Delay-based Congestion Control for Multipath TCP

Yu Cao*†, Mingwei Xu*‡, Xiaoming Fu§
*Tsinghua University, †National Data Switch Center, China
‡Tsinghua National Laboratory for Information Science and Technology (TNList)
§Institute of Computer Science, University of Goettingen
{caoyu08, xmw}@csnet1.cs.tsinghua.edu.cn, fu@cs.uni-goettingen.de

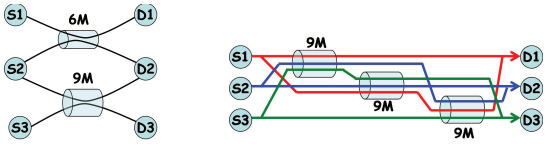
***Abstract*—With the aid of multipath transport protocols, a multihomed host can shift some of its traffic from more congested paths to less congested ones, thus compensating for lost bandwidth on some paths by *moderately* increasing transmission rates on other ones. However, existing multipath proposals achieve only coarse-grained load balancing due to a rough estimate of network congestion using packet losses.**

This paper formulates the problem of multipath congestion control and proposes an approximate iterative algorithm to solve it. We prove that a fair and efficient traffic shifting implies that every flow strives to equalize the extent of congestion that it perceives on all its available paths. We call this result "Congestion Equality Principle". By instantiating the approximate iterative algorithm, we develop *weighted Vegas (wVegas)*, a delay-based algorithm for multipath congestion control, which uses packet queuing delay as congestion signals, thus achieving fine-grained load balancing. Our simulations show that, compared with loss-based algorithms, wVegas is more sensitive to changes of network congestion and thus achieves more timely traffic shifting and quicker convergence. Additionally, as it occupies fewer link buffers, wVegas rarely causes packet losses and shows better intra-protocol fairness.

I. Introduction

With the aid of multipath transport protocols such as Multipath TCP (MPTCP) [1] and CMT-SCTP [2], a flow can split its traffic across multiple available paths between multihomed hosts, into multiple *subflows*, for improving throughput and robustness, thus utilizing network resources more efficiently than traditional TCP. One potential application for multipath transfer is that a wireless host may transfer data through the WiFi and the 3G paths in parallel, so as to keep its connections alive even if one of the network interfaces fails. Lately, MPTCP is also considered as promising for load balancing in Data Center Networks (DCNs) [3], where a host usually has plenty of divergent paths to others.

For multipath transfer, performing congestion control independently on each path would do harm to fairness, as shown by CMT-SCTP [2]. Thus, we believe that a major objective of multipath congestion control is to *couple* all the subflows belonging to a flow together so as to achieve both fairness and efficiency. By this kind of coupling method, most of existing multipath proposals [4]–[6] provide the ability of load balancing that can shift some traffic from more congested paths to less congested ones, thus compensating for lost bandwidth on some paths by moderately increasing transmission rates on other ones. However, these proposals achieve only coarse-grained load balancing, because they estimate network congestion and then trigger traffic shifting using packet losses that lack of the finer-grained information related to the extent of congestion. Furthermore, we argue that since packet losses indicate a quite serious congestion in most cases, traffic should be shifted as earlier as possible before losses occur, in order to avoid performance degradation caused by loss recovery.



(a) (b)

Fig. 1. The examples of resource pooling [7]

A fine-grained load balancing should outperform the coarse-grained in terms of both fairness and efficiency. We use two examples to illustrate the ideal outcome of load balancing. In Fig. 1(a), three flows compete for two bottleneck links with capacities of 6Mbps and 9Mbps, respectively. The most fair bandwidth sharing requires that S2 shifts some of its traffic from the lower bandwidth path onto the higher one until it occupies 1Mbps on the top link and 4Mbps on the bottom. Another example is about efficiency. In Fig. 1(b), which was firstly presented in [6], every flow has two paths available for data transfer. If each flow transmits data at rate $0 \leq x \leq 9$ on its one-hop path and at rate $(9-x)/2$ on its two-hop path, then bandwidth sharing is always fair. Among these fair outcomes, the most efficient one is $x = 9$, because every flow can obtain the maximum transmission rate.

This paper studies the issue of how a flow determines the quantity of traffic shifted from one path to others with only local knowledge on network resources and congestion status. Our contributions are three-fold. First, we proved that *a fair and efficient traffic shifting implies that every flow strives to equalize the extent of congestion that it perceives on all its available paths, namely the "Congestion Equality Principle"*. Second, we formulated the problem of multipath congestion control and proposed an approximate iterative algorithm to solve it. Third, by instantiating the approximate iterative algorithm, we developed *weighted Vegas (wVegas)*, a delay-based algorithm for multipath congestion control, which uses packet queuing delay as congestion signals, thus achieving fine-grained load balancing. wVegas assigns a weight for each subflow and adaptively adjusts it according to the Congestion Equality Principle. The weight quantifies the aggressiveness of competition for bandwidth. Thus, the subflow on less

978-1-4673-2447-2/12/$31.00

congested paths can get a larger weight hence competing more aggressively, which in turn will lead to an increase in the extent of congestion on the corresponding path, and vice versa. In theory, this cycle repeats until all the paths used by each flow in the network become equally congested. At the equilibrium point, network resources will be fairly and efficiently shared by all the flows. It is worth emphasizing that the Congestion Equality Principle and the approximate iterative algorithm together establish a general framework for designing an algorithm of multipath congestion control. wVegas is precisely derived from this framework.

As the name indicates, wVegas is originated from TCP-Vegas [8] that measures packet queuing delay to estimate the extent of network congestion and attempts to backlog α packets [1] in link queues [9]. When many flows are competing for a bottleneck link, the bandwidth obtained by each flow is proportional to its occupied buffer size in the link queue. Thus, it is reasonable for a flow to leverage the parameter α as a knob to controlling the aggressiveness of competition for bandwidth. From these observations comes the design philosophy of wVegas, which can be summarized as follows. First, on each path, wVegas performs in the same way as TCP-Vegas. Second, for a flow, the total sum of the parameters α of subflows is fixed, regardless of the number of subflows. This property contributes to the intra-protocol fairness of wVegas. Third, and most significantly, wVegas adaptively adjusts the parameter α thereby influencing the transmission rate of the corresponding subflow for the purpose of equalizing the extent of congestion on the path. We define the normalized α as the weight of subflows. Thus, in this sense, the weight quantifies the aggressiveness of competition for bandwidth. The core of wVegas is the weight adjustment algorithm. Incidentally, increasing the weight of a subflow may not always push up the transmission rate, albeit making that subflow more aggressive to compete for bandwidth, because other flows might also increase the weight of their own subflows.

Compared with packet loss events, packet queuing delay provides the fine-grained information related to the extent of congestion. This helps wVegas achieve more timely traffic shifting and quicker convergence, hence fine-grained load balancing, as shown in the simulations. Because of moderate buffer consumption in link queues, wVegas also rarely causes packet losses and shows better intra-protocol fairness. wVegas, on the other hand, inherits the limitations of TCP-Vegas. Specifically, the effectiveness of wVegas depends upon the measurement accuracy of Round Tip Times (RTTs). This requires high-resolution timers [10], especially in the network like DCNs, where RTTs are roughly on the order of hundreds of microseconds. Besides, wVegas behaves less aggressively when competing for bandwidth with loss-based algorithms, and less efficiently on high bandwidth-delay product paths. Despite of these limitations, we think wVegas is a good starting point in the realm of delay-based multipath congestion control. We will study the above problems in the future.

[1]Actually, TCP-Vegas has two configurable parameters, α and β, for adjusting the window size during the congestion avoidance period. Since α and β are commonly very close to each other, we use one of them for the sake of brevity.

The remainder of the paper is organized as follows. In Section II, we first formulate the problem of multipath congestion control and then derive the Congestion Equality Principle and the approximate iterative algorithm. Section III presents the details of wVegas. The implementation of wVegas is discussed in Section IV, and its performance is evaluated in Section V. Finally, we briefly overview related work in Section VI and conclude the paper in Section VII.

II. Problem Formulation and Approximate Iterative Algorithm

A. Network Utility Maximization Model

We model a network as the set L of links with the finite capacities $\mathbf{c} = (c_l, l \in L)$, which are shared by the set S of flows. A path $r \in R$ is defined as the subset $L_r \subseteq L$. The relationship between L and R is given by the routing matrix $\mathbf{A}$, where $a_{l,r} = 1$ if $l \in L_r$, and $a_{l,r} = 0$ otherwise. Each flow $s \in S$ is associated with a subset $R_s \subseteq R$. This relationship is given by the matrix $\mathbf{B}$, where $b_{s,r} = 1$ if $r \in R_s$, and $b_{s,r} = 0$ otherwise. Let $x_{s,r}$ be the rate of flow s on path r, and $y_s = \sum_{r \in R_s} x_{s,r}$ be the total rate of flow s. Denote the vector $(x_{s,r}, s \in S, r \in R_s)$ by $\mathbf{x}$, and the vector $(y_s, s \in S)$ by $\mathbf{y}$. When flow s transmits data at rate y_s, it obtains an utility $U_s(y_s)$. Suppose $U_s(\cdot)$ is increasing, strictly concave and twice continuously differentiable in the nonnegative domain. Define $U_s(0) = -\infty$. The objective of congestion control is to determine appropriate rates for the flows so as to maximize the total utility subject to link capacity constraints. Thus, we have

$$\begin{aligned} \max_{\mathbf{x} \geq \mathbf{0}} \ & \sum_{s \in S} U_s(y_s) \\ s.t. \quad & \mathbf{y} = \mathbf{B}\mathbf{x} \\ & \mathbf{A}\mathbf{x} \leq \mathbf{c}. \end{aligned} \tag{1}$$

There exists a unique optimal solution for $\mathbf{y}$ since the objective function is strictly concave and the feasible region is compact. However, it is not true that the optimal $\mathbf{x}$ is also unique, because the objective function is *not strictly* concave for $\mathbf{x}$. Consider the Lagrangian function

$$\begin{aligned} L(\mathbf{x}, \lambda) &:= \sum_{s \in S} U_s(y_s) + \sum_{l \in L} \lambda_l \left(c_l - \sum_{r \in R} a_{l,r} x_r \right) \\ &= \sum_{s \in S} U_s(y_s) - \sum_{l \in L} \sum_{r \in R} \lambda_l a_{l,r} x_r + \sum_{l \in L} \lambda_l c_l \\ &= \sum_{s \in S} U_s(y_s) - \sum_{r \in R} q_r x_r + \lambda \mathbf{c}^T \\ &= \sum_{s \in S} U_s(y_s) - \sum_{s \in S} \sum_{r \in R} b_{s,r} q_r x_r + \lambda \mathbf{c}^T \\ &= \sum_{s \in S} U_s \left(\sum_{r \in R_s} x_{s,r} \right) - \sum_{r \in R_s} q_r x_{s,r} + \lambda \mathbf{c}^T, \end{aligned}$$

where the multiplier $\lambda_l \geq 0$ can be interpreted as the price or the congestion signal associated with link l, and

$$q_r = \sum_{l \in L} \lambda_l a_{l,r} \tag{2}$$

is the aggregate price of the links constituting path r. Thus we call q_r the path price. Define

$$L_s(\lambda) := \max_{\substack{x_{s,r} \geq 0 \\ r \in R_s}} U_s \left(\sum_{r \in R_s} x_{s,r} \right) - \sum_{r \in R_s} q_r x_{s,r}, \tag{3}$$

$$D(\lambda) := \sum_{s \in S} L_s(\lambda) + \lambda \mathbf{c}^T. \tag{4}$$

So the dual problem of (1) is

$$\min_{\lambda \geq \mathbf{0}} D(\lambda). \tag{5}$$

By introducing link prices $\lambda = (\lambda_l, l \in L)$, the original problem (1) is decomposed into a master problem (5) and a number of sub-problems (3). Each sub-problem corresponds to a local optimality related to a flow with only local knowledge $q_r, r \in R_s$. This duality structure allows a decentralized approach to reaching the optimal solution of (1).

The gradient projection algorithm [11] can be used to solve (5) in an iterative way, just as did [12]. In brief, on the side of sources, given λ, flow s locally achieves optimality by solving (3) and then broadcasts the optimal solution $x^*_{s,r}(\lambda), r \in R_s$ to links. On the side of links, link l adjusts the price λ_l in the opposite direction to the gradient of (4), namely

$$\lambda_l(t+1) = \left[\lambda_l(t) - \gamma \left(c_l - \sum_{r: l \in L_r} x^*_{s,r}(\lambda) \right) \right]^+, \tag{6}$$

and then announces the updated price to flows, where t is the iteration index, $\gamma > 0$ is the search step size, and $[\cdot]^+$ denotes the projection onto the nonnegative orthant. This cycle repeats and it will ultimately converge to the dual optimal solution λ^*, hence the primal optimal solution $\mathbf{x}^*(\lambda^*)$, provided that γ is sufficiently small and $U_s(\cdot)$ satisfies some mild conditions [12]. However, because of the particularity of multipath congestion control, it is needed to improve the above iterative process. Before explaining our motivations, we first present the necessary conditions satisfied by the optimal solution of (3) as follows.

Proposition 1. *Suppose flow s has $n > 0$ paths and, given $\lambda \geq \mathbf{0}$, the corresponding path prices are sorted in ascending order: $q_1 = \cdots = q_m < q_{m+1} \leq \cdots \leq q_n$. Then the optimal solution $x^*_{s,r}(\lambda)$ of* (3) *satisfies*

$$U_s^{'} \left(\sum_{r=1}^{m} x^*_{s,r}(\lambda) \right) - q_1 = 0, \tag{7}$$

$$x^*_{s,r}(\lambda) = 0, \ r = m+1, \ \cdots, \ n, \tag{8}$$

where $U_s^{'}(\cdot)$ is the derivative of $U_s(\cdot)$.

Proof: See Appendix. ∎

This result shows that a flow tends to always use only the cheapest paths while giving up other expensive ones so as to maximize its utility. Note that the path price reflects the extent of congestion. So the behavior that every flow pours into the cheap paths will push up the price of those paths, and meanwhile, will make the price of the previously expensive paths decline. At the equilibrium point, all the paths used by a flow will eventually have the same price, or in other words, will become equally congested, if possible [2]. Also note that the concavity of the utility function guarantees the fairness of the optimal solution of (1). Therefore, we arrive at the following conclusion.

Corollary 1 (Congestion Equality Principle)**.** *In the model* (1)*, if every flow strives to equalize the extent of congestion that it perceives on all its available paths by means of shifting traffic, then network resources will be fairly and efficiently shared by all the flows.*

Now we return to the issue about why and how to modify the iterative process. The motivation comes from two observations on Prop. 1. First, at each step of the iterative process, every flow turns off all its paths except for the least congested ones. This behavior is too drastic and also impractical in real networks, because congestion signals are commonly measured by sources only with the aid of traffic in most protocol implementations. Thus, unless restarting the closed paths, a flow has no way to perceive any subsequent congestion signals on those paths even if those paths become under-utilized. Second, a flow can not determine a unique solution of (3) at receiving the same price on multiple paths. It is not quite reasonable to randomly choose one from all the optimal candidates. Therefore, we need to develop an algorithm for smoothly adjusting transmission rates on the side of sources.

B. An Approximate Iterative Algorithm

Our basic idea is that at each iteration step, the flow calculates an approximate solution $x_{s,r}(\lambda)$ of (3), instead of the optimal solution $x^*_{s,r}(\lambda)$, by means of advancing a distance in direction to the gradient of

$$G_s(\mathbf{x}) := U_s \left(\sum_{r \in R_s} x_{s,r} \right) - \sum_{r \in R_s} q_r x_{s,r} \tag{9}$$

with taking the current transmission rates as the starting points, and then broadcasts the new rates to links. The approximate solution does not destroy convergence of the iterative process because the evolution of rates follows the Congestion Equality Principle. That is to say, the rates of a flow tend to decline on more congested paths (expensive paths) while tending to increase on less congested ones (cheap paths). As t goes to infinity, the dual optimal solution λ^* and the primal optimal solution $\mathbf{x}^*(\lambda^*)$ will both be reached.

[2]Theoretically, a flow will ultimately give up the path whose price is always higher than that of its other paths. In practice, it is more reasonable to put a little bit of traffic on those expensive paths since they might become cheap in the future. See Section IV for more details.

Specifically, since

$$\frac{\partial}{\partial x_{s,r}} G_s(\mathbf{x}) = U_s^{'}(y_s) - q_r, \tag{10}$$

flow s uses

$$x_{s,r}(t+1) = \left[x_{s,r}(t) + \theta\left(U_s^{'}(y_s) - q_r\right)\right]^+, r \in R_s, \tag{11}$$

to update its rates at each iteration step, and then broadcasts the new rates to links, where $\theta > 0$ is the positive step size. Accordingly, on the side of links, $x^*_{s,r}(\lambda)$ in (6) should be replaced by $x_{s,r}(\lambda)$, namely

$$\lambda_l(t+1) = \left[\lambda_l(t) - \gamma\left(c_l - \sum_{r:l\in L_r} x_{s,r}(\lambda)\right)\right]^+. \tag{12}$$

Equations (11) and (12) together constitute the approximate iterative algorithm for solving the problem (5).

Next, we would like to emphasize the physical significance of (11). Specifically, $U_s^{'}(y_s)$ can be interpreted as the expected path price as though flow s transmitted data at rate y_s along a single path. If the current price of path r is higher than $U_s^{'}(y_s)$, the rate will decrease; otherwise, it will increase. As a consequence, traffic always moves from more congested paths to less congested ones and thus the extent of congestion on each path used by a flow tends to be equal.

The Congestion Equality Principle and the approximate iterative algorithm together establish a general framework for designing an algorithm of multipath congestion control. In the next section, we will use packet queuing delay to estimate the extent of congestion and develop a practical congestion control algorithm by instantiating (11).

III. Weighted Vegas

In this section, we provide a delay-based congestion control algorithm for MPTCP, named weighted Vegas (wVegas), which is originated from TCP-Vegas [8].

A. The Weight Adjustment Algorithm of wVegas

On each path, wVegas works in the same way as TCP-Vegas. Briefly speaking, wVegas calculates

$$diff = \left(\frac{cwnd}{baseRTT} - \frac{cwnd}{rtt}\right) \cdot baseRTT \tag{13}$$

on the end of every round during the phase of congestion avoidance, where $cwnd$ is the congestion window, rtt is the average RTT in the last round and $baseRTT$ is the minimal RTT that has been measured so far. If $diff > \alpha$, $cwnd$ is increased by one packet. If $diff < \alpha$, $cwnd$ is decreased by one packet. We presume TCP-Vegas is well-known and thus pay special attention to how to define the weight of subflows and how to adjust weights for shifting traffic. We first derive the utility function of wVegas, and then define the weight of subflows, and finally propose the weight adjustment algorithm.

We know that $diff$ converges to α at the equilibrium point [13]. Thus, by substituting $q_r = rtt - baseRTT$ and $x_{s,r} = cwnd/rtt$ into (13) and replacing $diff$ by $\alpha_{s,r}$, we have

$$x_{s,r} = \frac{\alpha_{s,r}}{q_r}, r \in R_s, \tag{14}$$

Algorithm 1: The weight adjustment algorithm

Initialization at $t = 0$**:**
- Flow s sets the initial transmission rates to its subflows;
- **for** $r \in R_s$ **do**
  - $k_{s,r}(t) \longleftarrow x_{s,r}(t) \Big/ \sum_{i\in R_s} x_{s,i}(t)$;
- Flow s broadcasts the transmission rates of subflows to links;

At time $t = 1, 2, \cdots$**:**
- Flow s receives prices from paths;
- **for** $r \in R_s$ **do**
  - $\alpha_{s,r}(t) \longleftarrow k_{s,r}(t-1)\alpha_s$;
  - **if** $q_r \neq 0$ **then**
    - $x_{s,r}(t) \longleftarrow \alpha_{s,r}(t)/q_r$;
  - **else**
    - $x_{s,r}(t) \longleftarrow x_{s,r}(t-1) + 1$;
  - $k_{s,r}(t) \longleftarrow x_{s,r}(t) \Big/ \sum_{i\in R_s} x_{s,i}(t)$;
- Flow s broadcasts the transmission rates of subflows to links;

where $\alpha_{s,r}$ can be interpreted as the number of packets that flow s expects to backlog in link queues of path r. By Prop. 1, since all the *working* paths of flow s have the same price, denoted as q_s, we have

$$y_s = \sum_{r\in R_s} \frac{\alpha_{s,r}}{q_r} = \frac{1}{q_s} \sum_{r\in R_s} \alpha_{s,r} = \frac{\alpha_s}{q_s}, \tag{15}$$

where α_s is the total number of packets backlogged in the network for flow s. Then from (7), (8) and (15), the utility function of wVegas can be solved as follows:

$$\begin{aligned} U^{'}(y_s) &= q_s = \frac{\alpha_s}{y_s}, \\ U(y_s) &= \alpha_s \log y_s. \end{aligned} \tag{16}$$

Clearly, the function (16) is increasing, twice continuously differentiable and strictly concave for y_s. Consider

$$\theta = \frac{x_{s,r}(t)}{q_r}. \tag{17}$$

Substituting (17) into (11) yields

$$x_{s,r}(t+1) = \frac{U_s^{'}(y_s)}{q_r} x_{s,r}(t), r \in R_s. \tag{18}$$

Recall that $U_s^{'}(y_s)$ can be interpreted as the expected path price. So (18) follows the Congestion Equality Principle. Substituting (16) into (18), we have

$$x_{s,r}(t+1) = \frac{x_{s,r}(t)}{y_s} \cdot \frac{\alpha_s}{q_r} = \frac{k_{s,r}(t)\alpha_s}{q_r}, r \in R_s, \tag{19}$$

where

$$k_{s,r}(t) = \frac{x_{s,r}(t)}{y_s} \tag{20}$$

is defined as the *weight* of flow s on path r. Considering the case where $q_r = 0$, we further define

$$x_{s,r}(t+1) = x_{s,r}(t) + 1, r \in R_s \tag{21}$$

as the supplementary rule for updating rates when $q_r = 0$.

The weight adjustment algorithm of wVegas is given by Algo. 1. Note that the initial weights do not matter much to convergence, though they could have effect on the position of the equilibrium point in the case where the optimal solution is not unique. The pseudo-code of wVegas will be given in the next section.

B. Discussion

1) The Nature of wVegas: Equations (19) and (20) reveal the nature of wVegas. Specifically, flow s adjusts its rate on path r by means of tweaking the parameter $\alpha_{s,r} = k_{s,r}(t)\alpha_s$, namely the number of packets that flow s expects to backlog in link queues of path r. For flow s, the total number of the expected backlogged packets is fixed at α_s, a preconfigured parameter of wVegas. α_s is allocated to all the subflows of flow s. The share on path r is indicated by the weight $k_{s,r}(t)$ that is further determined by the proportion for which the current equilibrium rate $x_{s,r}(t)$ accounts of the current total equilibrium rate y_s. We know that when many flows are competing for a bottleneck link, the bandwidth obtained by each flow is proportional to its occupied buffer size in the link queue. Since α_s is a constant and $\alpha_{s,r} = k_{s,r}(t)\alpha_s$, the weight $k_{s,r}(t)$ can be regarded as the normalized $\alpha_{s,r}$ and thus quantifies the aggressiveness of competition for bandwidth. According to the definition of weights (20), the subflow that can transmit data at a higher equilibrium rate will obtain a larger weight. Thus, traffic is shifted from more congested paths to less congested ones.

2) The Domino Effect: One of the key ideas of multipath congestion control is to couple the rate adjustment process on each subflow together by means of a specially designed algorithm so as to achieve traffic shifting. Thus, the lost bandwidth on a path due to congestion events can be compensated by increasing rates of other subflows. As a result, a congestion event occurring in one place might cause flows in other places to change rates. This phenomenon is somewhat similar to the domino effect. We provide such an example in Section V.

3) Measurement Accuracy of $baseRTT$: The measurement accuracy of the minimal packet propagation delay, also called $baseRTT$, has much effect on the effectiveness of wVegas, because the measurement error on $baseRTT$ might lead to the evolution of weights deviating from the correct direction hence arriving at an undesirable equilibrium state. Leith *et al.* [14] proposed an effective method to improve the measurement accuracy of $baseRTT$. We incorporate their method into wVegas with some minor modifications. Please see Section IV for more details.

C. Another Perspective on wVegas

Essentially, multipath congestion control can be regarded as a kind of traffic engineering at end systems. As an economical user, the multihomed host prefers to use the cheapest (least congested) paths, if it has multiple options available, so as to maximize its utility. The behavior of traffic shifting follows the Congestion Equality Principle. For wVegas, the weight is the knob to shifting traffic among subflows. If we view the weight as some kind of resources whose total number is a unit for each flow, then, in the context of wVegas, the network utility maximization model (1) can be transformed into the weight allocation model as follows.

Algorithm 2: The iterative algorithm for solving WAP

Input: Initial weights that satisfy $k_{s,r} > 0$ **and** $\sum_{r \in R_s} k_{s,r} = 1$.
Output: The weights that can solve WAP.

1: **repeat**
2: Solve the problem (22) so as to obtain the optimal rates;
3: Use (24) to update the weights;
4: **until** *The changes to weights are less than the predetermined threshold*;

Definition 1 (Weight Allocation Problem (WAP))**.** *Given a network topology and the flows, find a weight allocation scheme for the problem*

$$\begin{aligned} &\max_{\mathbf{x} \geq \mathbf{0}} \sum_{s \in S} \sum_{r \in R_s} k_{s,r} \log x_{s,r} \\ &s.t. \quad \mathbf{A}\mathbf{x} \leq \mathbf{c}, \end{aligned} \tag{22}$$

such that all the paths used by each flow have the same price, or, in other words, become equally congested.

Recall that $U_s^{'}(y_s)$ can be interpreted as the expected path price for flow s and q_r is the current price on path r. It is reasonable to use

$$k_{s,r}(t+1) = \frac{U_s^{'}(y_s)}{q_r} k_{s,r}(t), r \in R_s \tag{23}$$

to iteratively search the desired weight allocation scheme. The algorithm is given by Algo. 2.

Since each flow has a unit of weights for allocation, we set α_s to be one. Substituting (16) into (23) yields

$$k_{s,r}(t+1) = \frac{1}{y_s} \cdot \frac{k_{s,r}(t)}{q_r} = \frac{x_{s,r}(t)}{y_s}, r \in R_s, \tag{24}$$

which has the same form with the definition of weights (20) except for the iteration index.

IV. Implementation

This section primarily focuses on the implementation of the weight adjustment algorithm since the implementation of TCP-Vegas is well known. Because of multiple available paths, wVegas uses the arrays indexed by the identifier of subflows to record state variables. The pseudo-code is given by Algo. 3.

First of all, wVegas uses the average RTT measured in the last round (line 8 in Algo. 3), instead of the smoothed RTT, as the current RTT on path r in order to quickly respond to changes of network congestion. The array $equilibrium\_rates$ records the saturated rates of subflows, which are prepared for calculating weights (line 27–31). Note that if a subflow is experiencing packet losses, then its equilibrium rate variable is reset to zero (line 33) so as to have no effect on the weight of other subflows. For each subflow, $equilibrium\_rates$ is updated only when $diff$ is no less than $alpha$ (line 10, 11). The reasons are two-fold. Firstly, this condition guarantees the

equilibrium rate is not under-estimated. The over-estimation is acceptable since the instantaneous rate is decreasing and will ultimately reach the equilibrium point. Secondly, compared with the condition that $diff$ must be equal to $alpha$, the "no-less-than" condition makes wVegas more quickly tweak weights hence accelerating convergence.

To improve the accuracy of $baseRTT$, we incorporate the method in [14] into wVegas (line 23, 24). The idea is to make $cwnd$ back off once detecting the queuing delay is larger than some threshold, so that the bottleneck link can drain off the backlogged packets. And thus all the flows involved have a chance to obtain the more accurate propagation delay. Instead of configuring a constant threshold [14], [15], wVegas adopts an adaptive method to determine when to back off. Specifically, the array $queue\_delays$ records the minimal queuing delay (line 19–21, 25, 34) measured after the last backoff. When the current queuing delay is several times larger than $queue\_delays$ (line 22), $cwnd$ is decreased by a factor.

It is possible that the weight of some paths tends to zero. Given that those paths might be idle in the future, wVegas sets a lower bound for the parameter $alpha$ (line 14). However, this is an open issue.

Note that all the constants in Algo. 3 are configurable. For example, we set the parameter $total\_alpha$, namely α_s, to be 10 packets (line 2) and the initial value of $alpha$ to be 2 packets (line 4). Our simulations show that these settings work well. Besides, Algo. 3 do not involve the slow-start phase as well as any process of error handling, since these are the same as TCP-Vegas.

V. Evaluation

We implemented MPTCP and the two congestion control algorithms [3], wVegas and Linked Increases [6], in NS-3 [16]. We mainly focused on the fairness and efficiency of traffic shifting. The common simulation parameters are given in Table I, unless otherwise indicated. Note that the sending/receiving buffer size is set to be sufficiently large for each subflow so that the transmission rate is limited only by the congestion control window. We configured the type of link queues to be DropTail. The delay acknowledgement mechanism was also disabled. For brevity and clarity, in this section, we omit the unit of bandwidth, namely bps, and identify flows by the number of corresponding sources. Moreover, subflows are numbered, starting at 1, in sequence from left to right or from top to bottom.

The computational overhead of wVegas is inexpensive, though it involves floating-point division. This is because the frequency of weight adjustment is one time per RTT for each subflow and the number of paths used by a flow is also small in most cases. Additionally, there exist many approximation methods that can convert floating-point calculation to integer operations. So we believe the overhead of wVegas is negligible in the condition of modern hardware technology.

[3]Since the performance of CMT/RPv2 [5] is comparable to Linked Increases, we implemented only one of them.

Algorithm 3: The pseudo-code of wVegas

```
1:  Initialization:
2:      total_alpha ← 10 ; // namely α_s
3:      for r ∈ R_s do
4:          alpha[r] ← 2;
5:          equilibrium_rates[r] ← 0;
6:          queue_delays[r] ← 0;

7:  On the end of round for subflow r:
        /* average RTT estimated in the last round */
8:      rtt ← sampled_rtts[r]/sampled_num[r];
9:      diff ← cwnd[r] × (rtt − baseRTT[r])/rtt;
        /* tweak weights and alphas                 */
10:     if diff ≥ alpha[r] then
11:         equilibrium_rates[r] ← cwnd[r]/rtt;
12:         Adjust_Weights();
13:         alpha[r] ← weights[r] × total_alpha;
14:         alpha[r] ← max{2, alpha[r]} ; // lower bound
        /* window adjustment                        */
15:     if diff < alpha[r] then
16:         cwnd[r] ← cwnd[r] + 1;
17:     else if diff > alpha[r] then
18:         cwnd[r] ← cwnd[r] − 1;
        /* try to drain link queues if needed       */
19:     q ← rtt − baseRTT[r] ; // current queuing delay
20:     if queue_delays[r] = 0 or queue_delays[r] > q then
21:         queue_delays[r] ← q;
22:     if q ≥ 2 × queue_delays[r] then
23:         backoff_factor ← 0.5 × baseRTT[r]/rtt;
24:         cwnd[r] ← cwnd[r] × backoff_factor;
25:         queue_delays[r] ← 0;
26:     cwnd[r] ← max{2, cwnd[r]}; // lower bound

27: Adjust_Weights():
28:     total_rate ← Σ equilibrium_rates;
29:     for r ∈ R_s do
30:         if equilibrium_rates[r] ≠ 0 then
31:             weights[r] ← equilibrium_rates[r]/total_rate;

32: On packet loss for subflow r:
33:     equilibrium_rates[r] ← 0;
34:     queue_delays[r] ← 0;
```

TABLE I
Common Simulation Parameters

Sending Buffer	3200pkts	Packet Size	1000B
Receiving Buffer	6400pkts	RTT	100ms
Link Queue	50pkts	$total\_alpha$	10pkts

A. Validation on Algo. 2

By solving the problem (1) with automatic tools such as MATLAB, we easily know the optimal solution in Fig. 2(a) is that S2 gets 1.8972M on the top path and 1.2056M on the bottom while S1, S3 and S4 get 3.1028M, 8.7944M and 4.7944M, respectively. Algo. 2 can also output the same solution in an iterative way. As shown by Fig. 3(a), all the rates of subflows are quite close to the corresponding optimal values after about 20 steps and the path prices of S2 tend to be equal. Note that initial weights are insignificant for convergence. We deliberately set the initial weight of Flow 2-1 to be much less than that of Flow 2-2 for the purpose of demonstrating the effectiveness of wVegas. For comparison,

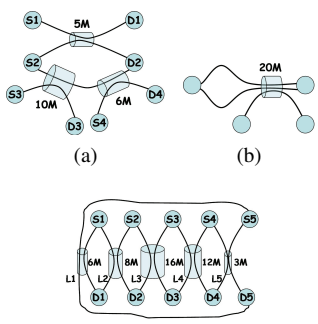
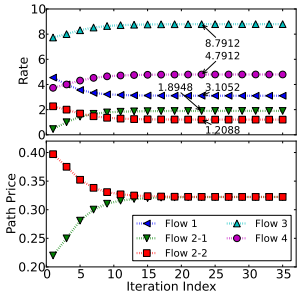
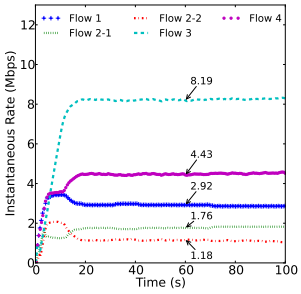


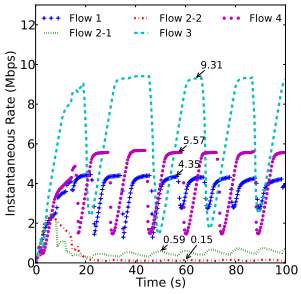
Fig. 2. The network topologies



(a) The outcome of Algo. 2 with the initial weights: S2(0.1, 0.9).



(b) wVegas



(c) Linked Increases

Fig. 3. The simulation in the topology of Fig. 2(a)

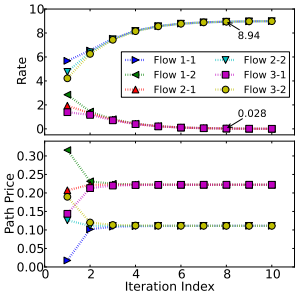
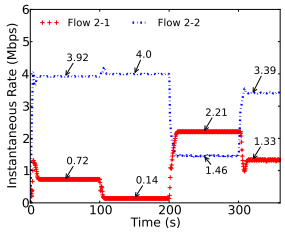
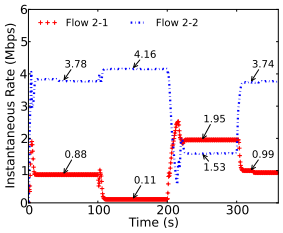


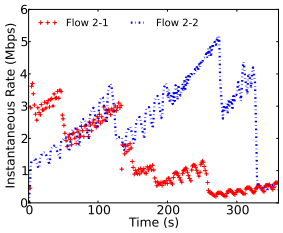
Fig. 4. The outcome of Algo. 2 in Fig. 1(b) with the initial weights: S1(0.1, 0.9), S2(0.4, 0.6) and S3(0.2, 0.8).



(a) wVegas, $\alpha_s = 5$



(b) wVegas, $\alpha_s = 15$



(c) Linked Increases

Fig. 5. Traffic shifting in the topology of Fig. 1(a)

Fig. 3(b) and (c) show the simulation results of NS-3. The instantaneous rates are estimated by dividing the congestion control window by the average RTT measured in a round. Clearly, the rates of wVegas are more stable and closer to the optimal values. For Linked Increases, the rates of Flow 2-1 and Flow 2-2 are much less than their fair shares. This is because link buffers are occupied by few flows in the case of low statistical multiplexing, resulting in synchronization. This phenomenon means Linked Increases might require routers to run appropriate Active Queue Management algorithms so as to achieve the desirable performance. Incidentally, the spikes appearing in Fig. 3(c) are caused by packet losses due to the depletion of link buffers.

Fig. 4 shows the iterative outcome in the topology of Fig. 1(b). The initial weights are randomly generated. As expected, wVegas can achieve the most efficient bandwidth sharing. Because the one-hop path always has a smaller queuing delay than that of the two-hop path, each flow would like to use the former while giving up another one.

B. Traffic Shifting

We use the topology in Fig. 1(a) to evaluate the effectiveness of wVegas in terms of traffic shifting. Specifically, Flow 1, Flow 2 and Flow 3 are started at 0s simultaneously. Then Flow 4 comes up on the top path at 100s, which will force Flow 2-1 to decrease its rate. As compensation, Flow 2-2 will increase the rate. Next, Flow 4 stops at 200s, and meanwhile, Flow 5 [4] begins to run on the bottom path. So Flow 2 has to adjust its rates again. Finally, Flow 5 quits at 300s and all the other flows stop at 360s.

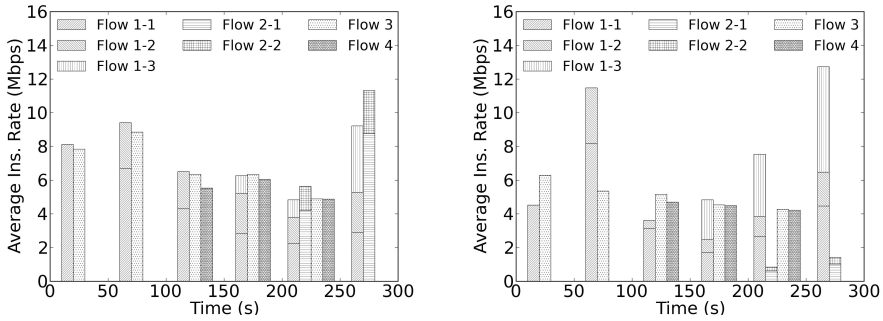
As showed in Fig. 5, wVegas can quickly complete traffic shifting, since it is more sensitive to changes of network congestion than Linked Increases. The different values of α_s produce the similar results. A larger α_s means more packets will be backlogged in link queues, thus leading to a larger RTT. On the other hand, a quite small α_s may cause inaccurate congestion detection and adversely affect performance of data transfer. We think $\alpha_s = 10$ is a good choice and use it as the default setting in our simulations.

TABLE II
AVG. INS. RATES OF FLOW 2 IN FIG. 1(A)

Interval	0–100s	100–200s	200–300s	300–360s
Optimal	1.00, 4.00	0.00, 4.50	2.25, 1.50	1.00, 4.00
wVegas	1.08, 3.53	0.15, 4.19	1.93, 1.62	0.97, 3.64
Linked	2.59, 1.89	1.65, 2.69	0.67, 3.83	0.45, 2.01

For convenience of comparison, we list in Table II the average instantaneous rates and the corresponding optimal rates of Flow 2 during each interval. Each item in the table consists of two floating-point numbers separated by a comma, which are the rates of Flow 2-1 and Flow 2-2, respectively.

[4]Flow 4 and Flow 5 are not painted in Fig. 1(a).



(a) wVegas (b) Linked Increases

Fig. 6. Fairness on bottleneck links in the topology of Fig. 2(b)

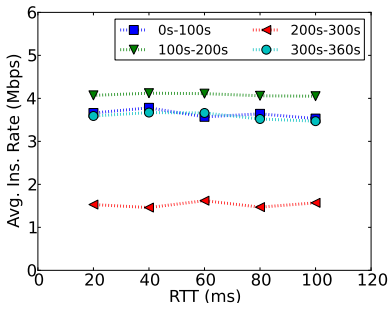


Fig. 7. The variation of RTTs has little effect on wVegas.

Note that, for wVegas, the rate of Flow 2-1 decreases to a quite low level (about 0.15) from 100s to 200s, rather than to zero. This is because wVegas sets a lower bound to the parameter $alpha$ (see line 14 in Algo. 3). Compared with Linked Increases, the rates of wVegas are closer to the optimal values.

wVegas attempts to backlog fewer packets in link queues, thus stabilizing links into a fully-utilized state with fewer losses. This property facilitates wVegas to cope with the variation of RTTs. To test this, we repeated the simulation in Fig. 1(a) with the RTT of Flow 2-2 being various values. The result is shown in Fig. 7. Clearly, the variation of RTTs has little effect on wVegas.

C. Fairness on Bottleneck Links

For multipath transfer, a natural concern is that if several subflows belonging to a flow pass through the same bottleneck link, then whether this flow would steal bandwidth from others. We use the network topology similar to Fig. 2(b) to validate the intra-protocol fairness of wVegas. Specifically, there are four flows numbered from 1 to 4, competing for one bottleneck link with capacity of 20M. Flow 1 has three subflows and Flow 2 has two subflows, while Flow 3 and Flow 4 are both single-path flows. For the convenience of comparison, we show in Fig. 6 the average instantaneous rates of the two algorithms during each interval of 50s. For wVegas, Flow 1-1 and Flow 3 evenly share link capacity from 0s to 50s. Then, from 50s to 100s, the rate of Flow 3 keeps roughly unchanged, though Flow 1-2 adds to Flow 1. Next, from 100s to 150s, due to the existence of Flow 4, every flow relinquishes a part of bandwidth to the newcomer, and consequently the link capacity is still fairly shared. From 150s to 200s, though Flow 1 initiates the third subflow (Flow 1-3), it can not steal bandwidth from Flow 3 and Flow 4. Next, Flow 2 is started with two subflows at 200s, so each flow obtains roughly 5M bandwidth. Finally, after Flow 3 and Flow 4 quit at 250s, the link capacity is fairly shared by Flow 1 and Flow 2.

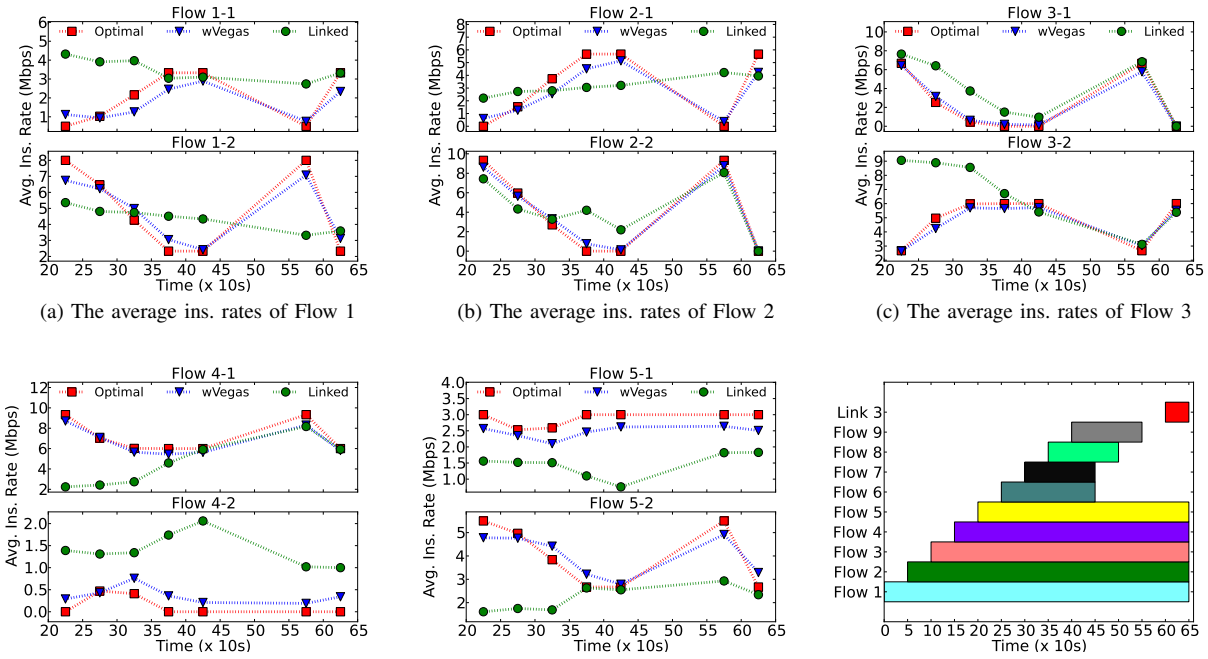
In contrast, Fig. 6(b) shows that the performance of Linked Increases is instable with respect to fairness. This is because the previously initiated flows usually occupy more link buffers than the subsequently started flows and thus obtains more bandwidth. Moreover, every flow does not voluntarily decrease transmission rates to meet demands of others unless packet losses occur. As a result, link capacity is hard to be fairly shared by every one.

D. The Domino Effect

Due to the rate complementation between subflows, a congestion event occurring in one place may cause flows in other places to change transmission rates. We construct a slightly complicated scenario to demonstrate this effect. In Fig. 2(c), if link L3 becomes increasingly congested or even fails, then not only S2 and S3 will decrease their transmission rates, but also S1, S4 and S5 will respond to the congestion events.

Specifically, there are five flows numbered from 1 to 5 starting one by one with a time interval of 50s. Then, after 250s, we continue to add background flows to link L3 in order to generate congestion events. Finally, a failure occurs on L3 at 600s. The details of start/stop times of each flow are shown in Fig. 8(f), while the simulation results are given in other five subfigures. The flows numbered from 6 to 9 are background flows passing through L3. We plot the average instantaneous rates of each flow during each interval of 50s from 200s to 650s. The optimal values are obtained by solving the problem (1). As expected, Flow 2-2 and Flow 3-1 continue to decrease their rates with L3 becoming more and more congested from 200s to 450s. As compensation, Flow 2-1 and Flow 3-2 gradually increase their rates. These actions then further force the rates of Flow 1-2 and Flow 4-1 to decline. By this kind of interaction between subflows, the influence of a congestion event is spread from its occurring position to other places in the network. After 450s, the background flows quit one by one, so the rates of each flow gradually recover to the values prior to 250s.

We can obtain three interesting observations from Fig. 8. First, the curves of wVegas are more consistent with the optimal ones than those of Linked Increases. Second, for a flow, if the curve of one subflow is concave, then another is convex, and vice versa. As mentioned before, this phenomenon is produced due to the property of rate complementation



(a) The average ins. rates of Flow 1

(b) The average ins. rates of Flow 2

(c) The average ins. rates of Flow 3

(d) The average ins. rates of Flow 4

(e) The average ins. rates of Flow 5

(f) The start/stop time of each flow

Fig. 8. The domino effect

between subflows. Third, the curves of Flow 4-2 and Flow 5-1 have a relatively flat slope. The reasons are two-fold. On one hand, link L5 is far away from L3, so it is less influenced by congestion events. On the other hand, the capacity of L5 is quite small thereby playing a minor role in bandwidth reallocation process.

VI. Related Work

Many protocols were proposed in an attempt to transfer data through multiple paths in parallel. pTCP [17], [18] allows a connection to utilize the aggregate bandwidth offered by multiple paths, and it assumes the wireless link is the bottleneck to ensure fairness. The work in [19] improves the fairness of parallel TCP in under-utilized networks by using a long virtual round trip time. mTCP [20] focuses on detecting shared congestion at bottleneck links by computing the correlation between fast retransmit intervals on different paths. cTCP [21] provides a single congestion window for all the paths and maintains a database at senders to record the relationship between packet sequences and the paths for the purpose of detecting losses. cTCP uses loss probability to estimate path capacity so as to put more packets on high bandwidth paths. CMT-SCTP [2] improves SCTP for the purpose of multipath transfer in parallel.

However, most of the above schemes perform uncoupled congestion control, similar to TCP-Reno, on each path, thus neither of them can achieve flexible load balancing. As one of the next generation transport protocols, MPTCP [1] incorporates many lessons learned from previous research efforts and development practice. MPTCP adopts a novel coupled congestion control algorithm, named Linked Increases [6], [22]. Briefly speaking, this algorithm increases the total congestion window by one packet only when the outstanding packets issued on every path are all acknowledged. The incremental share of the congestion window on each subflow is proportional to its current congestion window size. CMT/RPv1 [4] and CMT/RPv2 [5] also adopt the similar way.

In theoretical efforts on multipath congestion control, Kelly *et al.* [23] presented a sufficient condition for the local stability of end-to-end algorithms. Han *et al.* [24] proposed a class of algorithms derived from differential equation models, and also proved their stability. Wang *et al.* [25] developed two distributed algorithms to maximize the aggregate source utility.

VII. Conclusions and Future Work

Based upon the network utility maximization model, we proved the Congestion Equality Principle, and proposed an approximate iterative algorithm for solving the problem of multipath congestion control. These two components together establish a general framework for designing an algorithm of multipath congestion control. Using this framework, we developed wVegas and evaluated its performance in terms of fairness and efficiency.

Just as TCP-Vegas and TCP-Reno, wVegas and Linked Increases have their own respective advantages and defects. Thus they can complement each other in practice. Furthermore, we expect to combine the two algorithms together so as to cope with multiple long high-speed paths efficiently. In this regard, Compound TCP [26] provides a very good instance. In future work, we plan to investigate this issue. Besides, whether or

not to shut down seriously congested paths is also an open issue.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (61073166), the National Basic Research Program of China (973 Program) under Grant 2012CB315803, the National High-Tech Research and Development Program of China (863 Program) under Grants 2011AA01A101, and the National Science & Technology Pillar Program of China under Grant 2011BAH19B01.

REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC 6182, IETF, Mar. 2011.

[2] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Transactions on Networking*, vol. 14, no. 5, pp. 951–964, Oct. 2006.

[3] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. of ACM SIGCOMM*, 2011, pp. 266–277.

[4] T. Dreibholz, M. Becke, J. Pulinthanath, and E. P. Rathgeb, "Applying TCP-Friendly Congestion Control to Concurrent Multipath Transfer," in *Proc. of IEEE AINA*, 2010, pp. 312–319.

[5] T. Dreibholz, M. Becke, H. Adhari, and E. Rathgeb, "On the impact of congestion control for Concurrent Multipath Transfer on the transport layer," in *Proc. of IEEE ConTEL*, june 2011, pp. 397–404.

[6] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. of USENIX NSDI*, 2011, pp. 8–8.

[7] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 47–52, Sep. 2008.

[8] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proc. of ACM SIGCOMM*, 1994, pp. 24–35.

[9] J. Mo, R. La, V. Anantharam, and J. Walrand, "Analysis and comparison of TCP Reno and Vegas," in *Proc. of IEEE INFOCOM*, vol. 3, 1999, pp. 1556–1563.

[10] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and Effective Fine-grained TCP Retransmissions for Datacenter Communication," in *Proc. of ACM SIGCOMM*, 2009, pp. 303–314.

[11] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[12] S. H. Low and D. E. Lapsley, "Optimization flow control-I: basic algorithm and convergence," *IEEE/ACM Transactions on Networking*, vol. 7, no. 6, pp. 861–874, Dec. 1999.

[13] S. H. Low, L. Peterson, and L. Wang, "Understanding TCP vegas: a duality model," in *Proc. of ACM SIGMETRICS*, 2001, pp. 226–235.

[14] D. Leith, R. Shorten, G. McCullagh, L. Dunn, and F. Baker, "Making Available Base-RTT for Use in Congestion Control Applications," *IEEE Communications Letters*, vol. 12, no. 6, pp. 429–431, Jun. 2008.

[15] D. Leith, R. Shorten, G. McCullagh, J. Heffner, L. Dunn, and F. Baker, "Delay-based AIMD congestion control," in *Proc. of PFLDNeT Workshop*, 2007.

[16] The NS-3 simulator. [Online]. Available: http://www.nsnam.org/

[17] H. Y. Hsieh and R. Sivakumar, "pTCP: an end-to-end transport layer protocol for striped connections," in *Proc. of IEEE ICNP*, 2002, pp. 24–33.

[18] ——, "A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts," in *Proc. of ACM MobiCom*, 2002, pp. 83–94.

[19] T. J. Hacker, B. D. Noble, and B. D. Athey, "Improving Throughput and Maintaining Fairness Using Parallel TCP," in *Proc. of IEEE INFOCOM*, vol. 4, 2004, pp. 2480–2489.

[20] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson, and R. Wang, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *Proc. of USENIX Annual Technical Conference*, 2004, pp. 99–112.

[21] Y. Dong, D. Wang, N. Pissinou, and J. Wang, "Multi-Path Load Balancing in Transport Layer," in *Proc. of 3rd EuroNGI Conference on Next Generation Internet Networks*, 2007, pp. 135–142.

[22] D. Wischik, M. Handley, and C. Raiciu, "Control of Multipath TCP and Optimization of Multipath Routing in the Internet," in *Proc. of NET-COOP*, 2009, pp. 204–218.

[23] F. Kelly and T. Voice, "Stability of end-to-end algorithms for joint routing and rate control," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 2, pp. 5–12, Apr. 2005.

[24] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in the internet," *IEEE/ACM Transactions on Networking*, vol. 14, no. 6, pp. 1260–1271, Dec. 2006.

[25] W. H. Wang, M. Palaniswami, and S. H. Low, "Optimal flow control and routing in multi-path networks," *Performance Evaluation*, vol. 52, no. 2–3, pp. 119–132, Apr. 2003.

[26] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-Speed and Long Distance Networks," in *Proc. of IEEE INFOCOM*, 2006, pp. 1–12.

APPENDIX
PROOF OF PROP. 1

Proof: According to Karush-Kuhn-Tucker Conditions, the optimal solution of (3) should simultaneously satisfies

$$\frac{\partial G_s(\mathbf{x})}{\partial x_{s,i}} = U_s^{'}\left(\sum_{r=1}^{n} x_{s,r}\right) - q_i \leq 0, \tag{25}$$

$$x_{s,i} \geq 0, \tag{26}$$

$$\frac{\partial G_s(\mathbf{x})}{\partial x_{s,i}} x_{s,i} = 0, \tag{27}$$

where $G_s(\mathbf{x})$ is given by (9) and $i = 1, 2, \cdots, n$.

Case 1: Suppose $q_1 = \cdots = q_n$. Because $U_s(0) = -\infty$, there exists at least one subflow, denoted as j, whose rate is positive. Thus, from (27), we have $\partial G_s(\mathbf{x})/\partial x_{s,j} = 0$ and hence $U_s^{'}(\sum_{r=1}^{n} x_{s,r}) - q_j = 0$. Since every path has the same price, Equation (7) holds.

Case 2: Suppose $n > 1$ and $q_1 = \cdots = q_m < q_{m+1} \leq \cdots \leq q_n$. If there is a subflow j such that $j > m$ and $x_{s,j} \neq 0$, then from (27) we have $U_s^{'}(\sum_{r=1}^{n} x_{s,r}) - q_j = 0$ which together with (25) yields $U_s^{'}(\sum_{r=1}^{n} x_{s,r}) = q_j > q_1 \geq U_s^{'}(\sum_{r=1}^{n} x_{s,r})$. The contradiction occurs. Therefore, $x_{s,j} = 0$ where $j > m$, so Equation (8) holds. According to Case 1, we know that Equation (7) also holds. ■