Justice: Flexible and Enforceable Per-Source Bandwidth Allocation

Jakob Eriksson, Michalis Faloutsos, and Srikanth Krishnamurthy

University of California, Riverside
{jeriksson, michalis, krish}@cs.ucr.edu

Abstract. The main goal of this work is to provide bandwidth allocation that is robust against the behavior of greedy or malicious users. The traditional solution, Fair Queueing, allocates capacity per source-destination pair in accordance with the max-min fairness criterion. While Fair Queueing, defined as above, has been successful and popular to a large extent, it does not prevent greedy or malicious users from getting unfair shares of capacity. In particular, it is vulnerable to end-points simply establishing *multiple parallel connections* to increase their allocated capacity.

In order to overcome this limitation, we propose Justice, which allows for robust, yet flexible bandwidth allocation in the Internet. Justice employs *weighted* per source bandwidth allocation to accommodate traffic sources with varying bandwidth requirements. We describe an efficient and scalable mechanism for determining, for each source s, the weight $\phi_k^{(s)}$ at any given link k. We demonstrate through analysis and simulation that Justice is flexible, efficient, scalable, and robust to all identified attacks related to bandwidth allocation.

Keywords: Bandwidth Allocation, Per-Source Weighted Fairness.

1 Introduction

Currently greed and mischief pay off in terms of performance in the Internet. Through a number of reasonably simple tricks, greedy or malicious users can improve their own throughput at the cost of their well-behaved counterparts. The primary goal of this work is to make the bandwidth allocation of each user independent of the potentially malicious behavior of other users of the network.

The problem of isolating user behavior has been addressed before. In particular, Fair Queuing was introduced in RFC 890 [1], where Nagle advocated per-source max-min bandwidth allocation. Shortly thereafter, Demers, Keshav and Shenker, [2] brought up the fact that some sources may deserve a higher bandwidth allocation than others, making a per-source max-min allocation impractical. Their proposed solution was to instead allocate bandwidth per source-destination pair. Unfortunately, this solution makes Fair Queuing vulnerable to users sending traffic to multiple destinations to maximize their total throughput. An ideal bandwidth allocation scheme needs to be both flexible enough to accommodate a wide variety of traffic sources, and enforceable so that a well-behaved source can retain its allocation in despite the actions of malicious or greedy sources.

We propose Justice[1], a novel approach to bandwidth allocation. Justice provides a guaranteed minimum allocation to each source, at each link, independent of the behavior of other users. With the protection of well-behaved users as its first concern, Justice employs weighted per-source fairness at each link, and provides a mechanism for efficiently determining a unique per-source weight $\phi_k^{(s)}$ of each source s, at each link k in the network.

Justice exhibits the following properties. First, it is **robust**: through the careful computation of per-source weights, Justice ensures that greedy or malicious sources cannot, through manipulation of the weight calculation process, affect the bandwidth allocation to their advantage. Second, it is **enforceable**: the use of per-source fairness at each link ensures that the actual bandwidth achieved corresponds closely to the allocation assigned, independent of the behavior of competing traffic sources. Third, it is **flexible**: Justice computes a unique per-source weight for each source at each link. This provides administrators with the flexibility to allocate extra bandwidth to sources that deserve it. Fourth it is **scalable**: Administration is limited to local per-link settings, as opposed to global per-source weights. Moreover, a router needs only keep track of the weights of sources that are actively using a link incident on it, the weights of inactive sources are ignored. These properties have previously been achieved in isolation, but to the best of our knowledge, Justice is the first to provide all of them together.

2 Background and Motivation

Much research has gone into improving bandwidth allocation in the Internet. It is well established that FIFO-Droptail queuing, where packets are served in arrival order, and arriving packets are dropped if they find the queue full on arrival, does not provide an adequate level of protection. Aside from the congestion control algorithm used in TCP [3], most of the work has been done at the router level.

Queue Management algorithms such as RED [4] with variations [5] [6] [7] and CHOKe [8] attempt to notify and punish overly aggressive flows. These techniques rely on having conforming TCP clients at the endpoints. Moreover, other link-based schemes, like Weighted Round-Robin queuing, or proportional random dropping suffer from similar problems, in particular in conjunction with TCP congestion control, where aggressive flows can often cause TCP to back off indefinitely [9].

An approach toward isolating user behavior which has received much attention is that of Fair Queuing [10, 11, 1, 2, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21]. In general, the goal of Fair Queuing is to provide max-min fairness between

[1] The name Justice has been used by us in previously submitted work. Please note that although the high-level goal is similar, the proposed scheme is significantly different.

flows, where a flow can be defined in several ways, and with the implicit assumption that all flows have equal rights to network resources. In the original RFC [1], Nagle proposed to provide fairness per source. However, in [2], the authors find that per-source fairness is impractical, as some hosts, such as file servers, *deserve* a larger allocation of bandwidth. They suggest that this problem could be solved by providing routers with knowledge about the *deserved* allocation of each source, an approach commonly referred to as *weighted per-source fairness*. However, they do not provide a mechanism to determine and distribute such knowledge. Instead, they propose to provide max-min fairness between *conversations*, or source-destination pairs. However, quoting [2]: *"Allocation per source-destination pair allows a malicious source to consume an unlimited amount of bandwidth by sending many packets all to different destinations."* This vulnerability of per-conversation bandwidth allocation has yet to be successfully addressed in the literature.

Link Sharing is a related technique, which focuses on sharing bandwidth between incoming links or different classes of traffic. Examples of this are Class Based Queuing [22], and Weighted Fair Queuing [23]. These are also referred to as Differentiated Services policies [24]. In general, such approaches provide protection between traffic classes or between incoming links, but are powerless to provide protection between sources in the same class, or the same link, respectively.

We conclude that current approaches are not sufficient to protect well-behaved users from their greedy or malicious counterparts. Due to the inherent vulnerability of per-conversation allocation, we propose to use weighted per-source allocation. To accommodate sources that deserve a higher allocation than others, we provide an efficient mechanism for determining the unique weight $\phi_k^{(s)}$ of source s at link k, thus solving the original problem with per-source Fair Queuing.

Like other schemes based on Fair Queuing, we assume that the source address in the packet header is accurate. Under certain circumstances, an attacker could potentially falsify the source address of his packets to improve his allocation. A countermeasure against source address falsification is described in [25].

3 Defining Justice

In this section we will describe Justice as a concept, leaving implementation details for later. Justice constitutes a new class of source-based bandwidth allocation policies, where the primary objective is to protect well-behaved sources from their malicious or greedy counterparts. At a high level, Justice combines a network-wide method for determining the appropriate per-source weight $\phi_k^{(s)}$ of each source s at any link k, with weighted per-source fairness[2] locally on each router.

[2] Weighted per-source fairness is a variation of per-flow fairness where all traffic from a given source is treated as a single flow, and where each source may have a unique configured weight, indicating its *deserved* allocation on a given link.

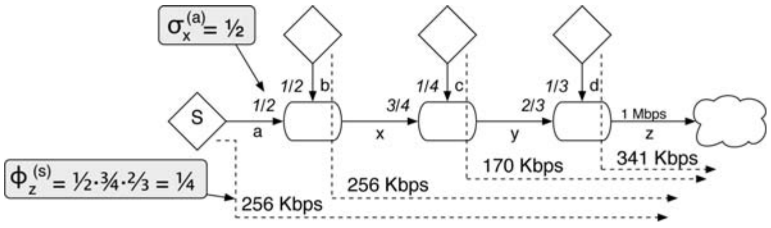


Fig. 1. Bandwidth allocation with Justice. The allocation of source s on link z, $\phi_z^{(s)}$, is determined by the per-link σ settings on the routers between s and z

Until now, weighted per-source bandwidth allocation has not been practical, because there has been no adequate mechanism for determining the appropriate weight of each source at each link, or for the distribution of this knowledge.

With Justice, we propose an efficient and scalable method for determining the weight $\phi_k^{(s)}$ of each source s at each link k, without the storage and administration overhead of configuring this weight at each router. The per-source weight is computed based on a set of configured per-link weights $\sigma_x^{(y)}$ along the path from the source to the link in question. Figure 1 shows an example of how bandwidth is allocated based on configured per-link weights. Here, link a has a weight $\sigma_x^{(a)} = \frac{1}{2}$ on the outgoing link x. Similar per-link weights are configured throughout the network. **Justice defines the per-source weight $\phi_k^{(s)}$ of source s at link k, as the product of the per-link weights along the path from s to k.** In this case, $\phi_z^{(s)} = \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{2}{3} = \frac{1}{4}$.

Note that the configured weight on each link could be changed arbitrarily, which would then result in a different bandwidth allocation. In general, Justice is very flexible with respect to the final allocation. For example, a naive approach to link weight assignment would be to assign the same weight to all links. This would result in an allocation where many users sharing a high capacity link are disadvantaged, as compared to a small number of users sharing a low capacity link. A more practical, but still naive, approach might be to assign weights according to link capacity. Finally, Justice also makes more advanced traffic engineering approaches possible, one of which we will describe in Section 5.

More formally, for each pair of links x,y incident on a router, a constant weight $\sigma_y^{(x)}$ is *configured*. We define $\sigma_y^{(x)}$ to be the fraction of the capacity of link y, that is to be the guaranteed minimum allocation for packets that arrive through link x. $\sigma_y^{(x)}$ is a value between 0 and 1, and for every outgoing link y

$$\sum_{x \neq y} \sigma_y^{(x)} = 1. \tag{1}$$

Given a path consisting of links $0, 1, \ldots, k$, we define the minimum weight $\phi_k^{(s)}$ of source s at link k, to be

$$\phi_k^{(s)} = \prod_{i=0}^{k-1} \sigma_{i+1}^{(i)} \tag{2}$$

Note that $\sigma_y^{(x)}$ describes the minimum per-link weight. Justice is work-conserving, so should packets arriving from an incoming link use less bandwidth than what that link's weight entitles it to, the surplus bandwidth on the outgoing link is distributed between the other incoming links in proportion to their respective weights, as discussed in the following section.

4 Enforcing Justice

In this section we will discuss how Justice can be practically implemented in a network. Enforcing Justice consists mainly of two challenges. First, an efficient mechanism for calculating the $\phi_k^{(s)}$ of each active source s on a link k is required. Second, it is necessary to ensure that each source actually gets its deserved $\phi_k^{(s)}$ share of the capacity of the link. The second part can be accomplished by using per-source Weighted Fair Queuing.

Described briefly, per-source Weighted Fair Queuing (WFQ) keeps a separate queue for each active traffic source, and sends packets from each queue in proportion to their respective weights. WFQ is well understood [23, 11], and has been shown to fairly distribute bandwidth between flows according to preconfigured weights. WFQ requires per-source state. In most routers, this is not a concern. However, in core routers, keeping per-source state may result in higher hardware costs. If per-source state in core routers is a concern, Justice can be implemented in a manner analogous to core-stateless Fair Queuing [21], as we report in [9]. This technique removes the need for per-source state in core routers, but is not described here due to space constraints. We will now focus on the efficient calculation and distribution of per-source weights.

We will now demonstrate how simple per-packet calculations can be used to compute the current per-source weight at any given link. Let us define $in_r(p)$ and $out_r(p)$ to mean the incoming and outgoing link of a packet p, at router r on the path from source to destination, and let $src(p)$ be the source of the packet. For every incoming packet p, a router needs an efficient and secure method to determine the share $\phi_{out(p)}^{(src(p))}$, that the packet's source $src(p)$ deserves on the outbound link, $out(p)$. From Eq. 2, we know that the weight of source $src(p)$ on the r:th outbound link $out_r(p)$ is

$$\phi_{out_r(p)}^{(src(p))} = \prod_{i=0}^{r} \sigma_{out_i(p)}^{(in_i(p))}. \tag{3}$$

It is easy to see that this can also be written as the recursion

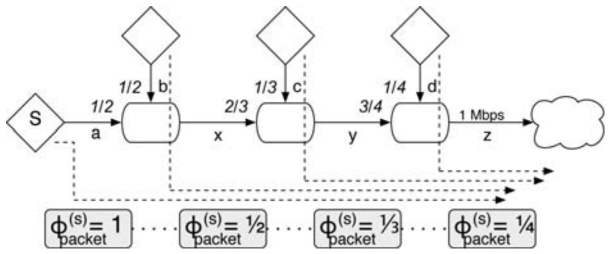

S
1/2
a
1/2
b
x
2/3
1/3
c
y
3/4
1/4
d
1 Mbps
z
$\phi^{(s)}_{packet} = 1$
$\phi^{(s)}_{packet} = 1/2$
$\phi^{(s)}_{packet} = 1/3$
$\phi^{(s)}_{packet} = 1/4$

Fig. 2. Iterative calculation of weights along a path from left to right. $\phi^{(s)}_{packet}$ indicates the weight of source s on the current outbound link, stored in the packet header

$$\phi^{(src(p))}_{out_r(p)} = \phi^{(src(p))}_{out_{r-1}(p)} \cdot \sigma^{(in_r(p))}_{out_r(p)} . \tag{4}$$

This observation suggests that the sequence of per-source weights along a path from source to destination can be computed iteratively, one hop at a time, as shown in Figure 2. In order to do this, we store weight of the source at the current link in the packet header. Let us call this value ϕ_{packet} Upon receiving a packet, router r can compute $\phi^{(src(p))}_{out_r(p)}$ by simply multiplying ϕ_{packet} by the incoming link's *configured* share of the next hop, $\sigma^{(in_r(p))}_{out_r(p)}$.

However, since the weight calculation now depends on receiving accurate ϕ_{packet} values from upstream routers, it is vulnerable to manipulation. A router could conceivably set the weights in its outgoing packets to artificially high values.This could inflate the downstream allocation of traffic passing through that router, giving those sources an unfair advantage. To address this vulnerability, we require that incoming ϕ_{packet} weights of any given link be normalized, so that $\sum_{s\in S} \phi^{(s)}_k = 1$. We call this **source-weight normalization**.

$$\phi^{(src(p))}_{out_r(p)} := \frac{\phi^{(src(p))}_{out_r(p)}}{\sum_{s\in S} \phi^{(s)}_{out_r(p)}} \tag{5}$$

Source-weight normalization has two major benefits. First, it effectively guards against manipulation of the ϕ_{packet} weights reported in packet headers, since whatever weights come in, their sum will always be equal to one after the normalization step, and thus will not adversely affect the allocation of sources using other links. Second, equation 5 scales the weight of active sources proportionally so that they add up to one. If some sources are not using their allocated share of the link, source-weight normalization will scale the weights of all active sources proportionally to fill any unallocated portion.

Like in core-stateless Fair Queuing [21], we assume that a value (ϕ_{packet}) can be stored in the packet header. There are several locations where the value can be stored. The 8-bit Type-of-Service field in the IPv4 header is sufficient to approximate Justice, as discussed in [9]. Other possibilities include the ToS and Flow ID fields in IPv6. Finally, ϕ_{packet} can be communicated at the link-layer.

This may be the best solution, as it leaves the IP header unmodified throughout the network.

5 Analysis

Given the mechanism provided for calculating per-source weights, ϕ, it seems natural to ask what is the proper configuration of link weights, σ. In this section, we will use analysis to study the effect a given set of σ values and other factors will have on the throughput of a flow. We will also showcase how Justice can be used for traffic engineering. In the scenario we will discuss, a set of traffic sources distributed across a large network need to share a single bottleneck link. We will describe a general method for how to allocate link weights to achieve any desired allocation on the bottleneck link.

Previous sections have discussed only the computation and distribution of per-source weights. While these accurately describe the minimum weight of a source at a *given* link, they say nothing about the actual throughput that can be expected during end-to-end communication. Let us define C_k to be the bandwidth capacity of link k, and $\gamma_k^{(s)}$ to be the minimum achieved throughput of source s on the path to, and including, link k. Where the identity of the source is unambiguous, we write simply γ_k.

Given a source s, a path consisting of links $0, 1, \ldots, i$, and an initial sending rate of R, it is clear that on the first link $\gamma_0^{(s)} = min(R, \phi_0^{(s)} C_0)$, since $\phi_0^{(s)} C_0$ represents the minimum guaranteed bandwidth allocation on link 0. In general, however, γ_k also depends on the achieved bandwidth through the previous link, γ_{k-1}, or

$$\gamma_k = min(R, \gamma_{k-1}, \phi_k C_k) \tag{6}$$

Theorem 1. $\gamma_k^{(s)}$ *is the guaranteed minimum throughput that source s can expect to see across the sequence of links* $0, 1, \ldots, k$.

Proof. We will proceed by induction. Assume that the first link, link 0, is exclusively used by the source. [3] In the base case, γ_1, there is a single source using link 0, we have $\phi_1 = \sigma_1^{(0)}$. Weighted Fair Queuing (WFQ) provably distributes bandwidth according to the weights provided, so $\gamma_1 = min(R, \phi_1 C_1)$. For the inductive step, assume that γ_{k-1} holds. Accordingly, the input link $k-1$ serves a flow from source s with rate γ_{k-1}. Again, WFQ will serve the flow according to the weight given, ϕ_k in this case. If $\gamma_{k-1} \leq \phi_k C_k$, then WFQ will allocate s γ_{k-1} of bandwidth. Else, it will allocate $\phi_k C_k$, which completes the induction. QED

In most cases, the actual throughput will be considerably higher than the guaranteed minimum throughput. This is because in general, not all sources will be

[3] This assumption can be relaxed to include switched LANs. For shared medium physical layers, the hosts using it are considered one entity and bandwidth allocation has to be resolved at the MAC layer.

sending their entire σ_i of traffic across every link i. A characterization of the actual throughput received cannot be presented here, due to space constraints.

5.1 Traffic Engineering with Justice

In this section, we will showcase how Justice can be used to aid network traffic engineering. In the example below, we will discuss a scenario within a single AS, but Justice can be used equally well for inter-AS traffic engineering.

So far, we have assumed that a network administrator will provide each router with the necessary per-link weights. The per-link configuration regime provides a simple, but powerful tool for specifying the relative importance of *nearby* traffic sources. However, it may well be the case that the network is large, and that the relative importance of *distant* traffic sources needs to be addressed as well.

As a typical example, an Internet Service Provider (ISP) may have multiple access points, serving customers of multiple service classes, all connected to the same back-bone network. If possible, the ISP would like to ensure that all customers in the same service class receive approximately the same level of service, and that the various service classes exhibit a clear difference in performance. We will now describe a method for assigning per-link weights that provably achieves these goals.

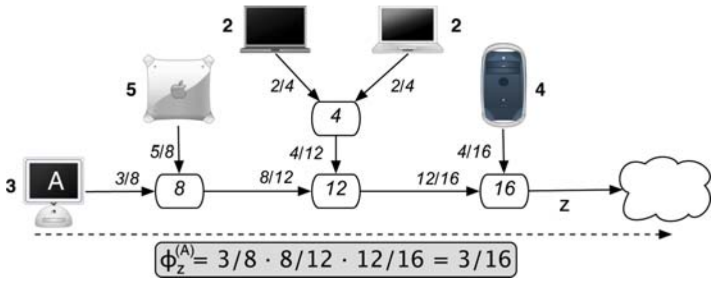


Fig. 3. Assignment of σ values based on prior entitlement information, shown next to each source. The allocation of any source at the bottleneck link will be directly proportional to its entitlement

Let us define *entitlement* η_s, to be an absolute number indicating the amount of the total network resources that source s is entitled to. We can think of η_s as an abstract expected "service level" of a given source. In Figure 3, every traffic source has been marked with an integer entitlement.

For this example, let us assume that all sources have a single bottleneck link, for example the link to the rest of the Internet. Assuming single-path routing, we can form a tree of routing paths, rooted at the gateway. Let us define the *aggregate entitlement* η_r of a router r in the tree to be

$$\eta_r = \sum_{i \in C} \eta_i \tag{7}$$

where C is the set of children of r. In Figure 3, the aggregate entitlement of each router is indicated. The optimal value of $\sigma_y^{(x)}$ can then be computed as

$$\sigma_y^{(x)} = \frac{\eta_n}{\eta_p}, \tag{8}$$

where x is the link between node n and its parent p, and y is the link leading from p toward the root. Figure 3 shows the σ according to Eq. 8 next to each link, and a sample ϕ_z for the traffic source marked "A". Note that the allocation $\phi_z^{(A)}$ equals 3/16, which is the entitlement of source "A", divided by the sum of entitlements in the entire network.

Theorem 2. *If aggregate entitlements and link shares are configured in accordance with Equations 7 and 8, each source s will receive a minimum guaranteed allocation at the bottleneck link b,*

$$\phi_b^{(s)} = \frac{\eta_s}{\sum_{i \in S} \eta_i}, \tag{9}$$

where S is the set of all traffic sources.

Proof. We will again use induction. Consider the base case, where N children are connected to a router r, which in turn is directly connected to the bottleneck link, b. For the base case, the set C in Eq. 7 contains all traffic sources, i.e., all traffic sources are children of r, or $C = S$. Thus, if we substitute Eq. 7 into the denominator of Eq. 8, we get

$$\sigma_b^{(s)} = \frac{\eta_s}{\sum_{i \in S} \eta_i}.$$

It remains only to compute $\phi_b^{(s)}$ from source s to link b which for the base case, according to Eq. 4, equals $1 \cdot \sigma_b^{(s)}$.

In the inductive step, assume that Eq. 9 holds for any tree of depth $\leq d$. We will now show that it holds for any tree T of depth $d+1$. Any tree of depth $d+1$ can be described as a root node r with j trees of depth $\leq d$ as its children. Let us consider a source s which belongs to one of the children of r, the subtree T_s.

We are interested in finding $\phi_b^{(s)}$, where b is the bottleneck link connected to the root node r. According to Eq. 4, this can be written as

$$\phi_b^{(s)} = \phi_{b'}^{(s)} \sigma_b^{(b')}$$

where b' is the link leading from T_s to the root node r. We know by our inductive assumption that

$$\phi_{b'}^{(s)} = \frac{\eta_s}{\sum_{i \in S_{T_s}} \eta_i},$$

where S_{T_s} is the set of all sources in T_s. Moreover, Eqs. 7 and 8 tell us that

$$\sigma_b^{(b')} = \frac{\eta_{T_s}}{\eta_T} = \frac{\sum_{i \in S_{T_s}} \eta_i}{\sum_{i \in S} \eta_i},$$

where S is the set of all sources in T. Finally, putting it all together, we have

$$\phi_b^{(s)} = \phi_{b'}^{(s)}\sigma_b^{(b')} = \frac{\eta_s}{\sum_{i \in S_{T_s}} \eta_i} \cdot \frac{\sum_{i \in S_{T_s}} \eta_i}{\sum_{i \in S} \eta_i} = \frac{\eta_s}{\sum_{i \in S} \eta_i}. \qquad QED$$

Although space does not permit a full discussion, we can offer an intuition on how to use the same framework to compute σ per-link weights from entitlement knowledge when there are more than one bottleneck links in the network. The idea is to form a separate tree for each bottleneck link, and compute the aggregate entitlements separately for each tree. Next, add up the aggregate entitlements out of each tree, for every router, to form the final aggregate entitlement values. Finally compute the σ values according to Eq. 8.

6 Simulation Results

In this section, we present a set of simulations that validate our claims about Justice as a bandwidth allocation scheme. Since the goal of Justice is not the same as that of, for example, per-conversation Fair Queuing, it is not possible to do a quantitative performance comparison. Instead, the purpose of the simulations is to show the behavior of Justice under realistic conditions, and to contrast and compare this with the behavior of previous queuing mechanisms.

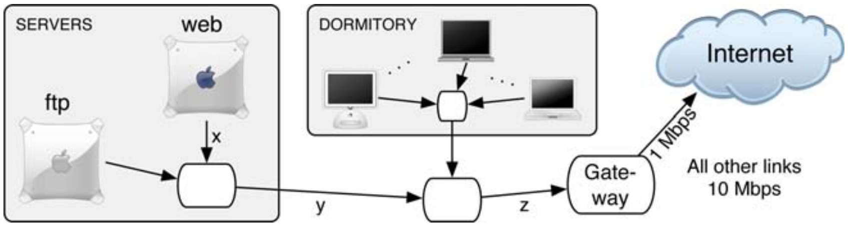
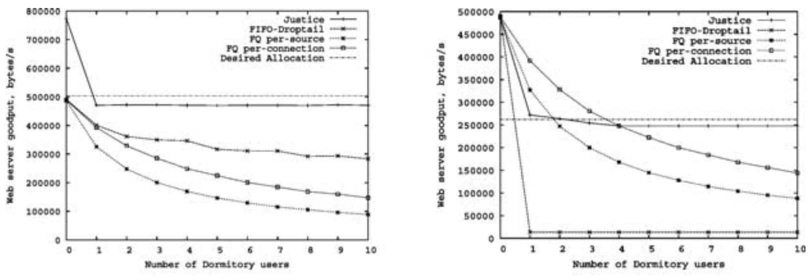


Fig. 4. University campus simulation topology. All intra-campus links are 10 Mbps, but the external link to the Internet is only 1 Mbps

We compare Justice with three other schemes, and the expected minimum allocation according to the analysis in Section 5. In all cases, we use the same queuing policy on all outgoing links on all routers. The current state of the Internet is best represented by the FIFO-Droptail policy, where packets are served in arrival order, and packets are dropped if they find the queue full on arrival. With per-source Fair Queuing, the router keeps a separate output queue per source, and thus ensures that all sources get the same allocation on the outgoing link. Per-flow Fair Queuing takes this one step further, and keeps a separate queue for each established connection, or source-destination pair. In our simulation setup, we assume that each established connection has a unique



(a) Scenario 1 (TCP) (b) Scenario 2 (UDP)

Fig. 5. Web server goodput under competition from dorm users. Only Justice provides the web server with its deserved bandwidth when there are many competing flows

remote end-point, so there is no distinction between connections and source-destination pairs.

We used the ns-2.27 simulator together with the built in implementation of DropTail queuing, and the CMU implementation of Worst-case Fair Weighted Fair Queuing (WF^{2}Q). We used the WF^{2}Q implementation to simulate per-source Fair Queuing, per-conversation Fair Queuing as well as Justice. For Justice, the ϕ values of all sources on all links were calculated using Eq. 2, and manually configured using WF^{2}Q settings. Our simulations were run on the topology shown in Figure 4. All intra-campus links are of 10 Mbps capacity, but the link to the Internet has a lower 1 Mbps capacity. The two servers each maintain 10 simultaneous TCP connections with end-points in the external Internet. The simulations stabilize quickly, and so it was sufficient to run each simulation for 10 seconds. All transmitted packets are of the same size, 1000 bytes. All results are averaged over 10 runs with randomized starting times for the connections. Except for the FIFO results, std. dev. was near zero.

In our first simulation scenario, results shown in Fig. 5 (a), there is between 0 and 10 dormitory traffic sources, each of which has 5 TCP connections to the Internet. The University network administrator has decided to allocate a minimum of 60% of the capacity on the bottleneck link to the server group, and ensures this by setting $\sigma_z^{(y)} = 0.6$. In addition, the administrator of the server group has decided to allocate 80% of his total capacity to the web server ($\sigma_y^{(x)} = 0.8$), and only 20% to the file server. Using Eq. 6, the expected share of the web server at the bottleneck link is $\gamma_z^{(web)} = 0.8 \cdot 0.6 \cdot 1Mbps = 0.48Mbps$. As expected, Justice effectively enforces the deserved allocation, in contrast with all other schemes which fail to preserve the allocation of the web server as the number of flows and users on the network increases.

Our second simulation scenario has dormitory users using UDP and sending 500 byte packets at a very high constant rate. In addition, the admins have set $\sigma_z^{(y)} = .5$ and $\sigma_y^{(x)} = .5$ which means the intended minimum allocation for the web server is $\gamma_z^{(web)} = 0.25Mbps$. As expected, the FIFO scenario performs very badly here, as allocation depends completely on the cooperativeness of end-hosts. Justice again enforces the allocation configured through the per-link weights. Note that it also enforces the 50% allocation of the dormitory when there is a small number of users, in this case protecting dorm users against the many flows of the web and ftp servers. Although the two Fair Queuing policies show a performance similar to what they showed in the TCP scenario, they still give the web server a progressively smaller share as the number of users in the dormitory increases.

7 Conclusion

In this paper we have presented Justice, a new take on bandwidth allocation. Justice is substantially different from previous approaches to bandwidth allocation. Justice provides a guaranteed minimum allocation to each source, at each link, independent of the behavior of other users. Moreover, Justice is flexible and allows network administrators to efficiently configure the *deserved* relative allocation of each host.

We argue that previously defined notions of per-flow fairness, and the algorithms that have been proposed to enforce them in the Internet, fall short in terms of protecting well-behaved users against their greedy counterparts. In contrast, if Justice is enforced in a network, users are guaranteed a certain level of service, independent of the actions of other users in the network.

We believe that Justice brings a radically new perspective on bandwidth allocation in the Internet. By focusing on per-source bandwidth allocation, and enforceability, Justice lays the foundation for an Internet with more predictable performance, better stability, and more flexibility.

References

1. John Nagle, "On packet switches with infinite storage," RFC 970, Dec 1985.
2. A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium proceedings on Communications architectures & protocols*. 1989, pp. 1–12, ACM Press.
3. V. Jacobson, "Congestion avoidance and control," *ACM Computer Communication Review*, vol. 18, 4, 1988.
4. Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM TON*, vol. 1, no. 4, 1993.
5. W. Feng, D. Kandlur, D. Saha, and Kang G. Shin, "BLUE: A new class of active queue management algorithms," Tech. Rep. CSE-TR-387-99, 15, 1999.
6. Dong Lin and Robert Morris, "Dynamics of random early detection," in *SIGCOMM '97*, Cannes, France, 1997.
7. Teunis J. Ott, T. V. Lakshman, and Larry H. Wong, "SRED: Stabilized RED," in *Proceedings of INFOCOM*, 1999, vol. 3.
8. Rong Pan, Balaji Prabhakar, and Konstantinos Psounis, "CHOKE, a stateless active queue management scheme for approximating fair bandwidth allocation," in *INFOCOM (2)*, 2000.

9. J. Eriksson, S. Krishnamurthy, and M. Faloutsos, "Justice: An enforceable alternative to fair bandwidth allocation," Tech. Rep., U. California, Riverside, 2004.
10. A. K. J. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case," IEEE/ACM TON, vol. 1, no. 3, 1993.
11. H. Zhang, "Wf2q: Worst-case fair weighted fair queueing," IEEE Infocom, 1996.
12. Z. Cao, Z. Wang, and E. Zegura, "Rainbow fair queuing: Fair bandwidth sharing without per-flow state," INFOCOM, 2000.
13. H. Zhu, A. Sang, and S.-Q. Li, "Weighted fair bandwidth sharing using scale technique," Computer Communications, vol. 24, 2001.
14. L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," ACM SIGCOMM 1990, pp. 19-29, 1990.
15. S. J. Golestani, "A self-clocked fair queueing scheme for broadband applications," IEEE Infocom 1994, 1994.
16. S. Suri, G. Varghese, and G. Chandranmenon, "Leap forward virtual clock:a new fair queueing scheme with guaranteed delays and throughput fairness," IEEE Infocom 1997, 1997.
17. M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," IEEE/ACM TON, vol. 4, no. 3, 1998.
18. D. Saha, S. Mukherjee, and S. Tripathi, "Carry-over round robin: A simple cell scheduling mechanism for atm networks," IEEE/ACM TON 6 (1998).
19. A. Banchs, "User fair queing: Fair allocation of bandwidth for users," in *Proceedings of INFOCOM*, 2002.
20. P. McKenney, "Stochastic fairness queuing," 1990.
21. I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: Achieving approximately fair bandwidth allocations in high speed networks," SIGCOMM, 1998.
22. Sally Floyd and Van Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM TON*, vol. 3, no. 4, 1995.
23. A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Symposium proceedings on Communications architectures & protocols*. 1989, ACM Press.
24. S. Blake et. al., "An architecture for differentiated services," RFC 2475, Dec 1998.
25. P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing.," In RFC 2267, January 1998.