLEX actually improves network performance. We chose to measure the performance experienced by end users: For each transferred HTTP object, we measure the achieved TCP bandwidth, and group these values into 20-second bins. Fig. 11 shows the development of the average (red; top lines) and median (green; middle lines) for each 20-second bin over time. Again we compare the scenario with communication (continuous lines) against the scenario without (dash-dotted lines). For comparison, we repeat the development of weight shifts in the communication scenario (blue line; bottom) from Fig. 10.

Fig. 11 moreover shows that even if we disallow communication (dash-dotted lines), REPLEX improves network performance significantly: While median (average) throughput begins at around 23,000 Bytes/s (42,000), but it increases to around 29,700 Bytes/s (46,500) within a short time. This means that REPLEX increases TCP median (average) throughput by 29% (11%).

If the routers can exchange information (continuous lines), the gain is even greater. In this case, REPLEX boosts median (average) throughput to around 37,300 Bytes/s (54,000), which corresponds to an increase of the performance as perceived by the end user by 64% (29%).

Fig. 11 also allows us to compare the development of the weight shifts against the increasing throughput. Again, we have marked  $t = 120$ s by a vertical line. We see that a considerable throughput gain has been achieved when the weights (continuous blue line at bottom) cease undergoing significant changes, i.e., after two minutes. Interestingly though, the algorithm has not fully converged at this point, as one can see the performance to increase significantly even beyond this point for some more minutes.

## 7. SUMMARY AND OUTLOOK

In this paper we have presented REPLEX, a traffic engineering protocol that automatically balances traffic between equal-cost routes provided by an underlying routing architecture, such as OSPF, MPLS or BGP. Our simulations show that the protocol con-

verges quickly, which is in good conformance with its game-theoretic background. REPLEX scales well even in large networks, since information distribution is done in a distance vector-like fashion. The protocol-induced communication overhead on a link is only linearly dependent on the number of prefixes. Furthermore the “protocol” is even applicable in a scenario where communication between the routers is not desired, e. g., in interdomain contexts.

We plan to perform further simulation-based analyses. These comprise simulations involving other AS topologies and comparisons against other TE techniques (e. g., [41]), and the behaviour if two neighbouring ASes use REPLEX without each other’s knowledge. While our evaluation focuses on minimising the maximum link utilisation as the traffic engineering goal, we intend to analyse other metrics in the future as well, e. g., minimising latencies by measuring queue lengths, or minimising packet losses. Moreover we intend to improve performance (a) by applying the weight changes in a “soft” manner [21] so as to reduce the number of TCP connections affected by shift-induced packet reordering per time interval, and (b) by changing the inter-update times on demand (“important news travels fast”). We believe that further improvements can be achieved (c) by making use of typical flow size distributions [44] when using improved hashing techniques such as DHFV [20] or aggregating same-egress prefixes [38], (d) by providing more path alternatives to select from by using MPLS tunnels, for which TeXCP’s link feedback mechanism [21] can provide valuable information for each LSP, or (e) by using PSTARA-like routing [31] instead of MPLS-based solutions.

## 8. REFERENCES

- [1] D. Applegate and E. Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs. In *Proc. ACM SIGCOMM Conference*, 2003.
- [2] B. Awerbuch and R. D. Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proc. 36th Ann. ACM. Symp. on Theory of Comput. (STOC)*, 2004.
- [3] P. Barford and M. Crovella. Generating representative Web workloads for network and server performance evaluation. In *Proc. ACM SIGMETRICS Conference*, 1998.
- [4] M. Beckmann, C. B. McGuire, and C. B. Winsten. *Studies in the Economics and Transportation*. Yale University Press, 1956.
- [5] P. Berenbrink, T. Friedetzky, L. A. Goldberg, P. Goldberg, Z. Hu, and R. Martin. Distributed selfish load balancing. In *Proc. 17th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2006.
- [6] A. Blum, E. Even-Dar, and K. Ligett. Routing without regret, 2005. Manuscript.
- [7] Z. Cao, Z. Wang, and E. W. Zegura. Performance of hashing-based schemes for Internet load balancing. In *Proc. IEEE INFOCOM Conference*, 2000.
- [8] H. Dreger, A. Feldmann, M. Mai, V. Paxson, and R. Sommer. Dynamic application-layer protocol analysis for network intrusion detection. In *Proc. 15th Usenix Security Symposium*, 2006 (to appear).
- [9] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proc. IEEE INFOCOM Conference*, 2001.
- [10] E. Even-Dar and Y. Mansour. Fast convergence of selfish rerouting. In *Proc. 16th Ann. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2005.
- [11] A. Feldmann, A. Gilbert, P. Huang, and W. Willinger. Dynamics of IP traffic: A study of the role of variability and the impact of control. In *ACM SIGCOMM Conference*, Sept. 1999.
- [12] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, and J. Rexford. NetScope: Traffic engineering for IP networks. *IEEE Network Magazine*, 2000.
- [13] S. Fischer, H. Räcke, and B. Vöcking. Fast convergence to Wardrop equilibria by adaptive sampling methods. In *Proc. 38th Ann. ACM. Symp. on Theory of Comput. (STOC)*, Seattle, WA, USA, May 2006. ACM.
- [14] S. Fischer and B. Vöcking. On the evolution of selfish routing. In S. Albers and T. Radzik, editors, *Proc. 12th Ann. European Symp. on Algorithms (ESA)*, number 3221 in Lecture Notes in Comput. Sci., Bergen, Norway, September 2004. Springer-Verlag.
- [15] S. Fischer and B. Vöcking. Adaptive routing with stale information. In M. K. Aguilera and J. Aspnes, editors, *Proc. 24th Ann. ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (PODC)*, Las Vegas, NV, USA, 2005.
- [16] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. In *IEEE Communication Magazine*, 2002.
- [17] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proc. IEEE INFOCOM Conference*, 2000.
- [18] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. In *IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 4, 2002.
- [19] W. Hao and R. Ito. Dynamics of load-sensitive adaptive routing. In *Proc. IEEE International Conference on Communications (ICC)*, 2005.
- [20] J.-Y. Jo, Y. Kim, H. J. Chao, and F. Merat. Internet traffic load balancing using dynamic hashing with flow volume. In *Proc. SPIE ITCOM*, 2002.
- [21] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: responsive yet stable traffic engineering. In *Proc. ACM SIGCOMM*, 2005.
- [22] D. Katz, K. Kompella, and D. Yeung. Traffic engineering extensions to OSPF version 2. Internet Draft, 2003.
- [23] A. Khanna and J. Zinky. The revised ARPANET routing metric. In *Proc. ACM SIGCOMM Conference*, 1989.
- [24] T. Krämer. IP traffic engineering—OSPF vs. MPLS. Master’s thesis, Universität des Saarlandes, 2003.
- [25] B. Krishnamurthy and J. Rexford. *Web Protocols and Practice*. Addison-Wesley, 2001.
- [26] M. Laor and L. Gendel. The effect of packet reordering in a backbone link on application throughput. *IEEE Network*, September/October 2002.
- [27] Y. Liu and A. L. N. Reddy. Multihoming route control among a group of multihomed stub networks. Technical Report TAMU-ECE-2006-01, 2006.
- [28] K. Park and W. Willinger, editors. *Self-Similar Network Traffic and Performance Evaluation*. Wiley-Interscience, 2000.
- [29] C. Pelsser, S. Uhlig, and O. Bonaventure. On the difficulty of establishing interdomain LSPs. In *IEEE IPOM*, 2004.
- [30] J. Postel et al. RFC 793, September 1981. <http://www.ietf.org/rfc/rfc793.txt>.
- [31] V. Raghunathan and P. R. Kumar. A Wardrop routing protocol for ad hoc wireless networks. In *IEEE CDC*, 2004.
- [32] V. Raghunathan and P. R. Kumar. Issues in Wardrop routing in wireless networks. In *IEEE WICON*, 2005.
- [33] Renesys. The SSFNet network simulator. <http://www.ssfnet.org/>.
- [34] M. Roughan, M. Thorup, and Y. Zhang. Traffic engineering with estimated traffic matrices. In *Proc. ACM SIGCOMM Conference*, 2003.
- [35] T. Roughgarden and É. Tardos. How bad is selfish routing? *J. ACM*, 49(2), 2002.
- [36] N. Skrypnuk. Load-sensitive routing. Master’s thesis, TU München, 2006.
- [37] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM Conference*, Pittsburgh, PA, USA, August 2002. ACM.
- [38] A. Sridharan, R. Guérin, and C. Diot. Achieving near-optimal traffic engineering solutions for current OSPF/IS-IS networks. In *Proc. IEEE INFOCOM Conference*, 2003.
- [39] R. Teixeira. *Network Sensitivity to Intradomain Routing Changes*. PhD thesis, University of California San Diego, August 2005.
- [40] R. Teixeira, N. Duffield, J. Rexford, and M. Roughan. Traffic matrix reloaded: Impact of routing changes. In *Proc. Passive and Active Measurement*, 2005.
- [41] TOTEM—TOolbox for Traffic Engineering Methods. <http://totem.run.montefiore.ulg.ac.be/>.
- [42] C. Vollmert. A Web workload generator for the SSFNet network simulator. Bachelor’s thesis, Technische Universität München, 2004.
- [43] J. Wallerich. Design and implementation of a WWW workload generator for the ns-2 network simulator. Master’s thesis, Universität des Saarlandes, 2001.
- [44] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger. A methodology for studying persistency aspects of Internet flows. *ACM SIGCOMM Computer Communication Review*, 35, 2005.
- [45] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg. COPE: Traffic engineering in dynamic networks. In *Proc. ACM SIGCOMM Conference*, 2006.
- [46] J. G. Wardrop. Some theoretical aspects of road traffic research. In *Proc. of the Institute of Civil Engineers, Pt. II*, 1952.
- [47] X. Xiao, A. Hannan, B. Bailey, and L. Ni. Traffic engineering with MPLS in the Internet. In *IEEE Network Magazine*, March 2000.
- [48] W. Xu and J. Rexford. Miro: Multi-path interdomain routing. In *Proc. ACM SIGCOMM Conference*, 2006.
- [49] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Towsley. Optimal routing with multiple traffic matrices: Tradeoff between average case and worst case performance. In *Proc. 13th International Conference on Network Protocols (ICNP)*, 2005.
- [50] Y. Zhang, M. Roughan, C. Lund, and D. Donoho. An information-theoretic approach to traffic matrix estimation. In *Proc. ACM SIGCOMM*, 2003.